Solving Disjunctive Temporal Problems with Preferences using Boolean Optimization solvers

Marco Maratea[1], Maurizio Pianfetti[1], and Luca Pulina[2]

[1] DIST, University of Genova, Viale F. Causa 15, Genova, Italy.
marco@dist.unige.it,maurizio.pianfetti@studenti.ingegneria.unige.it
[2] DEIS, University of Sassari, Piazza Università 11, Sassari, Italy.
lpulina@uniss.it

Abstract. The Disjunctive Temporal Problem (DTP), which involves Boolean combination of difference constraints of the form $x - y \leq c$, is an expressive framework for constraints modeling and processing. When a DTP is unfeasible we may want to select a feasible subset of its DTP constraints (i.e., disjunctions of difference constraints), possibly subject to some degree of satisfaction: The Max-DTP extends DTP by associating a preference, in the form of weight, to each DTP constraint for its satisfaction, and the goal is to find an assignment to its variables that maximizes the sum of weights of satisfied DTP constraints. In this paper we first present an approach based on Boolean optimization solvers to solve Max-DTPs. Then, we implement our ideas in TSAT++, an efficient DTP solver, and evaluate its performance on randomly generated Max-DTPs, using both different Boolean optimization solvers and two optimization techniques.

1 Introduction

The Disjunctive Temporal Problem (DTP), introduced in [8], is defined as the finite conjunction of DTP constraints, each DTP constraint being a finite disjunction of difference constraints of the form $x-y \leq c$, where x and y are arithmetic variables ranging over a domain of interpretation (the set of real numbers $\mathbb{R}$ or the set of integers $\mathbb{Z}$), and c is a numeric constant. The goal is to find an assignment to the variables of the problem that satisfies all DTP constraints. The DTP is recognized to be a good compromise between expressivity and efficiency, given that the arithmetic consistency of a set of difference constraints can be checked in polynomial time, and has found applications in many areas such as planning, scheduling, hardware and software verification, see, e.g., [19, 5]. Along the years several systems that can solve DTPs have been developed, e.g., SK [21], TSAT [2], CSPI [18], EPILITIS [23], TSAT++ [4], and MATHSAT [5]. Moreover, the competition of solvers for Satisfiability Modulo Theories (SMT-COMP)[3] has two logics that include DTPs (called QF_RDL and QF_IDL, respectively).

When a DTP is unfeasible, i.e., unsatisfiable, we may want to select a feasible subset of its DTP constraints, which can be possibly subject to some degree of satisfaction: The maximum satisfiability problem on a DTP (i.e., Max-DTP) extends DTP by associating a preference, in the form of cost, or weight, to each DTP constraint for taking

Proceedings of the 18^{th} RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2011).
In conjunction with IJCAI 2011, Barcelona, Spain, July 17-18, 2011.

[3] http://www.smtcomp.org/.

into account what the reward for the DTP constraint's satisfaction. The goal is to find an assignment to the variables of the problem that maximizes the sum of the reward of satisfied DTP constraints. The introduction of preferences in DTPs has been first presented in [20], where complex preferences can be assigned to each difference constraint.

In this paper we present an approach which extends the lazy SAT-based approach implemented in solvers for DTPs. The idea is to (i) abstract a Max-DTP P into a Conjunctive Normal Form (CNF) formula ϕ and an optimization function f; (ii) find a solution for φ under f with a Boolean optimization solver; and (iii) verify if the solution returned is consistent. Step (ii) can be implemented with a variety of approaches and solvers, ranging from Max-SAT[4] and Pseudo-Boolean (PB) [5], to Answer Set Programming (ASP) [12, 13]. Then, we implement our ideas by modifying the DTP solver TSAT++, a well-known and efficient solver for solving DTPs, and call the resulting system TSAT#. We finally evaluate its performance on randomly generated DTPs, using a well-known generation method from [21] extended with randomly generated weights. We focus our analysis on TSAT#, as representative of the solvers implementing the lazy SAT-based approach to DTPs, and consider Max-SAT and PB solvers as back-engines, as well as two optimization techniques that proved effective for solving DTPs. Our preliminary results show that the Max-SAT solver AKMAXSAT performs well on these benchmarks, and that the employed optimization techniques help to reducing the search time.

