Some Computational Aspects of DISTANCE-SAT *

Olivier Bailleux (olivier.bailleux@u-bourgogne.fr)
LERSIA/Université de Bourgogne
BP 47870
F-21078 Dijon cedex - France

Pierre Marquis (marquis@cril.univ-artois.fr)
CRIL-CNRS/Université d'Artois
rue de l'Université - S.P. 16
F-62307 Lens Cedex - France

Abstract. In many AI fields, the problem of finding out a solution which is as close as possible to a given configuration has to be faced. This paper addresses this problem in a propositional framework. The decision problem DISTANCE-SAT which consists in determining whether a propositional formula admits a model that disagrees with a given partial interpretation on at most d variables, is introduced. The complexity of DISTANCE-SAT and of several restrictions of it are identified. Two algorithms based on the well-known Davis/Logemann/Loveland search procedure for the satisfiability problem SAT are presented so as to solve DISTANCE-SAT for CNF formulas. Their computational behaviours are compared with the ones offered by SAT solvers on SAT encodings of DISTANCE-SAT instances. The empirical evaluation allows for drawing firm conclusions about the respective performances of the algorithms, and to relate the difficulty of DISTANCE-SAT with the difficulty of SAT from the practical side.

Keywords: Satisfiability, computational complexity.

1. Introduction

1.1. Motivations

In many AI fields, the problem of finding out a solution which is as close as possible to a given configuration must be faced. Such a configuration may encode some preferential situation (e.g., an expected state, or a normal state) which conflicts with the hard constraints of the problem, represented as a knowledge base. It may also represent some observations conflicting with a set of possible worlds, concisely encoded as a knowledge base.

For instance, in the consistency-based diagnosis framework (Reiter, 1987), the expected state of each component of a device is "non-faulty".

* A preliminary version of this paper appeared with the title "DISTANCE-SAT: Complexity and Algorithms" in the proceedings of the 16^{th} National Conference on Artificial Intelligence (AAAI'99), pages 642-647, 1999.

Whenever a failure occurs, such a diagnosis is not possible anymore: assuming that every component behaves as its model of correct behaviour requires it conflicts with the observations which have been made. In this situation, the normality assumption must be revised (some components are to be assumed faulty) so as to restore consistency. Since many fault assumptions can typically be made in order to achieve this goal, a principle of parsimony is often adopted: among the possible diagnoses, the selected ones are those including a minimal set (w.r.t. cardinality or set-inclusion) of faulty assumptions. Thus, the diagnoses that are "not so far" from the normal state of the device are preferred to the remaining ones: one-fault diagnoses are first considered, then two-faults diagnoses, and so on.

In this paper, the problem of computing a solution as close as possible to a given configuration is addressed within a propositional framework. Beyond the diagnosis field, this problem and its direct by-product (computing the minimum "distance" between a configuration and a knowledge base encoding the set of its models) appear as key operations in many AI computational tasks where propositional reasoning is involved; let us mention belief change and update, product configuration, similarity-based reasoning, non-classical planning, group decision making, optimization, and some forms of common-sense reasoning. Thus, in Forbus' approach to belief update (Forbus, 1989), a preferred model of the updated belief state is a model of the update formula which is as close as possible to a model of the original base w.r.t. Hamming distance. Computing the Hamming distance between an interpretation and its closest models among those of a knowledge base also is a basic operation for some belief revision operators (especially, Dalal's one (Dalal, 1988)) and some belief merging operators (Konieczny and Pérez, 1998). In the interactive solving of product configuration, the distance between a given configuration and a set of feasible products (represented in an implicit way as the models of a knowledge base) tells how many user's choices must be given up so as to render the configuration feasible. In similarity-based reasoning, the greatest distance between a set of interpretations encoding a first formula and a second formula indicates to what extent the former approximately entails the latter (Dubois *et al.*, 1997). A relaxation of the classical planning problem can also be obtained by considering that a plan is valid whenever it leads to a state which is close enough to a goal state; in such a setting, the configuration represents the goal state and the distance indicates how close to a goal state a final state has to be in order to be considered acceptable.[1] In group

[1] We shall return on this application in Section 6.2.

