Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011), pp.560-565

Dynamic SAT with Decision Change Costs: Formalization and Solutions

Daisuke Hatano and **Katsutoshi Hirayama**
Kobe University
5-1-1 Fukaeminami-machi, Higashinada-ku, Kobe 658-0022, Japan
daisuke-hatano@stu.kobe-u.ac.jp, hirayama@maritime.kobe-u.ac.jp

Abstract

We address a dynamic decision problem in which decision makers must pay some costs when they change their decisions along the way. We formalize this problem as Dynamic SAT (DynSAT) with decision change costs, whose goal is to find a sequence of models that minimize the aggregation of the costs for changing variables. We provide two solutions to solve a specific case of this problem. The first uses a Weighted Partial MaxSAT solver after we encode the entire problem as a Weighted Partial MaxSAT problem. The second solution, which we believe is novel, uses the Lagrangian decomposition technique that divides the entire problem into sub-problems, each of which can be separately solved by an exact Weighted Partial MaxSAT solver, and produces both lower and upper bounds on the optimal in an anytime manner. To compare the performance of these solvers, we experimented on the random problem and the target tracking problem. The experimental results show that a solver based on Lagrangian decomposition performs better for the random problem and competitively for the target tracking problem.

1 Introduction

With recent success in SAT solving, we observed a growing need to extend SAT to deal with more sophisticated problems in the real world. Dynamic SAT (DynSAT) [Hoos and O'Neill, 2000] is one such extension that aims to model the dynamic nature of real problems. DynSAT can be considered a special case of Dynamic CSP (DynCSP), which was originally proposed in [Dechter and Dechter, 1988]. In DynSAT and DynCSP, we are given a sequence of problem instances and required to solve it.

The solutions for DynCSP (including DynSAT) are largely divided into two categories [Verfaillie and Jussien, 2005]. The first encompasses *reactive approaches*, which use no knowledge about the possible directions of future changes. Their goal is to find a solution for a new problem instance, which was produced by changes in the current problem instance. The key idea of reactive approaches is to *reuse* something. Both *solution reuse* [Verfaillie and Schiex, 1994] and *reasoning reuse* [Bessière, 1991; Schiex and Verfaillie, 1993] are typical techniques in the literature.

The second encompasses *proactive approaches*, which exploit all the knowledge they have about possible future changes. With such knowledge, they have a chance to produce solutions that will resist those possible changes. The *robust solution* [Fargier *et al.*, 1996; Wallace and Freuder, 1998; Walsh, 2002] and the *flexible solution* [Bofill *et al.*, 2010; Freuder, 1991; Ginsberg *et al.*, 1998] are examples.

In this work, we address a dynamic decision problem in which decision makers must pay some costs when they change their decisions along the way. We can observe such costs in real-life problems, such as the setup costs in planning and scheduling. Suppose, for example, that you have to make this month's schedule for the hospital staff. You may have information about possible future events, such as who is on holiday or when a delicate surgery is planned. In the face of such future events, an efficient schedule is required by which day-to-day operations go smoothly and, more specifically, the cost of arrangement is minimized. Even though the costs of decision changes are widely observed in various dynamic decision problems, little work has dealt with this issue in the context of CSP or SAT.

One exception is a minimal-change solution for DynCSP [Ran *et al.*, 2002] that considers the cost of decision changes as the number of variables that are assigned new values. Similar approaches have also been proposed, including diverse and similar solutions for CSP [Hebrard *et al.*, 2005] and *distance-SAT*, whose goals are to find a model that disagrees with a given partial assignment on at most a specified number of variables [Bailleux and Marquis, 2006]. Clearly, the main concern of these works is a short-term reactive solution. On the other hand, to our knowledge, a proactive solution has not been fully investigated for a general decision problem with the cost of decision changes.

We first introduce a dynamic SAT with decision change costs. We restricted our attention to SAT, not CSP, as a decision problem because many efficient solvers and effective encoding methods already exist for SAT. The input of this problem is a sequence of SAT instances and the decision change costs for the variables. The output is a sequence of models (solutions) that minimizes the aggregation of the costs for changing variables. Obviously, this is one proactive approach for DynSAT.