2 Formal Background

Disjunctive Temporal Problems. Temporal constraints have been introduced in [8], as an extension of the Simple Temporal Problem (STP), which consists of conjunction of different constraints. Let $\mathcal{V}$ be a set of symbols, called *variables*. A *difference constraint*, or simply *constraint* is an expression of the form $x - y \leq c$, where $x, y \in \mathcal{V}$, and c is a numeric constant. A *DTP formula*, or simply *formula*, is a combination of constraints via the unary connective "$\neg$" for negation and the n-ary connectives "$\wedge$" and "$\vee$" ($n \geq 0$) for conjunction and disjunction, respectively. A *constraint literal*, or simply *literal*, is either a constraint or its negation. If a is a constraint, then $\overline{a}$ abbreviates $\neg a$ and $\overline{\neg a}$ stands for a. Let the set $\mathbb{D}$ (*domain of interpretation*) be either the set of the real numbers $\mathbb{R}$, or the set of integers $\mathbb{Z}$. An *assignment* is a total function mapping variables to $\mathbb{D}$. Let σ be an assignment and ϕ be a formula. Then $\sigma \models \phi$ (σ satisfies *a formula* ϕ) is defined as follows.

$\sigma \models x - y \leq c$ if and only if $\sigma(x) - \sigma(y) \leq c$,
$\sigma \models \neg\phi$ if and only if it is not the case that $\sigma \models \phi$,
$\sigma \models (\wedge_{i=1}^{n} \phi_i)$ if and only if for each $i \in [1, n]$, $\sigma \models \phi_i$, and
$\sigma \models (\vee_{i=1}^{n} \phi_i)$ if and only if for some $i \in [1, n]$, $\sigma \models \phi_i$.

If $\sigma \models \phi$ then σ will also be called a *model* of ϕ. We also say that a formula ϕ is *satisfiable* if and only if there exists a model for it.

[4] `http://www.maxsat.udl.cat/`.
[5] See, e.g., `http://www.cril.univ-artois.fr/PB10/`.

A DTP is the problem of deciding whether a formula is satisfiable or not in the given domain of interpretation $\mathbb{D}$. Notice that the satisfiability of a formula depends on $\mathbb{D}$, e.g., the formula $x - y > 0 \wedge x - y < 1$ is satisfiable if $\mathbb{D}$ is $\mathbb{R}$ but unsatisfiable if $\mathbb{D}$ is $\mathbb{Z}$. However, the problems of checking satisfiability in $\mathbb{Z}$ and $\mathbb{R}$ are closely related and will be almost always treated uniformly. Therefore, from now on, we will drop the distinction (and we will re-introduce it only when needed). The definition of DTP we have made in this section extends the classical one [8] given that the DTP is originally introduced in CNF, i.e., a DTP is represented as a conjunctively intended set of DTP constraints, each DTP constraint being a disjunctively intended set of difference constraints. From now on, we consider the formula to be in CNF.

Maximum satisfiability for DTPs. Given an unsatisfiable DTP ϕ, we may want to find an assignment σ' that maximizes the number of satisfied DTP constraints (called "Max-DTP"), or maximizes the sum of the weights associated to each satisfied DTP constraints (called "Weighted-Max-DTP"), in case the DTP constraints are subject to some degree of satisfaction. The name of the problems are borrowed from the terminology used in the Max-SAT Evaluations and Competitions, see, e.g., [1]. In related works, e.g., [7, 17] always Max-SMT (meaning, e.g., Max-DTP in case the optimization is defined on a DTP) is used, regardless of whether constraints are "weighted", or not. In the following, we will use the same terminology, for us meaning what we are going to define. Formally, a Max-DTP is a pair $\langle \phi, w \rangle$ where (i) ϕ is an (unsatisfiable) DTP, and (ii) w is a function that maps each DTP constraint to a positive integer number.

Our goal is to find an assignment σ' for ϕ that maximizes the sum of weights associated to satisfied DTP constraints, i.e., maximizes the following linear objective function

$$\sum_{d \in \phi, \sigma' \models d} w(d)$$

where d is a DTP constraint.