decision making (Lafage and Lang, 2000) (Lafage and Lang, 2001), the disutility of a world can be defined and computed as the aggregation of the "distances" between that world and some formulas representing the various preferences of the agents of the group. In cardinality-based circumscription (Liberatore and Schaerf, 1997), in absence of fixed variables, a preferred model of a knowledge base is a model of it in which a minimal number of abnormality variables are set to true. Designing algorithms for computing the models of a knowledge base as close as possible to a given interpretation can be viewed as a first step towards the implementation of propositional morphological operators (like the dilation and erosion ones) (Bloch and Lang, 2000). The problem studied in this paper also appears as a fundamental SAT-based optimization problem (Thiébaux *et al.*, 2000); especially, the well-known MAX-SAT problem can be polynomially reduced to it (Slaney and Walsh, 2002). Finally, this problem is closely related to the problem of repairing a supermodel (Ginsberg *et al.*, 1998) (Roy and Wilson, 2000).

1.2. Scope and organization of the paper

Let us first make precise what a knowledge base and a configuration mean in a propositional setting. In the following, a knowledge base is represented as a propositional formula Σ, a configuration as a partial interpretation PI, and we are interested in finding out a model of Σ that disagrees with PI on at most d variables. DISTANCE-SAT consists in determining whether or not such a model exists.

The contribution of this paper is twofold. On the one hand, the complexity of DISTANCE-SAT is investigated in the general case and in some restricted cases. Like the well-known satisfiability problem SAT (which can be viewed as a restriction of it), DISTANCE-SAT is NP-complete. However, DISTANCE-SAT is somewhat more difficult than SAT, in the sense that tractable restrictions for SAT do not always give rise to tractable restrictions for DISTANCE-SAT.

On the other hand, two algorithms for solving DISTANCE-SAT for CNF formulas are introduced. The first one, `DLL-distance`, is a straightforward adaptation of the well-known Davis/Logemann/Loveland search procedure (Davis *et al.*, 1962) for SAT. To every node of the search tree is associated a value that measures the disagreement between the given configuration and the partial interpretation that corresponds to the node (and can be read off directly by picking up the literals from the branch that ends up to the node under consideration). Whenever this value exceeds the given maximal bound d, the algorithm backtracks. Our second algorithm, `DLL-lasso`, is a variant of `DLL-distance`. The only difference between

them lies in the branching rule. While the branching rule used in `DLL-distance` is a standard, "efficient", branching rule for SAT, the branching rule used in `DLL-lasso` is much more oriented towards the satisfaction of the distance constraint. The objective is to lasso in priority a model that is close to the given configuration. Thus, among the clauses that are fully falsified by the given configuration, those of minimal length are considered. Among the variables of these clauses, one of those that maximize the standard branching rule heuristic (used in `DLL-distance`) is selected as the branching variable.

Both algorithms are empirically assessed on many random 3-CNF instances (generated using the now classical "fixed-length clauses model" (Chvátal and Szemerédi, 1988)), for several values of the ratio number of clauses/number of variables, for several sizes of the given configuration, and several values for the maximal disagreement number d. When d is small and all the variables are assigned by the given configuration, `DLL-lasso` performs much better than `DLL-distance`. Contrastingly, when d is large, `DLL-distance` is the best performer.

Those two algorithms are compared with a full-CNF approach to DISTANCE-SAT. An instance of DISTANCE-SAT is reformulated as a SAT instance, which can be carried out by any SAT solver. Compared with `DLL-distance` and `DLL-lasso`, this full-CNF approach leads to a more efficient way to solve some DISTANCE-SAT instances, especially when some variables are not fixed in the given configuration.

Those algorithms have also been tested on some additional instances, including hand-crafted ones coming from a non-classical planning problem and the parity learning problem, as well as larger random 3-CNF instances. Three additional SAT solvers (two efficient ones and a naive one) have been used downstream to the full-CNF approach. Such further experiments have shown the feasibility of solving "realistic" instances of DISTANCE-SAT; they also confirm that, while they are based on a naive DLL procedure, the two specialized algorithms `DLL-distance` and `DLL-lasso` prove practically useful.

The rest of this paper is organized as follows. Section 2 gives some formal preliminaries. Section 3 presents DISTANCE-SAT and its computational complexity in the general case, and in some restricted cases. Our algorithms `DLL-distance` and `DLL-lasso` are successively introduced in Section 4. Section 5 presents the full-CNF approach for solving DISTANCE-SAT instances using the encoding given in (Bailleux and Boufkhad, 2003). Section 6 presents an empirical evaluation. Section 7 concludes this paper. Proofs are reported in an appendix.