Then, we provide two solutions for a specific case of DynSAT with decision change costs. The first solution uses a Weighted Partial MaxSAT (WPMaxSAT) solver after encoding the entire problem as a WPMaxSAT problem. The second solution, which we believe is novel, uses the *Lagrangian decomposition* technique [Bertsekas, 1999] to devide the entire problem into sub-problems, each of which can be separately solved by an exact WPMaxSAT solver, and produces both lower and upper bounds on the optimal in an anytime manner.

The remainder of this paper is organized as follows. We first define SAT and DynSAT in Section 2 and present a DynSAT with decision change costs in Section 3. Next, we describe two solutions for a specific case of DynSAT with decision change costs in Section 4. Finally, we experimentally evaluate the performance of various solvers, each of which is based on either solution, and conclude this work in Sections 5 and 6.

2 SAT and Dynamic SAT

SAT is a decision problem whose goal is to decide whether a given *CNF formula* has a *model*. A CNF formula is a conjunction of *clauses*, where each clause is a disjunction of *literals* and a literal is a Boolean variable or its negation. A truth assignment is a mapping from Boolean variables to truth values, where we mean `true` by 1 and `false` by 0, and a model for a CNF formula is a truth assignment that makes the formula true.

Dynamic SAT (DynSAT) is an extension of SAT that models the dynamic nature of real problems. Here, we define DynSAT [Hoos and O'Neill, 2000]:

Definition 1 (DynSAT) *An instance of a dynamic SAT is given by (X, ϕ), where $X = \{x_1, \ldots, x_n\}$ is a set of Boolean variables, and ϕ is a function $\phi : T \to \mathrm{CNF}(X)$, where T is a set of non-negative integers and $\mathrm{CNF}(X)$ is the set of all possible CNF formulas that use only Boolean variables in X.*

An instance of DynSAT forms an (infinite) sequence of CNF formulas on the Boolean variables in X, where a CNF formula at time t will be given by function ϕ.

k-stage DynSAT is a DynSAT that does not change after fixed number k of time steps.

Definition 2 (k-stage DynSAT) *An instance of k-stage dynamic SAT is given by (k, X, ϕ), where $\forall t \geq k : \phi(t) = \phi(k-1)$.*

Given an instance of DynSAT, our goal is to find a sequence of models. This task is called *model tracking* [Hoos and O'Neill, 2000].

3 Dynamic SAT with Decision Change Costs

Naturally when a model has changed over time, the decision makers alter their decisions accordingly. We assume that if they change their decisions in the real world, they must pay a cost (such as the setup cost in planning and scheduling). We generally call this the *decision change cost*.

We define the cost for changing a variable at a certain time.

Definition 3 (cost for changing variable at a certain time) *The cost for changing variable x_i at time t is given by function $f : T\backslash\{0\} \times X \times \{1,0\} \times \{1,0\} \to \mathcal{R}^+$, where T is a set of non-negative integers, X is a set of Boolean variables, and $\mathcal{R}^+$ is a set of positive real numbers.*

For example, $f(t, x_i, 1, 0)$ returns cost $c_0^{i,t}$ when we change variable x_i from 1 to 0 at time t. Similarly, $f(t, x_i, 0, 1)$ returns cost $c_1^{i,t}$ when we change variable x_i from 0 to 1 at time t. Obviously, both $f(t, x_i, 1, 1)$ and $f(t, x_i, 0, 0)$ must return 0 for any x_i and t, since there should be no cost when we keep the same value for a variable.

