CircuitTSAT: A Solver for Large Instances of the Disjunctive Temporal Problem

Abstract

In this paper, we report on a new solver for large instances of the *Disjunctive Temporal Problem* (DTP). Our solver is based primarily on the idea of employing "compact" circuit-based representations of disjunctive temporal constraints (akin to *ripple-carry adders* used in computer arithmetic operations). These circuit-based representations are in turn converted to CNF clauses of a SAT instance, and a powerful SAT solver is subsequently employed to efficiently solve the resulting SAT instance. We refer to this efficient DTP solver as "CircuitTSAT". A thorough empirical evaluation of CircuitTSAT shows that it significantly outperforms TSAT++ and Yices on a wide range of DTP instances. We also comment on the generality of our approach and its potential usefulness in dealing with more expressive constraints.

Introduction

Expressive and efficient temporal reasoning is central to many areas of Artificial Intelligence. Many tasks in planning and scheduling, for example, involve reasoning about temporal constraints between actions and propositions in partial plans (Smith, Frank, and Jónsson 2000). These tasks may include *threat resolution* between actions in partial order planning and analyzing *resource consumption envelopes* to guide the search for a good plan (Kumar 2003). Among the important formalisms used for reasoning with metric time are *Simple Temporal Problems* (STPs) (Dechter, Meiri, and Pearl 1991) and *Disjunctive Temporal Problems* (DTPs) (Stergiou and Koubarakis 1998).

Unlike DTPs, STPs can be solved in polynomial time, but are not as expressive as DTPs. An STP is characterized by a graph $G = \langle \mathcal{X}, \mathcal{E} \rangle$, where $\mathcal{X} = \{X_0, X_1 \ldots X_N\}$ is a set of events (X_0 is the "beginning of the world" node and is set to 0 by convention) and $e = \langle X_i, X_j \rangle \in \mathcal{E}$, annotated with the bounds $[LB(e), UB(e)]$, is a *simple temporal constraint* (STC) between X_i and X_j indicating that X_j must be scheduled between $LB(e)$ and $UB(e)$ time units after X_i is scheduled ($LB(e) \leq UB(e)$). DTPs are significantly more expressive than STPs, and allow for disjunctive constraints. The general form of a DTP is as follows. We are given a set of events $\mathcal{X} = \{X_0, X_1 \ldots X_N\}$ and a set of constraints $\mathcal{C}$. A *disjunctive temporal constraint* (DTC) $c_i \in \mathcal{C}$ is a disjunction of the form $s_{(i,1)} \vee s_{(i,2)} \vee \ldots \vee s_{(i,T_i)}$. Here, $s_{(i,j)}$ $(1 \leq j \leq T_i)$ is an STC of the form $L_{(i,j)} \leq X_{b_{(i,j)}} - X_{a_{(i,j)}} \leq U_{(i,j)}$ for $a_{(i,j)}, b_{(i,j)} \in \{0, 1 \ldots N\}$.

Although DTPs are expressive enough to capture many important tasks in planning and scheduling, they require an exponential search space. The principal approach taken to solve DTPs has been to convert the original problem to one of selecting a disjunct from each constraint, and then checking that the set of selected disjuncts forms a consistent STP. Checking the consistency of, and finding a solution to an STP can be performed in polynomial time using shortest path computations (Dechter, Meiri, and Pearl 1991). However, the computational complexity of solving a DTP comes from the fact that there are an exponentially large number of disjunct combinations possible.

The above "disjunct selection problem" can in fact also be cast as a *Constraint Satisfaction Problem* (CSP) or a SAT problem, and subsequently solved using standard search techniques applicable to them. In the first case, a meta-level variable is associated with each clause, and the domain of this variable is just the set of disjuncts in that clause. We then seek an assignment to the meta-level variables that induces a consistent STP.[1] The EPILITIS system (Tsamardinos and Pollack 2003), for example, employs standard CSP search techniques like *conflict-directed backjumping* and *nogood recording* for pruning the search space. In the SAT-based approach, a DTP is abstracted into a propositional formula obtained by substituting each distinct binary difference constraint with a new propositional atom. Only those assignments that satisfy the SAT instance are first generated, and later checked for the consistency of the induced STP. The TSAT++ solver (Armando, Castellini, and Giunchiglia 2000; Armando et al 2004), for example, successfully incorporates many techniques akin to those used in modern SAT solvers (Moskewicz et al 2001). It employs preprocessing strategies, look-ahead and look-back techniques, and branching rules that enable TSAT++ to significantly outperform its previous competitors on a wide variety of DTP instances (Armando et al 2004).

Although TSAT++ employs a wide range of techniques akin to the ones used in many state-of-the-art SAT solvers, it does not provide a framework for directly employing a SAT solver. In fact, both CSP-based approaches and SAT-based

[1] A final solution is obtained by solving the consistent STP using shortest path computations.

approaches repeatedly use the Bellman-Ford algorithm for checking the consistency of the induced STPs (under different proposed meta-level assignments). In this paper, we report on a new solver for large instances of the DTP. Our solver (referred to as "CircuitTSAT") is based primarily on the idea of employing "compact" circuit-based representations of DTCs (akin to *ripple-carry adders* used in computer arithmetic operations). These circuit-based representations are in turn converted to CNF clauses of a SAT instance, and a powerful SAT solver is subsequently employed to efficiently solve the resulting SAT instance (thereby providing a framework for directly employing a state-of-the-art SAT solver). A thorough empirical evaluation of CircuitTSAT shows that it significantly outperforms TSAT++ on a wide range of DTP instances.