2. Formal Preliminaries

Let $PROP_{PS}$ denote the propositional language built up from a finite set PS of propositional symbols (also called variables), the Boolean constant $true$ and $false$, and the connectives in the standard way. The elements of $PROP_{PS}$ are called *formulas*. The *size* of a formula Σ, noted $|\Sigma|$ is the number of signs (symbols and connectives) used to write it. $Var(\Sigma)$ is the *set of propositional variables* occurring in Σ. Among the formulas of $PROP_{PS}$ are the *CNF* formulas and the *DNF* ones.

Formulas are interpreted in the classical way. An *interpretation* (or truth assignment) of a formula Σ is a mapping I that associates *every* propositional variable of PS to one of the two truth values of $BOOL = \{0, 1\}$. A *partial* interpretation of Σ is a mapping PI that associates *some* propositional variables of PS to one of the two truth values of $BOOL$. $Dom(PI) \subseteq PS$ denotes the *domain* of PI, and $|PI|$ the cardinal of $Dom(PI)$. A *complete* partial interpretation is just an interpretation. In the following, (partial) interpretations are represented as sets of literals. A positive literal x (resp. a negative literal $\neg x$) appears in PI if and only if $PI(x) = 1$ (resp. $PI(x) = 0$). An interpretation I is an *extension* of a partial interpretation PI if and only if $PI \subseteq I$ holds. A clause is said to be *fully falsified* by a partial interpretation whenever every literal of the clause appears in the partial interpretation with the opposite sign.

Many valuable subsets (or fragments) of $PROP_{PS}$ can be defined. Thus, a *k-CNF* formula is a CNF formula s.t. every clause in it contains at most k literals. A formula is *Horn CNF* (resp. *reverse Horn CNF*) if and only if it is a CNF formula s.t. every clause in it contains at most one positive (resp. negative) literal. A *Krom* formula is a 2-CNF formula, i.e., every clause in it contains at most two literals. A *Blake* formula is a CNF formula Σ consisting of the conjunction of all its prime implicates, i.e., the logically strongest clauses entailed by Σ (one representative per equivalence class is kept, only). Contrariwise to the k-CNF language (under the standard assumption $\mathsf{P} \neq \mathsf{NP}$), the Horn CNF, reverse Horn CNF, Krom and Blake fragments are tractable for SAT: for any of those fragments, there exists a polytime algorithm for checking whether any formula from the fragment is satisfiable or not.

Another important fragment tractable for SAT is the *DNNF* language, defined as follows (Darwiche, 1999; Darwiche, 2001): a sentence in DNNF is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with $true$, $false$, x or $\neg x$, $x \in PS$; each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children. Moreover, the *decomposability* property is satisfied: for each conjunction C in the

sentence, the conjuncts of C do not share variables. Interestingly, the DNNF language includes as proper subsets two influential propositional fragments, namely the DNF one and the ROBDD one (see (Darwiche, 2001; Darwiche and Marquis, 2001)).

In contrast to the k-CNF, Horn CNF, reverse Horn CNF, Krom fragments, the DNNF, DNF, ROBDD and the Blake ones are (functionally) complete, which means that for every propositional formula Σ, there exists an equivalent formula Φ belonging to the fragment.

In the following, we assume the reader familiar with some basic notions of computational complexity, especially NP-completeness (see e.g., (Garey and Johnson, 1979)).

3. Definition and Complexity

Before defining DISTANCE-SAT in a formal way, we first need the definition of disagreement between two partial interpretations:

DEFINITION 1 (disagreement).
A partial interpretation PI_1 is said to disagree *with a partial interpretation PI_2 on at most d variables if and only if the number of variables x of $Dom(PI_1) \cap Dom(PI_2)$ s.t. $PI_1(x) \neq PI_2(x)$ is less than or equal to d.*

We are now ready to define DISTANCE-SAT.