Given two consecutive models, $M(t-1)$ and $M(t)$, we can identify the cost for changing variable x_i between these two models, denoted by $cost(x_i, M(t-1), M(t))$, by referring to the above cost function of f. For example, given that x_i is 1 in $M(t-1)$ but 0 in $M(t)$, the value of $cost(x_i, M(t-1), M(t))$ must be $f(t, x_i, 1, 0)$. By aggregating all of the costs for variables with a *local aggregation operator* $\oplus$, we can compute the cost for changing a model from $M(t-1)$ to $M(t)$, which we will denote by $cost(M(t-1), M(t))$:

$$cost(M(t-1), M(t)) \equiv \bigoplus_{x_i \in X} cost(x_i, M(t-1), M(t)).$$

For example, if $\oplus$ is $'+'$, we have

$$cost(M(t-1), M(t)) \equiv \sum_{x_i \in X} cost(x_i, M(t-1), M(t)).$$

Furthermore, let M be a sequence of models over a set of non-negative integers T, i.e., $M = \{M(t) \mid t \in T\}$. We can define the cost of the sequence of models M by:

$$cost(M) \equiv \bigodot_{t \in T\backslash\{0\}} cost(M(t-1), M(t)), \qquad (1)$$

where $\odot$ is a *global aggregation operator* over the costs for changing models. For example, if $\odot$ is $'+'$, we have

$$cost(M) \equiv \sum_{t \in T\backslash\{0\}} cost(M(t-1), M(t)).$$

We can now define DynSAT with decision change costs.

Definition 4 (DynSAT with decision change costs) *An instance of dynamic SAT with decision change costs is given by a 5-tuple $(X, \phi, f, \oplus, \odot)$, where X and ϕ are in Definition 1, f is in Definition 3, and $\oplus$ and $\odot$ are local and global aggregation operators, respectively.*

As with the plain DynSAT, we can define k-stage DynSAT with decision change costs as follows.

Definition 5 (k-stage DynSAT with decision change costs) *An instance of k-stage dynamic SAT with decision change costs is given by a 6-tuple $(k, X, \phi, f, \oplus, \odot)$, where $\forall t \geq k : \phi(t) = \phi(k-1)$.*

Given an instance of (k-stage) DynSAT with decision change costs, our goal is to find a sequence of models whose cost, generally defined by (1), is minimized. We call this minimal cost the *optimal value* for a (k-stage) DynSAT with decision change costs. If the optimal value is finite, we refer to the sequence of models that achieves the minimal cost as an *optimal solution*.

4 Solutions

Among possible DynSATs with decision change costs, we focus on the k-stage problem specified by $(k, X, \phi, f, +, +)$, which we believe is one natural setting. In this section, we provide two solutions for it.

4.1 Weighted Partial MaxSAT Solving

In our first solution, we translate a given instance of $(k, X, \phi, f, +, +)$ into a Weighted Partial MaxSAT (WPMaxSAT) problem instance and solve it using any WPMaxSAT solver. A WPMaxSAT problem instance comprises *hard clauses* that must be satisfied and *soft clauses* that can be violated by paying some designated costs (called *weights*). The goal of WPMaxSAT solving is to find a truth assignment that satisfies all hard clauses and minimizes a weighted sum of the violated soft clauses.

The translation is as follows. For every clause of each CNF formula $\phi(t)$, we introduce a hard clause with its Boolean variables labeled by t. For example, clause $x_1 \vee x_2$ of $\phi(2)$ results in the hard clause of $x_1^2 \vee x_2^2$. Furthermore, for each mapping defined by f, we introduce a soft clause that bridges the same Boolean variables belonging to different times. For example, the mapping of $f(2, x_1, 0, 1) = 7$, indicating that we must pay a cost of 7 when changing variable x_1 from 0 to 1 at time 2, results in the soft clause of $x_1^1 \vee \neg x_1^2$ with weight 7. Generally, $f(t, x_i, 1, 0) = c_0^{i,t}$ results in $\neg x_i^{t-1} \vee x_i^t$ with weight $c_0^{i,t}$, while $f(t, x_i, 0, 1) = c_1^{i,t}$ results in $x_i^{t-1} \vee \neg x_i^t$ with weight $c_1^{i,t}$.

4.2 Lagrangian Decomposition

Our second solution uses the *Lagrangian decomposition* technique [Bertsekas, 1999] that can provide both lower and upper bounds on the optimal value of the problem.