It is also imperative for us to compare the performance of CircuitTSAT with that of Yices (a powerful SMT solver). An SMT (*satisfiability modulo theories*) problem instance is a generalization of a Boolean SAT instance in which the Boolean variables are replaced by binary-valued functions of suitable domain-specific variables (referred to as *predicates*). For example, the predicates in a DTP instance are simple difference inequalities. Yices (Dutertre and de Moura 2006) is a state-of-the-art SMT solver[2] that decides the satisfiability of formulas containing uninterpreted function symbols with equality, linear real/integer arithmetic, recursive datatypes, etc. Because of its generality, Yices can also be used to solve DTP instances. However, as we will show in this paper, CircuitTSAT significantly outperforms Yices on a wide range of DTP instances. This suggests that although Yices is a powerful state-of-the-art solver suitable for general SMT instances, it does not match the performance of CircuitTSAT on the more structured DTP instances.

It is worth noting that early attempts for solving SMT instances were in fact based on the idea of translating them to Boolean SAT instances and solving them using a powerful SAT solver; e.g., (Bryant, Lahiri, and Seshia 2002). However, this approach was superseded by solvers (like Yices) that tightly integrated the Boolean reasoning with theory-specific solvers. While the use of theory-specific solvers is certainly more productive than compilation to SAT instances in the general context of solving SMTs, our work suggests that it is not necessarily so for certain important combinatorial problems like DTPs. In fact, as mentioned before, it is highly inefficient for us to repeatedly employ the Bellman-Ford algorithm (in a search tree of exponential size) for solving large instances of the DTP. Our work studies and evaluates the direct SAT-based encodings for DTP instances using a circuit-based representation. The empirical success of CircuitTSAT (over TSAT++ and Yices) is indicative of the potential value in revisiting such SAT-based encodings/procedures for solving certain specific kinds of combinatorial problems. In this regard, we briefly comment on the extent of our approach and its potential usefulness in dealing with more expressive kinds of constraints.

[2]Yices won the SMT-COMP'06 competition in all eleven divisions, and won in seven of twelve divisions in the SMT-COMP'07 competition.

Background and Notation

We now introduce definitions and notations to be used throughout the paper. We also comment on some complexity results and/or algorithms associated with solving STPs and DTPs. We refer to a difference inequality of the form $X_j - X_i \leq z$ as an STC, and denote it by $\mathcal{S}_{i,j}(z)$. A DTC $\mathcal{D}$ and a DTP $\mathcal{P}$ can then be expressed as follows:

$$\mathcal{D}_{\Lambda_\ell} = \bigvee_{(\lambda_i,\gamma_i,z_i)\in\Lambda_\ell} \mathcal{S}_{\gamma_i,\lambda_i}(z_i) \qquad \mathcal{P} = \bigwedge_{\ell=1}^{M} \mathcal{D}_{\Lambda_\ell} \ . \qquad (1)$$

Here, Λ_ℓ is a set containing triplets of the form $(\lambda_i, \gamma_i, z_i)$; the λ_i's and γ_i's are indices, and the z_i's are constants. The number of triplets in Λ_ℓ is denoted by $K_\ell = |\Lambda_\ell|$, and $K = \max_\ell K_\ell$. We note that this variant of DTPs is equally general and is the same as the form of DTPs used in (Armando, Castellini, and Giunchiglia 2000).

It is also well known that a conjunction of STCs can be efficiently solved using shortest path computations on a directed graph. Central to this algorithm is the notion of a *distance graph* $D(G)$ associated with an STP $G = \langle \mathcal{X}, \mathcal{E} \rangle$. An edge $\langle X_i, X_j \rangle$ in the distance graph is annotated with a real number z, and encodes the constraint $X_j - X_i \leq z$. It can be shown that a consistent schedule exists for $X_0, X_1 \ldots X_N$ in $G = \langle \mathcal{X}, \mathcal{E} \rangle$ if and only if the distance graph $D(G)$ does not contain any negative cost cycles (Dechter, Meiri, and Pearl 1991). On the other hand, DTPs are NP-hard to solve in general — although rich tractable classes have been identified in (Kumar 2005) and (Kumar 2006).

Circuit-based Representation of DTCs

Given a DTP, there are many ways to cast it as a SAT instance. The most naïve way to do this is to first discretize the domain of each real-valued variable, and then introduce a Boolean variable for each landmark. This method, however, requires the absolute value of any variable to be bounded by some positive number (Q). It also requires an appropriate scale of discretization (s). If there are N variables in the DTP, the resulting SAT instance would contain an unwieldy NQ/s number of Boolean variables. Further, each constraint by itself would have a complicated representation, making this approach highly undesirable.

A second method to represent a DTP as a SAT or a CSP instance is akin to that used in popular systems like EPILITIS or TSAT++. Here, the DTP is viewed as a "disjunct selection problem," and meta-level variables are used to represent commitments to different disjuncts in each clause. Although the number of meta-level variables and the sizes of their respective domains are manageable, the constraints are no longer explicit. Instead, the constraints are implicitly characterized by the presence/absence of negative cost cycles in the induced distance graphs. Further, it is expensive to explicitly enumerate all constraints before search is carried out as there may be an exponential number of them.

A third approach that we propose in this paper is to leverage the structure of the DTCs — namely, the fact that they have "compact" circuit-based representations. In particular, we represent the value of any variable in a binary format,

and use Boolean variables corresponding to each bit position in this representation. If Q is the largest value that any variable can take, the total number of Boolean variables required would only be $N\lceil \log_2 Q \rceil$. Further, we show that each DTC can be represented "compactly" using a circuit that, in turn, can be encoded as a SAT instance with the use of a "small" number of auxiliary variables. This leads us to a representation scheme where both the number of variables and the number of constraints are manageable. The constraints have simple representations — making our approach not only scalable to large DTP instances, but also amenable to an off-the-shelf SAT solver.