DEFINITION 2 (DISTANCE-SAT).
DISTANCE-SAT *is the following decision problem:*

- **Input:** *A formula Σ, a partial interpretation PI, and a non-negative integer d.*
- **Question:** *Does there exist a model I of Σ s.t. I disagrees with PI on at most d variables?*

For every instance of DISTANCE-SAT, we call the constraint "I disagrees with PI on at most d variables" its *distance constraint*. Its strength diminishes as d decreases and as $|PI|$ increases. In particular, the distance constraint does not impose any restriction over the models of Σ whenever $|PI| \leq d$: under this requirement, Σ is a positive instance of SAT if and only if $\langle \Sigma, PI, d \rangle$ is a positive instance of DISTANCE-SAT.

Since we are interested in solving DISTANCE-SAT and the corresponding function problem, it is important to identify the computational complexity of DISTANCE-SAT. We did it in the general case and in several restricted cases.

Table I. The complexity of DISTANCE-SAT
Any PI.

KB	**any** d	**a fixed** d
any Σ	NP-complete	NP-complete
Σ CNF	NP-complete	NP-complete
Σ Horn	NP-complete	in P
Σ reverse Horn	NP-complete	in P
Σ Krom	NP-complete	in P
Σ Blake	NP-complete	in P
Σ DNNF	in P	in P

PROPOSITION 1 (complexity of DISTANCE-SAT).
The complexity of DISTANCE-SAT *and of several restrictions of it obtained by considering:*

- *a knowledge base* Σ *from various propositional fragments that are* tractable *for* SAT*,*
- *a* complete *partial interpretation* PI*,*
- *a* fixed *maximal distance* d

are reported in Tables I and II.

Clearly enough, SAT, the satisfiability problem of a CNF formula is a restriction of DISTANCE-SAT (taking $PI = \emptyset$ (or $d = n$) so as to reduce SAT to DISTANCE-SAT is sufficient to prove the NP-hardness of DISTANCE-SAT). Hence, it is not surprising that DISTANCE-SAT is intractable in the general case, i.e., there is no known polynomial algorithm to solve it (and there can be no such algorithm unless P = NP). Nevertheless DISTANCE-SAT is not much more difficult than SAT since it belongs to NP. Indeed, verifying that a guessed interpretation disagrees with PI on at most d variables can easily be achieved in polynomial time.

It can be noted that for some fragments for which SAT is tractable, DISTANCE-SAT is tractable as well. Especially, it is the case for the DNNF fragment. Actually, very simple algorithms can be designed for solving DISTANCE-SAT for proper subsets of DNNF, as DNF and

Table II. The complexity of DISTANCE-SAT
A complete PI.

KB	**any** d	**a fixed** d
any Σ	NP-complete	in P
Σ CNF	NP-complete	in P
Σ Horn	NP-complete	in P
Σ reverse Horn	NP-complete	in P
Σ Krom	NP-complete	in P
Σ Blake	NP-complete	in P
Σ DNNF	in P	in P

ROBDD. On the one hand, since each model of a DNF Σ is an extension of at least one term (viewed as a partial interpretation) of Σ and the converse also holds, $\langle \Sigma, PI, d\rangle \in$ DISTANCE-SAT if and only if there exists a term t_i of Σ s.t. $\langle t_i, PI, d\rangle \in$ DISTANCE-SAT; then, Definition 1 clearly shows that determining how much a term (viewed as a partial interpretation) disagrees with a given partial interpretation can be achieved in polynomial time. On the other hand, solving DISTANCE-SAT given a ROBDD formula Σ amounts to searching for a minimal-cost path in a 0/1 weighted digraph (just label every arc of Σ with 0, except those associated to a literal l s.t. $\neg l \in PI$, which are labeled with 1).

However, focusing on the standard fragments of propositional logic where SAT is known as tractable is not sufficient to ensure the (time) polynomiality of DISTANCE-SAT in the general case. Both NP-hardness of the restrictions where Σ is Horn, reverse Horn or Krom are consequences of the NP-hardness of DISTANCE-SAT under the restriction when Σ is a 2-CNF monotone formula, i.e., every literal of Σ has only either positive occurrences or negative occurrences in Σ. Since every monotone CNF formula is satisfiable, the complexity of DISTANCE-SAT does not come *solely* from the complexity of the satisfiability issue for its input Σ but from the interaction between Σ and the distance constraint. From this point of view, DISTANCE-SAT can be considered as at least as difficult as SAT.