3 Algorithm and Optimizations

Algorithm Our approach for solving DTPs extends the lazy SAT-based approach employed by TSAT++ [4] for solving formulas in Separation Logic [22], which involves difference constraints. For convenience, here we restrict its algorithm to DTPs. Given a Max-DTP $P{:=}\langle \phi, w \rangle$, our approach works by

1. abstracting the DTP ϕ into a propositional CNF formula φ: Each difference constraint is replaced by a new distinct propositional atom;
2. solving φ subject to a linear optimization function defined on the clauses of φ: This is done by relying on Boolean optimization solvers that return a propositional assignment μ for φ;
3. checking μ for arithmetic consistency in $\mathbb{D}$: Each propositional literal $l \in \mu$ is replaced by the corresponding constraint literal, and the consistency of all constraint literals is verified. If the consistency check succeeds, then we have found a solution to P, otherwise the algorithm goes back to step 2

It is a well known fact that BF can be used in step 3 to check the satisfiability of a finite set Q of constraints of the form $x - y \leq c$. This is done by first building a *constraint graph for* Q, see, e.g., [6]. The soundness and completeness of the algorithm is guaranteed by the soundness and completeness of the underlying solving procedure for solving DTPs, i.e., the solving procedure for ϕ (showed in [4]), and from the soundness and completeness of the Boolean optimization procedures employed. For solving step 2 a wide range of formulations, solving procedures and techniques can be employed, e.g., Weighted Max-SAT, Pseudo-Boolean, and ASP.

In the next paragraphs we present two optimization techniques that can help to improve the performance of the basic algorithm presented in this section. However, there is an optimization to the basic procedure that it is set by default in TSAT#: If the consistency check at step 3 does not succeed, we add a "reason" to the abstraction formula φ, i.e., a clause that prevents the solver employed to re-compute an assignment μ having the literals corresponding to constraint literals that caused the arithmetic inconsistency assigned in the same way. Given the BF, computing such reason can be done efficiently, by considering the difference constraints involved in (one of the) negative cycles. Of course, methods for limiting the number of added reasons is needed, in order to let the procedure to still working in polynomial space, e.g., given a positive integer b, by adding only the reasons that contain a number of literals less or equal than b.
Optimizations. We herewith highlight two optimization techniques, one theory dependent and one theory independent, that proved to be effective for solving DTPs, and that can be fruitfully used with black-box engines. Their general idea is to reduce the enumeration of unfruitful assignments at a reasonable price. The first one, denoted with IS_2, is a preprocessing step: For each unordered pair $\langle c_i, c_j \rangle$ of distinct difference constraints appearing in the formula ϕ and involving the same variables, all possible pairs of literals built out of them are checked for consistency. Assuming c_i and c_j are inconsistent, the constraint $\overline{c_i} \vee \overline{c_j}$ is added to the input formula before calling TSAT#. The second technique, called "model reduction", is based on the observation that an assignment μ generated by TSAT# can be redundant, that is, there might exist an assignment $\mu' \subset \mu$ that propositionally entails the input formula. When this is the case, we can check the consistency of μ' instead μ. Details for both techniques can be found in [4].

4 Implementation and Experimental Analysis

We have implemented TSAT# as an extension of the TSAT++ solver [3], by integrating some Max-SAT and PB solvers as back-engines for reasoning about Boolean optimization problems. Specifically, the employed solvers are: MINIMAXSAT ver. 1.0 [14], MINISAT+ [9] ver. 1.14, and AKMAXSAT [15], the version submitted to the last Max-SAT 2010 Competition. These are well-known solvers for Boolean optimization and among the best Partial Weighted Max-SAT[6] and Pseudo-Boolean (focusing on the OPT-SMALL-INT[7] category) solvers, in various Max-SAT and PB Evaluations and Compe-

[6] The "partial" version of the problem, where both hard and soft clauses are present, is needed because the original clauses of the abstracted problem are soft, while the added ones are hard.
[7] We remind that this is a category of PB Evaluations and Competitions where (i) no constraint has a sum of coefficients greater than 2^{20} (20 bits), and (ii) the objective function is linear.