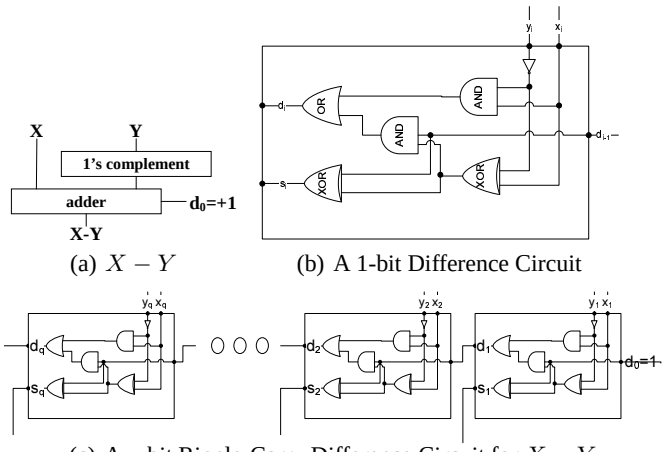
Formulating a DTP directly as a SAT instance has the following advantages. First, modern SAT solving techniques can be directly employed for solving DTPs. Any new techniques developed for bettering SAT solvers would bear immediate and direct implications on the efficiency of solving DTPs — hence obviating the need to redesign our DTP solver and/or implement the analogues of such new techniques especially for DTPs. Second, by decreasing the number of bits used to represent the possible values of a variable, we can trade-off various aspects of the problem's complexity; for example, we can simplify the search space by compromising on the precision of the numerical constants used in the given DTP instance. We note once again that although the idea of employing circuit-based SAT encodings of constraints may not outperform powerful solvers like Yices on general SMT instances, it can exploit additional structure in more specific combinatorial problems like DTPs.

In this section, we will primarily address DTPs with non-negative integral temporal variables. Later, we will discuss how our framework can be extended easily to include negative integers by a mechanism that simply translates the upper and lower bounds on the variables. We will also discuss how it can be extended further to include floating-point numbers as well by using a simple scaling mechanism.

Representing STCs using Circuits:

Consider two non-negative integral variables X and Y whose values can be represented in a binary format using a sequence of q bits (logical variables) each; i.e., $X = \langle x_1, x_2 \ldots x_q \rangle$ and $Y = \langle y_1, y_2 \ldots y_q \rangle$. As depicted in Figure 1(a), the binary representation of $X - Y$ in *2's complement notation*[3] can be found by simply negating the bits of Y and using an *adder circuit* with an initial carry bit $d_0 = 1$.[4] The composite circuit calculates the difference between X and Y "logically," and more importantly, it computes the sign of the difference as its final carry-bit d_q. Thus, d_q is a propositional variable equivalent to $X - Y \geq 0$.

In order to calculate the required carry-bit, we need to look at the circuitry of the adder. Figure 1(c) shows a simple cascading chain of full adders capable of computing the difference and the final carry-bit of two q-bit numbers (Heuring, Jordan, and Murdocca 1997). This configuration is commonly known as a *ripple-carry adder*, and the underlying units — as shown in Figure 1(b) — are referred to as 1-bit *full adders*. We note that the bits of Y are negated explicitly in this difference circuit since the only operation of interest is the difference operation. More complicated circuits could be used to directly allow for negative integers or floating-point difference computations; but this complicates the resulting inequality test. By considering only non-negative integers, we are assured that no overflow is possible in computing the difference $X - Y$. Thus, the inequality test only concerns the final carry-bit d_q rather than a more complicated test. Further, as mentioned before, translation/scaling techniques can also be used to deal with DTPs on negative/floating-point numbers.



(a) $X - Y$ (b) A 1-bit Difference Circuit

(c) A q-bit Ripple-Carry Difference Circuit for $X - Y$

Figure 1: The above diagrams illustrate a difference circuit. Figure 1(a) shows that an adder circuit suffices to compute the difference $X - Y$. By merging the 1's complement operation (negation) with the adder, the full adder in Figure 1(b) computes bitwise differences. Chaining these together produces a difference circuit in Figure 1(c) that computes the sum and carry bits, s_i and d_i. The last carry-bit d_q indicates the sign of the difference: $d_q \Leftrightarrow (X - Y \geq 0)$. These circuit designs are found in (Heuring, Jordan, and Murdocca 1997).

While alternative adder circuits also exist (e.g., the *carry lookahead adder*), the ripple-carry adder exhibits a simple recursive nature. To better understand the difference computation, we examine a single adder. As Figure 1(b) shows, the 1-bit full adder produces a sum-bit s_i and a carry-bit d_i expressed logically as:

$$s_i = d_{i-1} \oplus (x_i \oplus \bar{y}_i) \quad (2)$$

$$d_i = [(x_i \oplus \bar{y}_i) \wedge d_{i-1}] \vee (x_i \wedge \bar{y}_i)$$
$$= [(x_i \vee \bar{y}_i) \wedge d_{i-1}] \vee (x_i \wedge \bar{y}_i) \ . \quad (3)$$

As previously noted, only the final carry-bit is relevant to determine whether $X - Y \geq 0$. This leads us to attempt expanding Eq. 3 explicitly in terms of the input bits $\langle x_1, x_2, \ldots, x_q \rangle$ and $\langle y_1, y_2, \ldots, y_q \rangle$. In fact, as Eq. 3 shows, the recursion for d_i is a first-order recursion that can

[3] This is a common representation for integers in which the most significant bit represents the sign. For negative integers, inverting the bits and adding 1 yields its absolute value. It can be shown that many arithmetic circuits are simplified by using this convention.

[4] For addition, we use $d_0 = 0$, and we do not negate Y.

be unwrapped as follows:

$$d_q = \bigvee_{j=0}^{q} \left[(x_j \wedge \bar{y}_j) \wedge \bigwedge_{i=j+1}^{q} (x_i \vee \bar{y}_i) \right] \ .$$

However, further simplification of this expression seems prohibitive — thereby warranting a different approach.