Decomposition

First, we translate the cost function of f into the *0-1 integer programming (IP) problem*. Suppose we have $f(t, x_i, 1, 0) = c_0^{i,t}$, $f(t, x_i, 0, 1) = c_1^{i,t}$, $f(t, x_i, 1, 1) = 0$ and $f(t, x_i, 0, 0) = 0$. These cost mappings can be achieved by solving the following 0-1 IP problem:

$$\begin{aligned} \text{min.} \quad & c_0^{i,t} y_0^{i,t} + c_1^{i,t} y_1^{i,t} \\ \text{s. t.} \quad & x_i^{t-1} - x_i^t - y_0^{i,t} + y_1^{i,t} = 0, \\ & x_i^{t-1}, x_i^t, y_0^{i,t}, y_1^{i,t} \in \{0, 1\}, \end{aligned}$$

where x_i^{t-1} and x_i^t are variable x_i at times $t-1$ and t, respectively, and both $y_0^{i,t}$ and $y_1^{i,t}$ are auxiliary 0-1 variables. For example, assuming $x_i^{t-1} = 1$ and $x_i^t = 0$, the above problem has $c_0^{i,t}$ as its optimal value. Obviously, this translation is applied to every combination of x_i and t.

Since we are dealing with the case where local and global aggregation operators are additive, the entire problem of $(k, X, \phi, f, +, +)$ can be represented as what we call a *CNF-included 0-1 integer programming problem*, formalized as follows:

$$\begin{aligned} \mathcal{P} \ : \ \text{min.} \quad & \sum_{t=1}^{k-1} \sum_{i=1}^{n} (c_0^{i,t} y_0^{i,t} + c_1^{i,t} y_1^{i,t}) \\ \text{s. t.} \quad & x_i^{t-1} - x_i^t - y_0^{i,t} + y_1^{i,t} = 0, \qquad (2) \\ & i = 1, ..., n, \ t = 1, ..., k-1, \\ & \phi(t), \quad t = 0, ..., k-1. \qquad (3) \end{aligned}$$

Hereafter we omit the 0-1 constraints on the variables. Since problem $\mathcal{P}$ has all of the CNF formulas (3) as constraints, a feasible solution for $\mathcal{P}$ is a sequence of models. Furthermore, the objective value of such a feasible solution is the total sum of the costs for changing variables. Therefore, we can get an optimal solution for the entire problem by solving $\mathcal{P}$, or more specifically, by projecting an optimal solution for $\mathcal{P}$ onto the variables of x_i^t.

Since solving $\mathcal{P}$ is complex, we will relax this problem so that it can be tractable. In this work, we produce a *Lagrangian relaxation problem* for $\mathcal{P}$ by *dualizing* constraints (2), each of which is defined over the variables belonging to different times:

$$\begin{aligned} \mathcal{L} \ : \ L(\mu) \ = \ \text{min.} \quad & \sum_{t=1}^{k-1} \sum_{i=1}^{n} (c_0^{i,t} y_0^{i,t} + c_1^{i,t} y_1^{i,t}) \\ & + \sum_{t=1}^{k-1} \sum_{i=1}^{n} \mu^{i,t} (x_i^{t-1} - x_i^t - y_0^{i,t} + y_1^{i,t}) \\ \text{s. t.} \quad & \phi(t), \quad t = 0, ..., k-1, \end{aligned}$$

where μ is called a *Lagrange multiplier vector*, each element $\mu^{i,t}$ of which can take any real number. Furthermore, a simple calculation reveals that this problem can be decomposed into the following $k+1$ sub-problems:

$$\begin{aligned} L^{aux}(\mu) \ = \ \text{min.} & \sum_{t=1}^{k-1} \sum_{i=1}^{n} (c_0^{i,t} - \mu^{i,t}) y_0^{i,t} \\ & + \sum_{t=1}^{k-1} \sum_{i=1}^{n} (c_1^{i,t} + \mu^{i,t}) y_1^{i,t}, \qquad (4) \end{aligned}$$

$$L^0(\mu) \ = \ \text{min.} \sum_{i=1}^{n} \mu^{i,1} x_i^0, \quad \text{s. t.} \quad \phi(0), \qquad (5)$$

and, for each time t from 1 to $k-2$,

$$L^t(\mu) \ = \ \text{min.} \sum_{i=1}^{n} (\mu^{i,t+1} - \mu^{i,t}) x_i^t, \quad \text{s. t.} \quad \phi(t), \qquad (6)$$

and

$$L^{k-1}(\mu) \ = \text{min.} \sum_{i=1}^{n} (-\mu^{i,k-1}) x_i^{k-1}, \text{s. t.} \quad \phi(k-1). \qquad (7)$$