titions. We remind that MINISAT+ is a PB solver, AKMAXSAT is a Max-SAT solver, while MINIMAXSAT accepts problems in both formalisms: Given it has been mainly evaluated on Max-SAT formulations, we rely on such format in our analysis. Given a CNF formula φ, and a function w[8] mapping each clause to a positive integer number that represents its weight, the main implementation part has been devoted to introduce weights and formulate the optimization problems in Max-SAT and PB formats. For Max-SAT problems, there is an immediate formulation by directly assigning weights to clauses, while PB problems need "clause selectors" to be added to each soft clause, and the optimization function to be defined over the clause selectors. In the following, given a Boolean optimization solver X,

1. TSAT#(X) is TSAT# in plain configuration employing X for step 2;
2. TSAT#+p(X) is TSAT# with model reduction enabled employing X for step 2;
3. TSAT#+is(X) is TSAT# with IS_2 preprocessing enabled employing X for step 2;
4. TSAT#+is+p(X) is TSAT# with both model reduction and IS_2 preprocessing enabled employing X for step 2.

About the benchmarks, we randomly generated Max-DTPs, using a well-known generation method from [21] extended with random weights. In particular, in our model Max-DTPs are randomly generated by fixing the number k of disjuncts per clause, the number n of arithmetic variables, a positive integer L such that all the constants are taken in $[-L, L]$, and a positive integer w such that all the weights are taken in $[1, w]$. Then, (i) the number of clauses m is increased to create bigger problems, (ii) for each tuple of values of the parameters, 10 instances are generated and then fed to the solvers, and (iii) the median of the CPU times is plotted against the m/n ratio. We fix $k = 2$, $L = 100$, $w = 100$, $n = 5, 10$, and the ratio m/n varying from 6 to 10. The lower bound of m/n has been fixed as the lower positive integer for which there is a majority of unsatisfiable underlying DTPs. Further note that the DTP is already a "difficult" problem, and the analysis in literature on DTPs have been performed on problems with few tens of variables for the setting used in this paper[9]: adding preferences further increase the difficulty. The timeout for each problem has been set to 1800s on a Linux box equipped with a Pentium IV 3.2GHz processor and 1GB of RAM.

Fig. 1 shows the results for TSAT# employing MINIMAXSAT (top), MINISAT+ (middle) and AKMAXSAT (bottom), respectively, on randomly generated Max-DTPs with $n = 5$. For each plot, the left one considers real-valued variables, while the right one considers integer-valued variables. First, we can note that the optimization techniques described help to significantly improve the efficiency: Enhancing the plain TSAT# version with one of the technique helps reducing the overall CPU time of around a factor of 3, while enabling both techniques in conjunction improves the performance of around one order of magnitude. Comparing the performance of the various Boolean optimization solvers employed, AKMAXSAT is clearly the best underlying

[8] w has been originally defined on DTPs. After the abstraction, there is a one-to-one correspondence between each DTP constraint and the related clause in φ. Thus, with a slight abuse of notation, we consider the optimization function to be defined on the clauses of φ.

[9] In [16], DTPs with many variables n are used, but the analysis is focused on problems with $k > 2$ having ratios m/n such that a vast majority of instances are satisfiable.