Converting STCs to CNF Efficiently:

We will now show how to propositionally represent an STC using auxiliary-variables. This method will be referred to as the *auxiliary-variable approach*. We begin by defining d_q to be $d_q \Leftrightarrow (X - Y \geq 0)$. We know that Eq. 3 yields constraints of the form

$$d_i \Leftrightarrow [(x_i \vee \bar{y}_i) \wedge d_{i-1}] \vee (x_i \wedge \bar{y}_i) \ , \qquad (4)$$

that in turn can be translated into a set of clauses. Recursively considering d_{i-1}, every difference inequality requires the addition of q auxiliary-variables and $O(q)$ clauses into the SAT formulation. As will be shown, the resulting SAT problems are still quite manageable for modern SAT solvers.

Using the auxiliary-variable logic formalism for expressing $X - Y \geq 0$, we can build the logically equivalent form of an STC "$Y - X \leq z$" for non-negative integers X and Y, and a (signed)-integer constant z. This is done by first rewriting the STC as $(X - Y) + z \geq 0$, and then evaluating it in two phases: (1) expressing $X - Y$ propositionally, and (2) adding the constant z to the result. The sum and carry bits of $(X - Y)$ are denoted by s_i and d_i respectively, and the corresponding bits of $(X - Y) + z$ are denoted by r_i and c_i respectively. While the propositional formulas for s_i and d_i are expressed in Eqs. 2 and 3 with $d_0 = 1$, the formulas for r_i and c_i are more complex since one of the arguments is actually the result of the difference $X - Y$. We must also allow for the constant z to be *any* signed integer. The carry-bits are given by:[5]

$$\begin{aligned} c_i &= \begin{cases} ((s_i \vee \bar{z}_i) \wedge c_{i-1}) \vee (s_i \wedge \bar{z}_i) & \text{if } z \leq 0 \\ ((s_i \vee z_i) \wedge c_{i-1}) \vee (s_i \wedge z_i) & \text{if } z > 0 \end{cases} \\ &= \begin{cases} c_{i-1} \vee s_i & \text{if cond}_A \\ c_{i-1} \wedge s_i & \text{if cond}_B \end{cases} \ . \end{aligned} \qquad (5)$$

Here we have $c_0 = 1$ for $z \leq 0$ and $c_0 = 0$ for $z > 0$, and the conditions cond_A and cond_B are given by

$$\text{cond}_A = [(z \leq 0) \wedge (z_i = 0)] \vee [(z > 0) \wedge (z_i = 1)] \qquad (6)$$
$$\text{cond}_B = [(z \leq 0) \wedge (z_i = 1)] \vee [(z > 0) \wedge (z_i = 0)] \qquad (7)$$

The simplified form in Eq. 5 is possible because the z_i bits are constants for any STC — enabling us to compile a set of customized clauses for each STC based on the value of z. As we build the CNF for each STC, we can test for conditions cond_A and cond_B, adding the customized clauses for the STC's constant.

[5]Computing the sum-bits r_i is unnecessary since we are only concerned with the sign of the expression.

Converting Constraints to CNF: The constraints for the auxiliary-variables d_i and c_i are converted into clauses. We build our CNF clauses from the truth table for the logical functions described in Eqs. 4 and 5. The CNF equivalent for each d_i constraint is:

$$(d_i \vee d_{i-1} \vee \bar{x}_i \vee y_i) \wedge (d_i \vee \bar{d}_{i-1} \vee y_i) \wedge (d_i \vee \bar{d}_{i-1} \vee \bar{x}_i)$$
$$(\bar{d}_i \vee \bar{d}_{i-1} \vee x_i \vee \bar{y}_i) \wedge (\bar{d}_i \vee d_{i-1} \vee \bar{y}_i) \wedge (\bar{d}_i \vee d_{i-1} \vee x_i)$$

Similarly, the CNF for each c_i depends on the conditions in Eqs. 6 and 7 listed below:

condition A	**condition B**
$c_i \vee \bar{c}_{i-1}$	$\bar{c}_i \vee c_{i-1}$
$c_i \vee c_{i-1} \vee d_{i-1} \vee x_i \vee y_i$	$c_i \vee \bar{c}_{i-1} \vee d_{i-1} \vee x_i \vee y_i$
$c_i \vee c_{i-1} \vee d_{i-1} \vee \bar{x}_i \vee \bar{y}_i$	$c_i \vee \bar{c}_{i-1} \vee d_{i-1} \vee \bar{x}_i \vee \bar{y}_i$
$c_i \vee c_{i-1} \vee \bar{d}_{i-1} \vee \bar{x}_i \vee y_i$	$c_i \vee \bar{c}_{i-1} \vee \bar{d}_{i-1} \vee \bar{x}_i \vee y_i$
$c_i \vee c_{i-1} \vee \bar{d}_{i-1} \vee x_i \vee \bar{y}_i$	$c_i \vee \bar{c}_{i-1} \vee \bar{d}_{i-1} \vee x_i \vee \bar{y}_i$
$\bar{c}_i \vee c_{i-1} \vee d_{i-1} \vee \bar{x}_i \vee y_i$	$\bar{c}_i \vee \bar{c}_{i-1} \vee d_{i-1} \vee \bar{x}_i \vee y_i$
$\bar{c}_i \vee c_{i-1} \vee d_{i-1} \vee x_i \vee \bar{y}_i$	$\bar{c}_i \vee \bar{c}_{i-1} \vee d_{i-1} \vee x_i \vee \bar{y}_i$
$\bar{c}_i \vee c_{i-1} \vee \bar{d}_{i-1} \vee x_i \vee y_i$	$\bar{c}_i \vee \bar{c}_{i-1} \vee \bar{d}_{i-1} \vee x_i \vee y_i$
$\bar{c}_i \vee c_{i-1} \vee \bar{d}_{i-1} \vee \bar{x}_i \vee \bar{y}_i$	$\bar{c}_i \vee \bar{c}_{i-1} \vee \bar{d}_{i-1} \vee \bar{x}_i \vee \bar{y}_i$