Note that each of these sub-problems is actually the WPMaxSAT problem. Furthermore, (4) is trivial because it consists of only soft unit clauses on auxiliary variables, but each

of the other sub-problems consists of hard clauses in $\phi(t)$ and soft unit clauses on the variables of time t.

On the other hand, the *Lagrangian dual problem* is formally defined by

$$\mathcal{D} \ : \ \max. \quad L(\mu) \quad \text{s. t.} \quad \mu \in \Re,$$

where $L(\mu)$ is the optimal value for $\mathcal{L}$, which should vary on μ. This is obviously an unconstrained maximization problem over Lagrange multipliers. The value of the objective function of this Lagrangian dual problem is a lower bound on the optimal value of $\mathcal{P}$. The decomposition of $\mathcal{L}$ results in the decomposition of $\mathcal{D}$, which produces

$$\mathcal{D} \ : \ \max. \quad L^{aux}(\mu) + \sum_{t=0}^{k-1} L^t(\mu) \quad \text{s. t.} \quad \mu \in \Re. \qquad (8)$$

Our procedure solves this (decomposed) Lagrangian dual problem to search for the values of Lagrange multipliers that give the highest objective, i.e., the highest lower bound on the optimal value of $\mathcal{P}$. This lower bound is useful because it can be exploited in a search algorithm for $\mathcal{P}$. For specific values to μ, we can compute a lower bound on the optimal value of $\mathcal{P}$ by simply taking a total sum of the optimal values of the sub-problems from (4) to (7).

Outline of Procedure

Here we describe the outline of our procedure that can provide both lower and upper bounds on the optimal value of $\mathcal{P}$ along with a feasible solution for $\mathcal{P}$.

Step 1: Set every element in μ to 0.

Step 2: Solve all of the sub-problems from (4) to (7) using an exact WPMaxSAT solver.

Step 3: Compute highest lower bound *LB*, lowest upper bound *UB*, and feasible solution *M* with the lowest upper bound.

Step 4: If *CanTerminate?* then return *LB*, *UB*, and *M*; otherwise update μ and go to Step 2.

Starting from Step 1, this procedure repeats Steps 2 through 4 until the termination condition is met. We refer to one iteration from Steps 2 to 4 as a *round*. Next, we focus on Steps 3 and 4.

Lower and Upper Bounding

As mentioned, we can compute a lower bound on the optimal value of $\mathcal{P}$ as a total sum of the optimal values of sub-problems. We can also provide a feasible solution for $\mathcal{P}$ by forcing some auxiliary variables to flip in their optimal solutions so that they can satisfy each of the constraints (2) that were relaxed to produce the Lagrangian relaxation problem. Such a feasible solution for $\mathcal{P}$ clearly provides an upper bound on the optimal value of $\mathcal{P}$. Therefore, at each round, we can compute both lower and upper bounds on the optimal value of $\mathcal{P}$ as well as a feasible solution for $\mathcal{P}$. At Step 3 in the procedure, we keep the best one among those found in the previous rounds.

Termination

We have two ways to detect the fact that the procedure has found an optimal solution for $\mathcal{P}$. The first relies on the following theorem on the relation between the optimal solutions for the sub-problems and the optimal solution for $\mathcal{P}$.

Theorem 1 *If all optimal solutions for the sub-problems from (4) to (7) satisfy all of the constraints (2) that have been relaxed, then these optimal solutions constitute an optimal solution for $\mathcal{P}$.*

This is obvious because such optimal solutions provide not only a lower bound but also an upper bound on the optimal value of $\mathcal{P}$. Accordingly, we can terminate the procedure when the optimal solutions for the sub-problems satisfy all constraints (2).