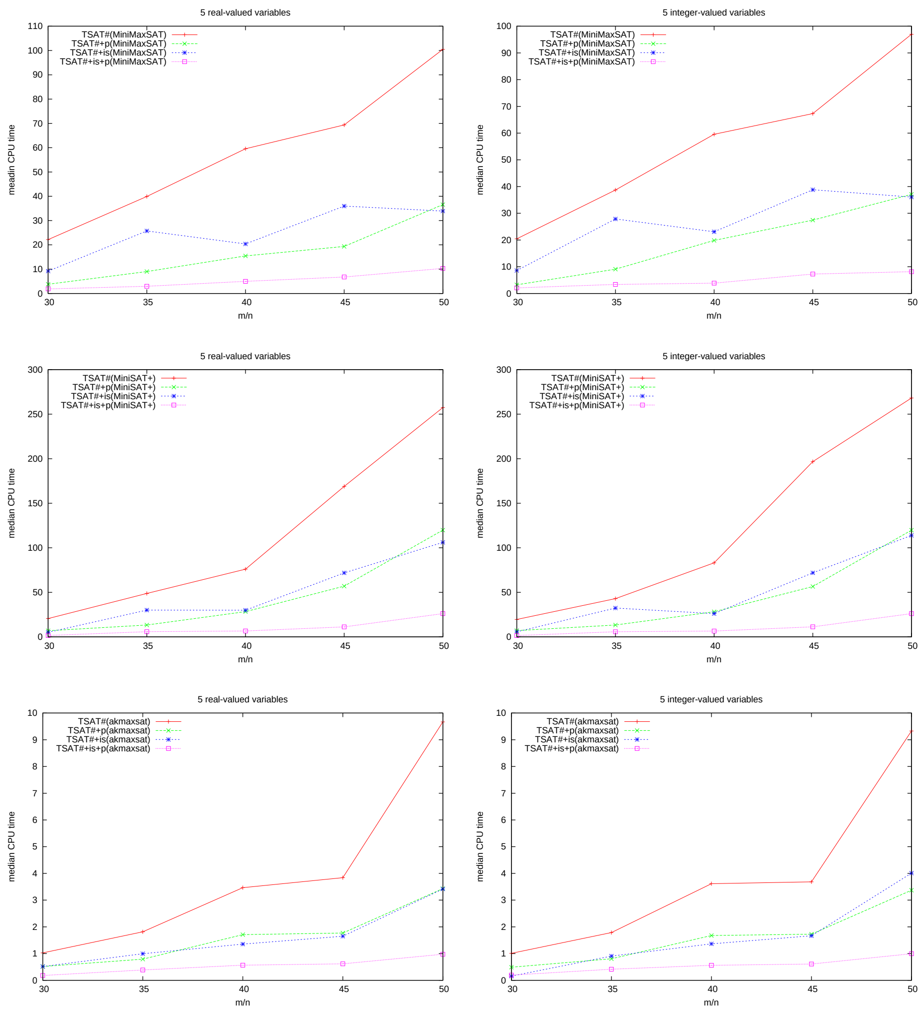

Fig. 1. Results of TSAT# employing MINIMAXSAT (top), MINISAT+ (middle), and AKMAXSAT (bottom) on random Max-DTPs with 5 real-valued (left) and integer-valued (right) variables.

solver on these benchmarks among the ones analyzed: On the biggest instances, it gains around one order of magnitude over MINIMAXSAT, and more than a factor of 20 over MINISAT+. All considerations made hold with both real- and integer-valued variables.

The intuition for the superior performance of AKMAXSAT is that, given the results of the 2010 Max-SAT Competition, AKMAXSAT seems to be very effective on randomly generated and synthetic benchmarks, Given our approach, the starting abstraction formula has the following structure: It is a fixed-length formula where each variable occurs once (with high probability) in the formula. But this was not fully expected: During the

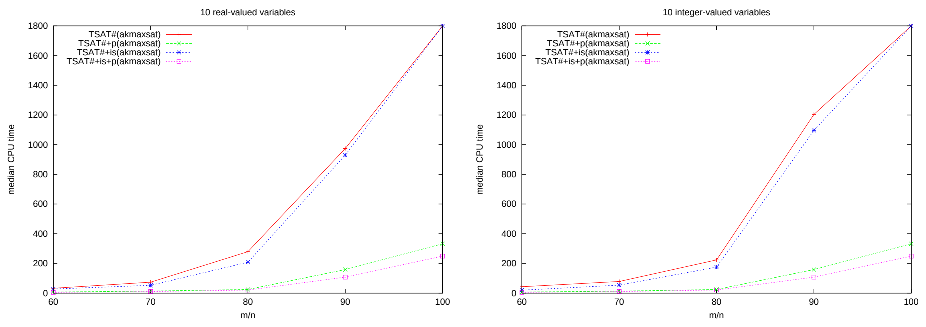

Fig. 2. Results of TSAT# employing AKMAXSAT on random Max-DTPs with 10 real-valued (Left) and integer-valued (Right) variables.

search, adding constraints corresponding to reasons, the number of occurrences of literals increase, giving to the formula a less "synthetic" structure.