The CNFs for the d_i and c_i constraints allow us to build auxiliary-variables equivalent to any STC of the form $X - Y + z \geq 0$. However, the logical expression for an STC is more complex than the simple difference $X - Y$ since the combined operation $(X - Y) + z$ could overflow. Nonetheless, $(X - Y) + z \geq 0$ can be logically expressed in terms of the carry-bits d_q and c_q. For the case that $z \leq 0$, $d_q = 0$ implies that $X - Y < 0$, and regardless of c_q, we have $(X - Y) + z < 0$. Otherwise, $d_q = 1$ implies that $X - Y \geq 0$ and that adding the non-positive z cannot cause an overflow. Thus, the bit c_q correctly identifies the sign of $X - Y + z$, and the required test is given by

$$(X - Y + z \geq 0) \Leftrightarrow (d_q \wedge c_q) \ . \qquad (8)$$

Similarly, for $z > 0$, $d_q = 1$ is sufficient since it implies $X - Y \geq 0$, and thus $X - Y + z \geq 0$. If instead, $d_q = 0$, then we cannot have an overflow since $X - Y < 0$, and thus having c_q implies that $X - Y + z \geq 0$. For $z > 0$, therefore, the constraint for the STC becomes

$$(X - Y + z \geq 0) \Leftrightarrow (d_q \vee c_q) \ . \qquad (9)$$

We have now completely described the mechanism for transforming an STC into CNF. Eqs. 8 and 9 give logical conditions for testing whether an STC is satisfied, and Eqs. 4 and 5 describe the constraints on the auxiliary-variables.

Transforming the DTCs:

Since the STCs are themselves disjuncts in the DTCs, our last task is to replace these disjuncts with their propositional forms provided in Eqs. 8 and 9. In some cases, this substitution directly yields clauses, but the resulting substitution often yields complex logical expressions due to the conjunction in Eq. 8. Rather than expanding them into CNF directly,

we introduce another set of auxiliary-variables a_w. These variables occur when $z \leq 0$, and are defined as:

$$a_w \;\Leftrightarrow\; c_{q,w} \wedge d_{q,w} \;\; . \tag{10}$$

a_w represents the w-th STC of the disjunction, and $d_{q,w}$ and $c_{q,w}$ are the q-th carry-bits of the w-th STC. These constraints are easily translated into CNF clauses. Similar auxiliary-variables can be used when $z > 0$, but are not necessary since the STC from Eq. 9 is already characterized by a disjunction. Finally, we can express the DTC as a disjunction of literals: $a_1 \vee a_2 \vee \ldots \vee a_w$.

Choosing the Bit-Space for CircuitTSAT:

We note that in CircuitTSAT, a DTP with parameters N, K and M is converted into a SAT instance that has $O(qN + qKM)$ variables and $O(qKM)$ clauses (where q is the number of bits representing each temporal variable's value). One of the issues with CircuitTSAT is that it is non-trivial to obtain a tight bound on q. If we choose q to be too small, there may exist solutions to a given DTP that cannot be represented within the bit-space. In such a case, CircuitTSAT can report the absence of a solution even when one actually exists; we refer to this as a *miscue*. However, small values of q reduce the size of the CNF and may result in faster execution times of the SAT solver.

A loose upper bound on q for non-negative integer DTPs is given by

$$q_{max} = \left\lceil \log_2 \sum_{i=1}^{M} \max_{(\lambda_j,\gamma_j,z_j)\in\Lambda_i} |z_j| \right\rceil \;\; . \tag{11}$$

This bound is based on the fact that, in the worst case, the largest constant from each DTC would be introduced into the distance graph. Although q_{max} is a sound upper bound on the number of bits required, many other heuristically chosen values of q may perform better in practice.

Experimental Results and Comparisons

In this section, we provide an empirical evaluation of CircuitTSAT, TSAT++ and Yices on a wide range of DTP instances. These DTP instances were generated randomly using the "random DTP generation model" described in (Armando, Castellini, and Giunchiglia 2000; Armando et al 2004). The parameters used in this process are K: the number of disjuncts per DTC, N: the number of temporal variables in the DTP, M: the number of DTCs in the DTP, and L: the maximum absolute value of any constant in the DTP. An exact description of the "random DTP generation model" is provided later in this section.

To compare the efficiency and scalability of the different solvers, we tested them on larger values of K and N than those used in (Armando, Castellini, and Giunchiglia 2000; Armando et al 2004). In particular, we used $K \in \{3, 5, 7\}$, $N \in \{50, 100, 150, 200\}$, $M/N \in \{2, 4, 6, 8, 10, 12, 14\}$, and $L = 100$. We generated 50 DTP instances for each combination of values for the above parameters, and we ran CircuitTSAT, TSAT++, and Yices on these instances. All trials were given at most 10 minutes to complete. All experiments were run on a single machine with a 2.8 GHz Pentium 4 processor and 2 GB of RAM. All solvers were timed externally to ensure fairness.

Other solver-specific decisions were made to ensure fairness. For TSAT++, we enabled *early pruning*, *IS(2) preprocessing*, *triggering optimization*, and *shortest reason detection*.[6] For Yices, we used two different variants. In the first version, the temporal variables were specified to be real-valued. In the second version, they were specified as integers. However we did not find any significant difference in the performances of these variants; and for the purposes of this paper, we report only on the integer-variant of Yices. Finally, for CircuitTSAT, we used the JeruSAT SAT solver (Nadel 2002) and we used $q = q_{max}$.

We also report on the performance of other variants of CircuitTSAT. In particular, we computed $q_{min} = \lceil \log_2 L \rceil$, and designed several "bit-variants" of CircuitTSAT that progressively employed q_{min}, $q_{min} + 1$, $q_{min} + 2 \ldots q_{max}$ bits. The corresponding CircuitTSAT solvers are referred to as $\text{CTSAT}_{\text{min}}$, $\text{CTSAT}_{\text{+1}}$, $\text{CTSAT}_{\text{+2}} \ldots \text{CTSAT}_{\text{max}}$. However, unless otherwise stated, CircuitTSAT refers to the $\text{CTSAT}_{\text{max}}$ variant of our algorithm.