As the previous proposition shows it, focusing on KBs belonging to usual tractable fragments for SAT is sufficient to obtain tractable restrictions of DISTANCE-SAT as long as d is considered as a fixed

constant. Some other restrictions can be considered so as to achieve tractability. Thus, while imposing PI to be a complete interpretation does not lower the complexity of DISTANCE-SAT in the general case (even when Σ is known as tractable for SAT), determining whether Σ has a model that disagrees with a complete interpretation I on at most d variables, where d is a constant, is in P. To be more precise, if K is the maximal length of clauses of Σ, there exists a $\mathcal{O}(|\Sigma| \times K^d)$ time (deterministic) algorithm that solves this last problem (cf. Section 4).

Interestingly, as a by-product of Proposition 1, some new results about the complexity of the satisfiability issue for some extended propositional languages can be derived. Given a propositional language $PROP_{PS}$, let a *cardinality constraint* be an ordered pair $\langle\{l_1, ..., l_k\}, m\rangle$, where each l_i ($i \in 1..k$) is a literal of $PROP_{PS}$ and m is a non-negative integer that is less than or equal to k. Given an interpretation I, the semantics of such a cardinality constraint in I is 1 if and only if *at least* m literals from $\{l_1, ..., l_k\}$ belong to I (i.e., are interpreted as 1 in I as well). A *cardinality formula* is a (finite) conjunction of cardinality constraints (Benhamou *et al.*, 1994) (Van Henteryck and Deville, 1991).

Clearly enough, expressing that we are looking for a model of Σ that disagrees with $PI = \{l_1, ..., l_k\}$ on at most d variables amounts to looking for a model of the formula obtained by adding to the clauses of Σ the single cardinality constraint $\langle\{l_1, ..., l_k\}, k-d\rangle$. Many more clauses, but a polynomial number of it, are required to reduce DISTANCE-SAT to SAT in the general case.[2] Such a reduction underlies what we called the full-CNF approach to DISTANCE-SAT; it is presented in Section 5. As a direct consequence of Proposition 1, we obtain the following corollary:

COROLLARY 3.1. *Determining whether a cardinality formula Σ is satisfiable is* NP*-complete, even when Σ contains only (classical) clauses (i.e., with $m = 1$) that form a Horn CNF formula (or a reverse Horn CNF formula or a Krom one), plus one cardinality constraint with $m \neq 1$.*

Obviously enough, this corollary applies as well to linear pseudo-Boolean formulas (see e.g., (Bockmayr, 1995)) since every cardinality constraint is a linear pseudo-Boolean constraint; thus, the satisfiability of a set of Horn (resp. reverse Horn, Krom) clauses given a single linear pseudo-Boolean constraint is an NP-complete problem.

[2] This comes from the fact that SAT is NP-complete: every problem in NP can be polynomially reduced to it.

4. Two Algorithms for DISTANCE-SAT

In this section, two algorithms for DISTANCE-SAT in the CNF case are introduced. These algorithms are based on the standard Davis / Logemann / Loveland search procedure for SAT (Davis *et al.*, 1962). This choice is motivated by the two following facts:

- A naive approach that would consist in enumerating in a successive way the interpretations that do *not* disagree with PI on at most d variables is not computationally feasible in the general case, even for quite small values of n, the number of variables of Σ, and d. For instance, with $n = 100$, $d = 10$ and PI is any complete interpretation, more than 10^{13} interpretations should be considered, which makes such a naive enumerative technique far from being practical.
- The most effective algorithms for SAT one can find in the literature are typically based on the Davis / Logemann / Loveland search procedure, and SAT is a restriction of DISTANCE-SAT. Especially, if $\Sigma \notin$ SAT, then $\forall PI \forall d, \langle \Sigma, PI, d \rangle \notin$ DISTANCE-SAT.

Our first algorithm, `DLL-distance`, mainly is the standard Davis/ Logemann / Loveland search procedure, equipped with a counter that indicates for every node of the search tree the number of variables on which the partial interpretation associated to that node disagrees with the given configuration. As soon as the value of the counter exceeds d, the algorithm backtracks.