We increase the number of variables n to 10, and focus the analysis on TSAT# employing AKMAXSAT, i.e., our best Boolean optimization solver on these benchmarks. From Fig. 2 we can note that the impact of the optimization techniques is now different: Model reduction improves dramatically the performance of TSAT#(AKMAXSAT), while the impact of the preprocessing is limited with this setting.

5 Conclusions and Future Work

In this paper we have presented an approach to solving weighted maximum satisfiability on DTPs. The approach extends the one implemented in TSAT++, by employing Boolean optimization solvers as reasoning engines. The performance of the resulting system, TSAT#, employing some Max-SAT and PB solvers are analyzed on randomly generated benchmarks, together with the impact that both theory dependent and independent optimization techniques have on its performance. The AKMAXSAT Max-SAT solver is the best option among the solvers analyzed, and both optimization techniques help to improve the overall performance. Current research includes (i) the integration of other solvers in TSAT#, e.g., the ASP solver CLASP [11], that have proved to be very competitive at the 2009 PB Competition, (ii) the extension of our algorithm to deal with other forms of preferences, e.g., where weights can be associated to each different constraint, for which it is still possible to rely on PB and ASP formalisms and systems, and (iii) a comparative analysis with rival tools for which such more complex preferences can be easily specified, e.g. MAXILITIS and HYSAT [10].

Acknowledgments. We would like to thank Adrian Kügel for providing, and getting support to, AKMAXSAT, and to Michael D. Moffitt and Bart Peintner for discussions about MAXILITIS and WEIGHTWATCHER.

References

1. J. Argelich, C. M. Li, F. Manyà, and J. Planes. The first and second Max-SAT evaluations. *Journal of Satisfiability, Boolean Modelling and Computation*, 4(2-4):251–278, 2008.
2. A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. *Proc. of ECP 1999*, volume 1809 of *LNCS*, pages 97–108. Springer, 1999.
3. A. Armando, C. Castellini, E. Giunchiglia, M. Idini, and M. Maratea. TSAT++: An open platform for satisfiability modulo theories. *ENTCS*, 125(3):25–36, 2005.
4. A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. The SAT-based approach to Separation Logic. *Journal of Automated Reasoning*, 35(1-3):237–263, 2005.
5. M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. MathSAT: Tight integration of SAT and mathematical decision procedures. *Journal of Automated Reasoning*, 35(1-3):265–293, 2005.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.
7. L. de Moura. `http://yices.csl.sri.com/`.
8. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, Jan. 1991.
9. N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
10. M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.
11. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proc. of IJCAI 2007*, pages 386–392. Morgan Kaufmann Publishers, 2007.
12. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*, pages 1070–1080, 1988.
13. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
14. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: A new weighted Max-SAT solver. *Journal of Artificial Intelligence Research (JAIR)*, 31:1–32, 2008.
15. A. Kuger. Improved exact solver for the weighted Max-SAT problem. In *Proc. of the 2nd Pragmatics of SAT (PoS-10) workshop*, 2010.
16. B. Nelson and T. K. S. Kumar. CircuitTSAT: A solver for large instances of the disjunctive temporal problem. *In Proc. of ICAPS 2008*, pages 232–239. AAAI Press, 2008.
17. R. Nieuwenhuis and A. Oliveras. On sat modulo theories and optimization problems. In *Proc. of SAT 2006*, volume 4121 of *LNCS*, pages 156–169. Springer, 2006.
18. A. Oddi and A. Cesta. Incremental forward checking for the disjunctive temporal problem. In *Proc. of ECAI-2000*, pages 108–112, Berlin, 2000.
19. A. Oddi, R. Rasconi, and A. Cesta. Project scheduling as a disjunctive temporal problem. In *Proc. of ECAI 2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 967–968. IOS Press, 2010.
20. B. Peintner and M. E. Pollack. Low-cost addition of preferences to DTPs and TCSPs. In *Proc. of AAAI 2004*, pages 723–728. AAAI Press / The MIT Press, 2004.
21. K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.
22. O. Strichman, S. A. Seshia, and R. E. Bryant. Deciding Separation formulas with sat. In *Proc. of CAV 2002*, volume 2404 of *LNCS*, pages 209–222. Springer, 2002.
23. I. Tsamardinos and M. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151:43–89, 2003.