Generating Random DTPs: Random instances of the DTP were generated according to the model introduced in (Stergiou and Koubarakis 1998; 2000). However, as in (Armando, Castellini, and Giunchiglia 2000; Armando et al 2004), we used M/N — instead of M — as a parameter in the generation process, and as with previous studies, our experiments only generated integer constants for the STCs. The DTCs were generated by the following process:

for $\ell = 1$ to K **do**
  Choose distinct i and j uniformly from the n variables.
  Draw an integer z uniformly from the interval $[-L, L]$.
  The ℓ-th disjunct in this DTC is set to be "$X_j - X_i \leq z$".
end for
if any pair of disjuncts are identical **then**
  Discard this DTC.
end if

This process is repeated until M DTCs have been generated.

Results and Analysis:

Figure 2 shows graphical comparisons of the performances of CircuitTSAT, TSAT++, and Yices. The three graphs correspond to $K = 3$, $K = 5$, and $K = 7$ respectively. In Figure 2(a) (where $K = 3$), we observe that Yices exhibits the best overall performance for all values of N. However, the margin between the performances of CircuitTSAT and Yices decreases with increasing N. Further, in Figure 2(b) (where $K = 5$), we observe that while CircuitTSAT scales very well with increasing N, both TSAT++ and Yices fail to match its superior performance. In fact, when $N \geq 100$, CircuitTSAT significantly outperforms both TSAT++ and

[6] As suggested by the authors of TSAT++ (in personal communication) for its best possible performance. For further details on these features, see (Armando, Castellini, and Giunchiglia 2000; Armando et al 2004).

Yices. Finally, Figure 2(c) demonstrates further disparity between CircuitTSAT and the other solvers for $K = 7$. Here, the running time of CircuitTSAT is less than that of TSAT++ and Yices by roughly an order of magnitude for $N \geq 100$.

We now describe several performance trends with increasing values of K and N. Firstly, for fixed K, increasing the value of N has more impact on the running times of TSAT++ and Yices than on CircuitTSAT. CircuitTSAT consistently takes only 3–7 times more CPU time to solve DTP instances with increasing $N \in \{50, 100, 150, 200\}$ for all values of K. On the other hand, in several regions, the running times of TSAT++ and Yices increase by more than an order of magnitude for increasing $N \in \{50, 100, 150, 200\}$. Secondly, as K increases, the performance of CircuitTSAT improves dramatically. This is in sharp contrast to the behavior of TSAT++ and Yices. While TSAT++ and Yices benefit from larger K when $N \leq 100$, their performance stagnates and even deteriorates for $N > 100$. We observed a third trend in the percentage of problem instances for which the solvers timed out. These statistics are provided in Table 1. As K increases, CircuitTSAT times out on a smaller percentage of instances than the other solvers. For example, when $K = 7$ and $N = 200$, CircuitTSAT finds solutions to all the problem instances while TSAT++ and Yices time out on 65.9% and 80.6% of the instances respectively. Consistent with our previous observation, the table shows that while the percentage of time-outs for TSAT++ and Yices increases with K, it actually decreases for CircuitTSAT.

We attribute the superior scalability of CircuitTSAT to the following brief explanation. In TSAT++ and Yices, as K increases, the number of Boolean variables created for the predicates also increases. This causes a combinatorial explosion in the search tree, thereby causing the performances of these solvers to deteriorate. In CircuitTSAT, the number of Boolean variables in the SAT encoding also increases. However, fixing the bits that represent the original temporal variables determines the values of all other auxiliary variables (by unit propagation). Since the number of these "input" bits doesn't change with K, adding disjuncts to each DTC actually serves as a relaxation for the search space.

Variants of CircuitTSAT:

To further investigate CircuitTSAT, we performed additional experiments to compare different variants of it. We used the different bit-variants of CircuitTSAT in combination with the JeruSAT/MiniSAT SAT solvers (Nadel 2002; Eén and Sörensson 2003). We ran experiments with parameters $K \in \{2, 3, 5\}$, $N \in \{50, 100, 150, 200\}$, $M/N \in \{2, 3, 4, 5, 6, 8\}$, and $L = 100$. Figure 3 shows graphical comparisons of the relative performances of these variants as well as the performances of TSAT++ and Yices for $N = 150$. Firstly, we observed that the MiniSAT versions were outperformed by the JeruSAT version. Secondly, consistent with our previous observations, CircuitTSAT (with the JeruSAT solver) outperforms TSAT++ and Yices for larger values of K. Further, we notice that the performance gap between the *min* and *max* versions of CircuitTSAT significantly decreases with increasing K. Upon further examination, we also found that the remarkable performance of the *min* version for $K = 2$ resulted from a high percentage of miscued problem instances by the solver. For higher values of K, no problem instances were miscued even by the *min* version. This is indicative of a relaxation in the search space for increasing K as previously noted.