The second one is straightforward. When a "forced" feasible solution for $\mathcal{P}$ has the value of an objective function that equals *LB*, this feasible solution is optimal because both *LB* and *UB* now reach the same value. This fact can also be used for terminating the procedure.

On the other hand, the procedure can also be terminated anytime after performing at least one round. In that case, we can get the best bounds and the best feasible solution found so far.

Update Lagrange Multipliers

When an optimal solution for $\mathcal{P}$ is not found, we update μ to find a tighter lower bound on the optimal value of $\mathcal{P}$. This involves a search algorithm for the Lagrangian dual problem. We solve this problem with the *sub-gradient ascent* method [Bertsekas, 1999]. Starting from the initial values to μ, this method systematically produces a sequence of values to Lagrange multiplier $\mu^{i,t}$ as follows:

1. Compute *sub-gradient*

   $$G^{i,t} = x_i^{t-1} - x_i^t - y_0^{i,t} + y_1^{i,t},$$

   for each i and t, which is the LHS of the (2) or the coefficient for $\mu^{i,t}$ in the objective function of $\mathcal{L}$, using the current optimal solutions for the sub-problems.

2. Update $\mu^{i,t}$ for each i and t as

   $$\mu^{i,t} \leftarrow \mu^{i,t} + D \cdot G^{i,t},$$

   where D, called *step length*, is typically computed by

   $$D = \frac{\pi(UB - LB)}{\sum_{i,t}(G^{i,t})^2},$$

   in which π is a scalar parameter gradually reduced from its initial value of 2.

This rule implies that we increase (decrease) $\mu^{i,t}$ if its coefficient $G^{i,t}$ in the objective function of $\mathcal{L}$ is positive (negative), hoping that $L(\mu)$, a lower bound on the optimal value of $\mathcal{P}$, increases in the next round.

Although the sub-gradient ascent method is quite simple, it does not necessarily converge to an optimal solution for $\mathcal{P}$. If the termination condition is not met after convergence, we only have a strict lower bound on the true optimal value of $\mathcal{P}$.

5 Experiments

We compared the performance of the following solvers, each of which is based on one of the two solutions:

- Solvers based on Weighted Partial MaxSAT solving
  - SAT4J: an exact WPMaxSAT solver that uses SAT encoding and a state-of-the-art SAT solver that works better empirically for structured instances [Berre, 2010].
  - WMAXSATZ: an exact WPMaxSAT solver based on the branch and bound method that works better empirically for random instances [Li *et al.*, 2007].
  - IROTS: an incomplete and stochastic WPMaxSAT solver that enhances iterative local search with the tabu list [Stützle *et al.*, 2003].
- Solvers based on Lagrangian decomposition
  - LD: a Lagrangian decomposition method, where each subproblem is solved by SAT4J. A feasible solution is identified at every round by forcing some auxiliary variables to flip in the optimal solutions of the subproblems so that they can satisfy each of the relaxed constraints.
  - LDIROTS: a Lagrangian decomposition method, where each sub-problem is also solved by SAT4J. A feasible solution, on the other hand, is further improved by starting from the one obtained by LD and running IROTS for a constant number of flips. This further improvement is performed only when the best lower bound is updated.

Given a certain time bound, our goal is to see how tight the obtained bounds are. For time-critical dynamic applications, this property should be crucial for any solvers. When a run is cut off at a certain time bound, WPMaxSAT can provide only an upper bound, but Lagrangian decomposition can provide not only an upper bound but also a lower bound.

We solved the following two problems. The first is the random problem, where we generated 30 instances of $(k, X, \phi, f, +, +)$ for each $k \in \{10, 15, \ldots, 35\}$ such that

- X is a set of Boolean variables of size 100, $\{x_1, ..., x_{100}\}$;
- ϕ returns, for each t, a CNF formula randomly selected from the satisfiable instances of `uf100-430` in SATLIB;
- f returns an integer, the cost for changing a variable at a certain time, randomly selected from $\{1, 2, ..., 10^6\}$.