Procedure $DP_{distance} : BOOLEAN$
Input : an instance $\langle \Sigma, PI, d \rangle$ of DISTANCE-SAT.
Output : *true* if and only if $\langle \Sigma, PI, d \rangle \in$ DISTANCE-SAT.
Begin
unit_propagate(Σ);
if `disagree(`PI_C`, PI) > d` **then return** `(`$false$`);`
if the empty clause is generated **then return** ($false$);
else if all clauses are satisfied **then return** ($true$)
else begin
x **:=** $branching(\Sigma, PI_C)$;
return ($DP_{distance}(\Sigma \wedge x)$ or
$DP_{distance}(\Sigma \wedge \neg x)$);
end;
End

In this algorithm, PI_C is the current partial interpretation, i.e., the one associated to the current node of the search tree. PI_C gathers all

the variables that have been fixed from the root of the tree to the current node. *unit_propagate* is a function that performs unit-propagation through Σ. PI_C is updated by *unit_propagate*.

It is well-known that the design of a branching rule is a critical factor in the performance of any Davis/Logemann/Loveland-like algorithm for SAT (without learning). Our *branching* function implements the branching rule given in (Dubois *et al.*, 1996), one of the best performer for SAT. To be more precise, the weight of a literal l of a CNF formula Σ is given by $w(l) = \sum_{\forall \gamma \in \Sigma, l \in \gamma} -log_2(1 - 1/(2^{|\gamma|} - 1)^2)$ and the score of a variable x by $s(x) = w(x) + w(\neg x) + 1.5min(w(x), w(\neg x))$. A variable maximizing s is elected as the branching variable.

Clearly enough, `DLL-distance` is very close to the standard DLL procedure. Actually, the unique difference between them is the additional backtrack instruction that is triggered as soon as the current partial interpretation PI_C disagrees with PI on more than d variables. The distance constraint is not exploited in an aggressive way, but only in a passive way.

Our second algorithm `DLL-lasso` is a variant of `DLL-distance` in which the *branching* function that is used does not correspond to a standard branching rule for SAT but has been especially tailored for DISTANCE-SAT. The purpose is to take advantage of both the best branching rules available for SAT but also to exploit the distance constraint much more aggressively than in `DLL-distance`. The variables that appear in the set S_{PI_C} of the clauses of Σ simplified by PI_C that are fully falsified by PI, and are of minimal size, are filtered. Then, the weights of these variables are computed using the same weight function as in *branching*, and a variable with a maximal weight is elected. If S_{PI_C} is empty then the branching variable is selected according the same heuristic as `DLL-distance`. The idea of choosing the branching variable among the variables which occur in clauses that are falsified by a reference interpretation already appeared in SCORE(FD/B), a local search based complete algorithm for SAT (Chabrier *et al.*, 1995).

Let γ be a clause of S_{PI_C} and x a variable of γ. When x is assigned the sign it has in γ, γ becomes satisfied by the updated partial interpretation PI_C. When x is given the opposite sign, the resulting simplified clause (i.e., γ in which the literal corresponding to x has been removed) is still fully falsified by PI, and necessarily is of minimal size. This forces the remaining variables of γ to be among the candidate variables for branching at the next choice node. Interestingly, whenever PI is a complete interpretation and the size of the longest clause of Σ is bounded by a constant K, only $\mathcal{O}(K^d)$ choice nodes will be generated by `DLL-lasso`, provided that a variable of S_{PI_C} is always elected as the branching variable. We call such a property the *lasso effect*.

PROPOSITION 2 (lasso effect). *Let* $\langle \Sigma, PI, d \rangle$ *the input of* `DLL-lasso`*, where* PI *is a complete interpretation. Let* K *be the number of literals in the largest clause of* Σ*. The run time of* `DLL-lasso` *is* $\mathcal{O}(|\Sigma| \times K^d)$.

When the lasso effect works (i.e., when PI is complete), the number of clauses occurring in Σ influences the computational performance of `DLL-lasso` by a *linear* factor, only. This is far from being expected for `DLL-distance`.

The design of the $branching_{lasso}$ function is done so as to take advantage of both the lasso effect (considering only the variables of S_{PI_C}), and the best branching rules for SAT (the variables are ordered so as to select one that maximizes a standard weight function).

Clearly enough, since the lasso technique simply consists in filtering out some candidate variables before applying to them any branching function, some other branching functions for SAT can be used, giving rise to additional branching functions for DISTANCE-SAT. Moreover, the distance constraint can be exploited in a more integrated way within such branching functions for DISTANCE-SAT, especially for the propagation-based ones, like the one used in `satz` (Li and Anbulagan, 1997). Propagating a literal through a CNF formula results in a partial interpretation (encoding the literals that have been fixed) and a corresponding simplified CNF formula. The value of the disagreement between such a partial interpretation and the reference one, and the tightness of the associated simplified formula are two parameters that can be used to evaluate heuristically whether propagating a literal is promising for DISTANCE-SAT.