	CircuitTSAT	TSAT++	Yices
K = 3			
N = 50	66.9% 0% 33.1%	68.1% 0% 31.9%	86.3% 0% 13.7%
N = 100	49.4% 0% 50.6%	45.0% 0% 55.0%	67.7% 0% 32.3%
N = 150	40.3% 0% 59.7%	35.2% 0% 64.8%	57.4% 0% 42.6%
N = 200	30.9% 0% 69.1%	28.6% 0% 71.4%	50.9% 0% 49.1%
K = 5			
N = 50	100% 0% 0%	100% 0% 0%	100% 0% 0%
N = 100	99.7% 0% 0.3%	68.1% 0% 31.9%	100% 0% 0%
N = 150	98.6% 0% 1.4%	41.8% 0% 58.2%	36.9% 0% 63.1%
N = 200	92.0% 0% 8.0%	30.8% 0% 69.2%	14.6% 0% 85.4%
K = 7			
N = 50	100% 0% 0%	100% 0% 0%	100% 0% 0%
N = 100	100% 0% 0%	89.0% 0% 11.0%	99.1% 0% 0.9%
N = 150	100% 0% 0%	51.7% 0% 48.3%	34.6% 0% 65.4%
N = 200	100% 0% 0%	34.1% 0% 65.9%	19.4% 0% 80.6%

Table 1: This table shows the percentage of problem instances solved by CircuitTSAT, TSAT++, and Yices in terms of the number of disjuncts per clause ($K = 3, 5, 7$) and the number of temporal variables in the problem instance ($N = 50, 100, 150, 200$). For each setting we show a triplet (a, b, c) where a, b, and c are the percentages of instances the algorithm (a) found a solution to, (b) found that no solution existed, and (c) timed-out on.

Discussions and Future Work

Several immediate improvements can be incorporated into future versions of CircuitTSAT. First, preprocessing steps similar to those in (Armando et al 2004) can be integrated into our approach. Unit DTCs (i.e., STCs) can also be precompiled to reflect the implications of the known distance graph. Second, better strategies for choosing q — possibly even allowing distinct variables to have different bit-lengths — could significantly boost the performance of CircuitTSAT (especially in dealing with floating-point numbers). Third, post-processing steps can be implemented to retrieve a consistent STP from any solution to the SAT instance. Such an STP represents a class of solutions to the given DTP — thereby giving CircuitTSAT the same representational benefits as the "disjunct selection"-based approaches.

The success of our approach also has implications in other application domains. For one, our method can naturally incorporate additional propositional variables in the system. This suggests that "hybrid" constraints — involving both propositional and temporal constraints — would also be readily amenable to our techniques. In turn, solving these kinds of hybrid constraints opens up the possibility of a number of applications of our techniques in planning and scheduling domains — where causal interactions between actions together with the underlying temporal and resource contentions between them can be cast as hybrid CSPs. Another vein in our future work is therefore to apply our tech-

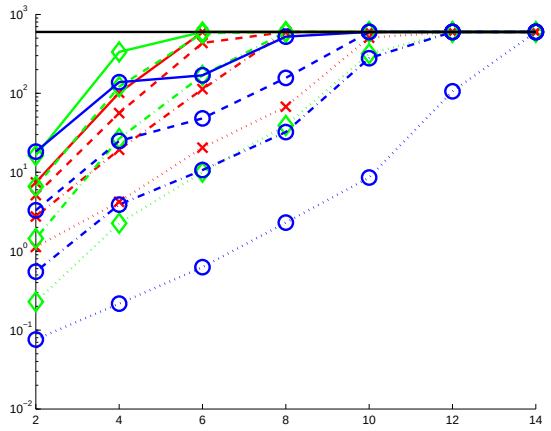


Fig. 2(a) Running times ($K = 3$)

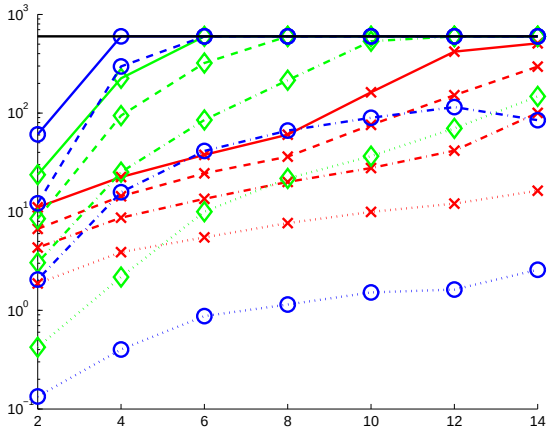


Fig. 2(b) Running times ($K = 5$)

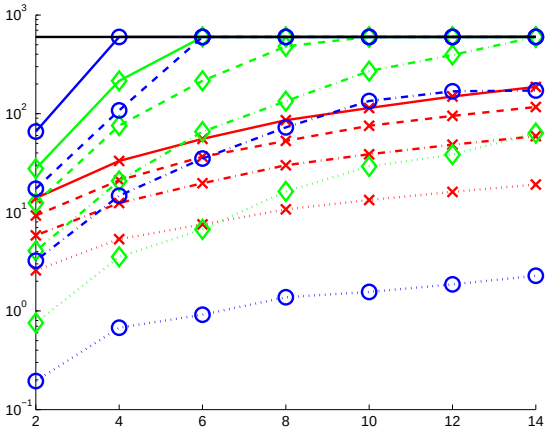


Fig. 2(c) Running times ($K = 7$)

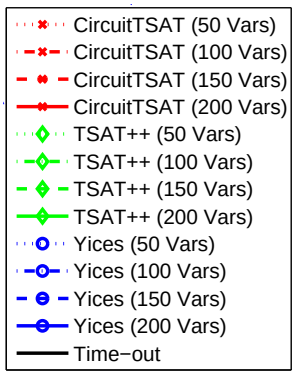


Fig. 2(d) Legend

Fig. 2 Shows the relative performances of CircuitTSAT, TSAT++, and Yices. Figures 2(a), 2(b) and 2(c) show the log of the median CPU times plotted against M/N ranging from 2 to 14 for $K = 3, 5, 7$ respectively. Each data point represents the median of 50 trials.

niques in planning and scheduling domains — e.g. planning with durative actions and resources (Do and Kambhampati 2003), over-subscription planning (Benton, Do, and Kambhampati 2005), planning in metric domains, etc.