The second problem is the target tracking problem, where 25 sensor stations arranged on a grid with four sensors each must track targets while satisfying the following three constraints: 1) if there is a target in one region, at least two sensors should turn on; 2) if there is no target in one region, no sensor turns on; 3) only one sensor turns on in a sensor station. Given a snapshot of the targets, this problem can be formulated as a SAT instance, where for each sensor, a variable takes 1 when the sensor is on and 0 when the sensor is off. Given also a series of k snapshots that is a sample of possible future moves of the targets, the problem can be formulated as DynSAT. To minimize the numbers of switching sensors from on to off and off to on, the problem can be formulated as DynSAT with decision change costs. More specifically, 30 instances for each $k \in \{10, 15, \ldots, 35\}$ were generated such that

- X is a set of Boolean variables of size 100, $\{x_1, ..., x_{100}\}$;
- ϕ returns, for each t, a satisfiable CNF formula obtained from a snapshot of reasonable moves of targets;
- f always returns the same integer, 10^5.

As the performance measure, we used quality upper bound UB/LB of a feasible solution found by each solver within a specified time bound. Note that LB is the highest lower bound that has been found by LD or LDIROTS. Needless to say, this quality upper bound should be closer to one. Each run was made on an Intel Xeon X5460@3.16 GHz with 4 cores and 32 GB memory. The code was basically written in Java and compiled with JDK 1.6.0_11 on RedHat Enterprise Linux 5 (64 bit). SAT4J and WMAXSATZ were downloaded from the authors' web pages, and their latest versions were run with the default settings. IROTS was from the UBCSAT version 1.1 and was run with the '-w' option. Lagrangian decomposition fits parallel processing very well because, once μ is fixed to some specific values, the decomposed sub-problems are virtually independent. Therefore, we exploited the a multi-core processor in our LD-based solvers to allow sub-problems to be solved in parallel. Furthermore, to make a fair comparison, we also exploited a multi-core processor in other methods by performing portfolio type parallelization.

For the random problem, SAT4J, LD, and LDIROTS never fail to find a feasible solution within a 5-minute time bound. WMAXSATZ, on the other hand, sometimes fails and never finds feasible solutions for sequences of size 35. WMAXSATZ did not work well for these instances because each is actually a highly-structured MaxSAT instance being composed of k random SAT instances sequentially connected through soft clauses. IROTS never found a feasible solution for any k. Only for the solvers that successfully found feasible solutions within five minutes, we plotted the average quality upper bounds in Figure 1(a), where the x-axis is size k of a sequence and the y-axis is the average quality upper bounds. LD and LDIROTS worked very well in these experiments. In Figure 1(b), for 30 sequences of size 35, we also plotted the average quality upper bounds when setting different time bounds over 1, 5, 15, and 30 minutes. This figure clearly shows that, even with increased time bounds, LD and LDIROTS still outperformed SAT4J. Other WPMaxSAT solvers, WMAXSATZ and IROTS, still fail to find a feasible solution even with a 30-minute time bound.

For the target tracking problem, except for WMAXSATZ, the solvers always found feasible solutions within the 5-minute time bound. We plotted the average quality upper bounds for those solvers in Figure 2(a). LDIROTS is competitive with IROTS with all size k's. In Figure 2(b), for 30 sequences of size 35, we also plotted the average quality upper bounds when setting different time bounds over 1, 5, 15, and 30 minutes. Even with increased time bounds, LD-based solvers remain competitive with IROTS. The target tracking problem generally has fewer hard clauses. For such problems, a stochastic solver like IROTS might work. However, looking

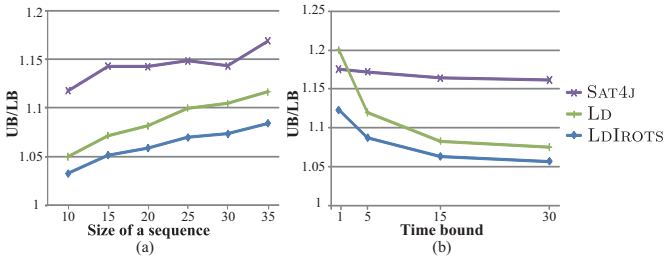

Figure 1: (a) Quality upper bounds vs. size of sequences on random problem (5-minute time bound). (b) Quality upper bounds vs. time bounds on random problem (sequences of size 35)