Finally, let us note that both `DLL-distance` and `DLL-lasso` can be easily modified to address the function problem associated to DISTANCE-SAT, i.e., to return a model of Σ that disagrees with PI on at most d variables whenever such a model exists. Instead of returning *true* when an implicant of Σ is found, it is sufficient to return any extension of the current partial interpretation PI_C.

5. The full-CNF approach

In this section, we briefly explain how DISTANCE-SAT in the CNF case can be reduced to SAT. Such a reduction underlies a family of algorithms to which our own algorithms `DLL-distance` and `DLL-lasso` have to be compared.

First, let us note that there are several ways to encode cardinality constraints (including distance constraints) as CNF formulas. We

used the approach introduced in (Bailleux and Boufkhad, 2003), which proves more efficient that usual Warners' encoding (Warners, 1998), since it exploits unit propagation in order to restore the generalized arc consistency property on the encoded constraints.

Maintaining generalized arc consistency (Bessiere and Regin, 2001) is a filtering technique, widely used for solving constraint satisfaction problems; this technique consists in removing of the domain of each variable any value which cannot belong to any solution because every solution which includes it violates at least one constraint. In the DISTANCE-SAT setting, maintaining generalized arc consistency consists in applying the following rules at each node of the search tree: Let d be the distance and k be the number of variables fixed in the partial current interpretation PI_C that disagree with the reference interpretation PI. If $k > d$ then backtrack (because of inconsistency of the distance constraint). If $k = d$ then fix each free variable in PI_C with the corresponding value in PI.

Observe that `DLL-distance` and `DLL-lasso` do not explicitly maintain generalized arc consistency, but they do it implicitly whenever the reference interpretation PI is complete; indeed, when $d = k$, one of the two values of each branching variable induces an immediate backtrack.

The encoding we considered is denoted UT-MGAC (for Unit Totalizer Maintaining Generalized Arc Consistency). It is based on a unary representation of integer intervals. Namely, a set of Boolean variables $x_0, \ldots, x_{\max}$ is used to represent any interval $\mu, \ldots, \lambda$ in the range $0, \ldots, \max$ by setting $x_0, \ldots, x_\mu$ to 1 and $x_{\lambda+1}, \ldots, x_{\max}$ to 0.

An adder based on this representation can be used to derive the interval where an integer c is located, given the intervals of integers a and b s.t. $c = a + b$. The resulting CNF formula allows unit propagation for deriving all the consequences w.r.t. generalized arc consistency of assigning truth values to variables. The cardinality constraint is then encoded as a totalizer structured as a pyramidal adder network, extended by unary clauses which restrict the possible output values.

The encoding clauses and additional encoding variables can be generated as shown on Figure 1, where each adder with inputs $a_1, \ldots, a_{m_1}$, $b_1, \ldots, b_{m_2}$, and output $r_1, \ldots, r_m$ is encoded as

$$\bigwedge_{\substack{0 \le \alpha \le m_1 \\ 0 \le \beta \le m_2 \\ 0 \le \sigma \le m \\ \alpha+\beta=\sigma}} (C_1(\alpha, \beta, \sigma) \wedge C_2(\alpha, \beta, \sigma)) \tag{1}$$

using the following notations:

$$a_0 = b_0 = r_0 = 1, a_{m_1+1} = b_{m_2+1} = r_{m+1} = 0$$

$$C_1(\alpha, \beta, \sigma) = (\overline{a_\alpha} \vee \overline{b_\beta} \vee r_\sigma),\ C_2(\alpha, \beta, \sigma) = (a_{\alpha+1} \vee b_{\beta+1} \vee \overline{r_{\sigma+1}})$$

Given an integer a in unary notation, if the bit a_α is equal to 1, then a is such that $a \geq \alpha$ and, conversely, if $a_\alpha = 0$ then we have $a < \alpha$. On this ground, the clause $(\overline{a_\alpha} \vee \overline{b_\beta} \vee r_\sigma)$, which is called of type C_1, ensures that at least one of the three following inequalities holds: $a < \alpha,\ b < \beta,\ r \geq \alpha+\beta$; the clause $(a_{\alpha+1} \vee b_{\beta+1} \vee \overline{r_{\sigma+1}})$, which is called of type C_2, ensures that at least one of the three following inequalities is true : $a > \alpha,\ b > \beta,\ r \leq \alpha + \beta$.