Finally, the work presented in this paper suggests a closer examination of the utility of encoding specific combinatorial problems as SAT instances. Previous criticisms of this approach are based on the inability of standard SAT solvers to directly exploit theory-specific properties. However, the empirical evidence in this paper suggests that modern SAT solvers are able to exploit some structural aspects of the DTPs in ways not captured by theory-specific solvers. Firstly, our work suggests further study of other combinatorial problems that may be amenable to circuit-based SAT encodings. Secondly, although CircuitTSAT outperforms TSAT++ and Yices for large values of K and N, its performance is comparatively somewhat poorer for small K and N. This suggests that the solvers exploit different structural aspects of the DTP instances. Therefore, another important line of research is to design DTP solvers that perform well in all regions possibly by directly incorporating theory-specific information into a specialized SAT solver.[7]

Conclusions

We reported on a new solver (CircuitTSAT) for solving large instances of the DTP. This solver is based primarily on the idea of employing "compact" circuit-based representations of DTCs (akin to ripple-carry adders used in computer arithmetic operations). We described several implementation details of CircuitTSAT, and empirically showed that it significantly outperforms TSAT++ and Yices on a wide range of DTP instances. We also commented on the implications

[7] As an example, The MODOC planner (Gelder and Okushi 1999) employs a SAT solver that is customized particularly for solving SAT instances that encode planning problems.

of CircuitTSAT and how circuit-based SAT encodings can exploit important structural information in instances of the DTP. Consequently, we also commented on the importance of revisiting these techniques in a more general context.

References

Armando, A.; Castellini, C.; Giunchiglia, E.; and Maratea, M. 2004. A sat-based decision procedure for the boolean combination of difference constraints. In *Procs. of SAT '04*.

Armando, A.; Castellini, C.; and Giunchiglia, E. 2000. Sat-based procedures for temporal reasoning. In *Procs. of ECP '99*. Springer-Verlag.

Benton, J.; Do, M. B.; and Kambhampati, S. 2005. Over-subscription planning with numeric goals. In *Procs. of IJCAI '05*.

Bryant, R. E.; Lahiri, S. K.; and Seshia, S. A. 2002. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *CAV '02: 14th International Conference on Computer Aided Verification*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1–3).

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)* 20.

Dutertre, B., and de Moura, L. 2006. The Yices SMT solver. http://yices.csl.sri.com/tool-paper.pdf.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Proceedings of SAT '03*.

Gelder, A. V., and Okushi, F. 1999. A propositional theorem prover to solve planning and other problems. *Annals of Mathematics and AI* 26(1-4).

Heuring, V. P.; Jordan, H. F.; and Murdocca, M. 1997. *Computer Systems Design and Architecture*. Addison-Wesley.

Kumar, T. K. S. 2003. Incremental computation of resource-envelopes in producer-consumer models. In *Procs. of CP '03*.

Kumar, T. K. S. 2005. On the tractability of restricted disjunctive temporal problems. In *Procs. of ICAPS '05*.

Kumar, T. K. S. 2006. Tractable classes of metric temporal problems with domain rules. In *Procs. of AAAI '06*.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: engineering an efficient sat solver. In *Design Automation Conference '01*.

Nadel, A. 2002. Backtrack search algorithms for propositional logic satisfiability: Review and innovations. Master's thesis, Hebrew University of Jerusalem.

Smith, D. E.; Frank, J.; and Jónsson, A. K. 2000. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review* 15(1).

Stergiou, K., and Koubarakis, M. 1998. Backtracking algorithms for disjunctions of temporal constraints. In *Procs. of AAAI/IAAI '98*, 248–253.

Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120(1).

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151(1-2).

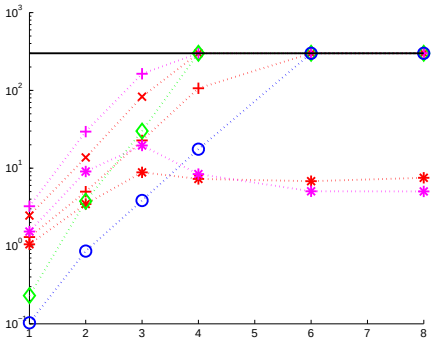


Fig. 3(a) Running times ($K = 2$)

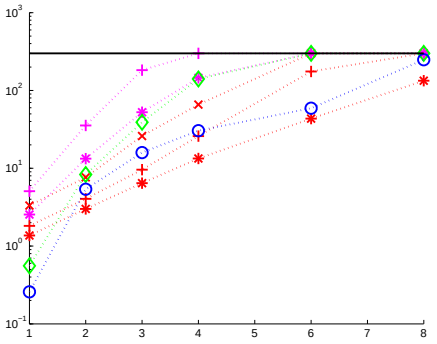


Fig. 3(b) Running times ($K = 3$)

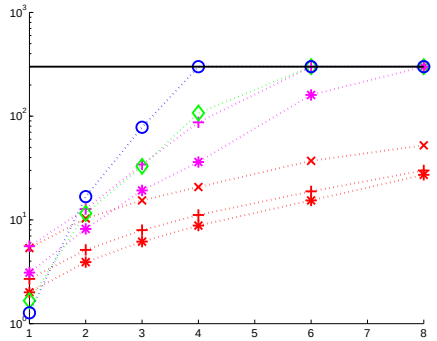


Fig. 3(c) Running times ($K = 5$)

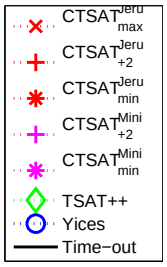


Fig. 3(d) Legend

Fig. 3 The above figures show the relative performances of different variants of CircuitTSAT, TSAT++, and Yices. The variants of CircuitTSAT use different SAT solvers and different bit-spaces. The superscript indicates the solver (Jeru = JeruSAT, Mini = MiniSAT) and the subscript indicates the bit-space. Figures 3(a), 3(b) and 3(c) show the log of the median CPU times plotted against M/N ranging from 2 to 8 for $K = 2, 3, 5$ respectively. Each data point represents the median of 20 trials.