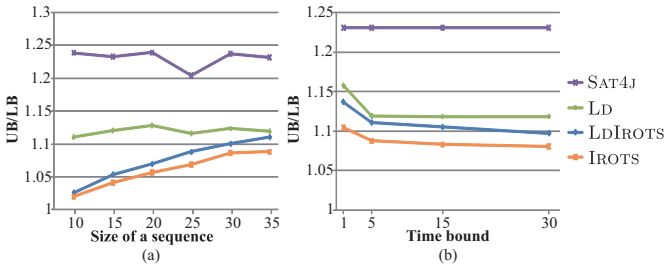

Figure 2: (a) Quality upper bounds vs. size of sequences on target tracking problem (5-minute time bound).(b) Quality upper bounds vs. time bounds on target tracking problem (sequences of size 35)

at the results for the random problem, its performance is far from robust.

6 Conclusions

In this work, we provided a DynSAT with decision change costs and two solutions, Weighted Partial MaxSAT solving and Lagrangian decomposition, for solving a specific case of this problem. Among these solutions, only Lagrangian decomposition provided lower bounds for the problem. Furthermore, a solver based on Lagrangian decomposition, LDIROTS, seems very promising since it worked very well empirically to find a quality feasible solution even when a time bound is tight. For dynamically changing problems, this property is crucial.

References

[Bailleux and Marquis, 2006] Olivier Bailleux and Pierre Marquis. Some computational aspects of distance-SAT. *Journal of Automated Reasoning*, 37(4): 231–260, 2006.

[Berre, 2010] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation 7*, pages 59–64, 2010.

[Bertsekas, 1999] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[Bessière, 1991] Christian Bessière. Arc-consistency in dynamic constraint satisfaction problems. *AAAI-91*, pages 221–226, 1991.

[Bofill *et al.*, 2010] Miquel Bofill, Didac Busquets and Mateu Villaret. A Declarative Approach to Robust Weighted Max-SAT. *12th Intl. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, pages 67–76, 2010.

[Dechter and Dechter, 1988] Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint networks. *AAAI-88*, pages 37–42, 1988.

[Fargier *et al.*, 1996] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. *AAAI-96*, pages 175–180, 1996.

[Freuder, 1991] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. *AAAI-91*, pages 227–233, 1991.

[Ginsberg *et al.*, 1998] Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and robustness. *AAAI-98*, pages 334–339, 1998.

[Hebrard *et al.*, 2005] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. *AAAI-05*, pages 372–377, 2005.

[Hoos and O'Neill, 2000] Holger H. Hoos and Kevin O'Neill. Stochastic local search methods for dynamic SAT – an initial investigation. *AAAI-00 Workshop: Leveraging Probability and Uncertainty in Computation*, pages 22–26, 2000.

[Stützle *et al.*, 2003] Thomas Stützle, Kevin Smyth, Holger H. Hoos. Iterated robust tabu search for MAX-SAT. *AI-03*, pages 129–144, 2003.

[Li *et al.*, 2007] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for MAX-SAT. *Journal of Artificial Intelligence Research*, 30: 321–359, 2007.

[Ran *et al.*, 2002] Yongping Ran, Nico Roos, and Jaap van den Herik. Approaches to find a near-minimal changes solution for dynamic CSPs. *CP-AI-OR-02*, 2002.

[Schiex and Verfaillie, 1993] Thomas Schiex and Gérard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *ICTAI-93*, pages 48–55, 1993.

[Verfaillie and Jussien, 2005] Gérard Verfaillie and Narendra Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10: 253–281, 2005.

[Verfaillie and Schiex, 1994] Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. *AAAI-94*, pages 307–312, 1994.

[Wallace and Freuder, 1998] Richard J. Wallace and Eugene C. Freuder. Stable solutions for dynamic constraint satisfaction problems. *CP-98*, pages 447–461, 1998.

[Walsh, 2002] Toby Walsh. Stochastic constraint programming. *ECAI-02*, pages 111–115, 2002.