Now, let n be the number of input and output variables of the totalizer, and let Q be the cardinality constraint $\mu \leq\ N \leq \lambda$, where N is the number of input variables of the totalizer which can be fixed to 1 according to Q. Q is ensured by additional unit clauses enforcing the output variables to match the interval $\mu \ldots \lambda$. Namely,

$$\bigwedge_{1 \leq i \leq \mu} (s_i) \bigwedge_{\lambda+1 \leq j \leq n} (\overline{s_j}) \tag{2}$$

As proved in (Bailleux and Boufkhad, 2003), given any partial truth assignment of the input variables, altogether the C_1 and C_2 clauses allow unit propagation for restoring the generalized arc consistency of Q.

EXAMPLE 1. *Let $T(\langle i_1, \ldots, i_n \rangle, \langle o_1, \ldots, o_n \rangle)$ denote the formula encoding a totalizer with input variables $i_1, \ldots, i_n$ and output variables $o_1, \ldots, o_n$. Let $A(\langle a_1, \ldots, a_{m_1} \rangle, \langle b_1, \ldots, b_{m_2} \rangle, \langle r_1, \ldots, r_{m_1+m_2} \rangle)$ denote the formula encoding an adder with input variables $a_1, \ldots, a_{m_1}, b_1, \ldots, b_{m_2}$ and output variables $r_1, \ldots, r_{m_1+m_2}$.*

Let us consider a cardinality constraint Q over 3 variables i_1, i_2, i_3 s.t. $N \leq 2$. The encoding of Q involves a totalizer $T(\langle i_1, i_2, i_3 \rangle, \langle o_1, o_2, o_3 \rangle)$, completed by the clause $(\overline{o_3})$.

The totalizer $T(\langle i_1, i_2, i_3 \rangle, \langle o_1, o_2, o_3 \rangle)$ includes a totalizer $T(\langle i_1, i_2 \rangle, \langle u_1, u_2 \rangle)$, where u_1, u_2 are additional encoding variables, and an adder $A(\langle u_1, u_2 \rangle, \langle i_3 \rangle, \langle o_1, o_2, o_3 \rangle)$. Because it has only two input variables, the totalizer $T(\langle i_1, i_2 \rangle, \langle u_1, u_2 \rangle)$ includes only one adder $A(\langle i_1 \rangle, \langle i_2 \rangle, \langle u_1, u_2 \rangle)$.

Then we have
$T(\langle i_1, i_2, i_3 \rangle, \langle o_1, o_2, o_3 \rangle) =$
$$A(\langle i_1 \rangle, \langle i_2 \rangle, \langle u_1, u_2 \rangle) \wedge A(\langle u_1, u_2 \rangle, \langle i_3 \rangle, \langle o_1, o_2, o_3 \rangle).$$

The corresponding CNF formulas are
$A(\langle i_1 \rangle, \langle i_2 \rangle, \langle u_1, u_2 \rangle) =$
$$(\overline{i_1} \vee u_1) \wedge (i_2 \vee \overline{u_2}) \wedge (\overline{i_2} \vee u_1) \wedge (i_1 \vee \overline{u_2}) \wedge (\overline{i_1} \vee \overline{i_2} \vee u_2) \wedge (i_1 \vee i_2 \vee \overline{u_1})$$

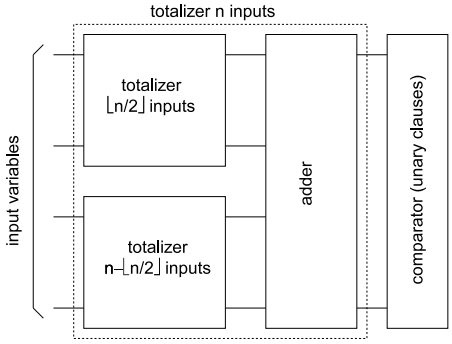

Figure 1. UT-MGAG encoding scheme for Boolean cardinality constraints.

and
$A(\langle u_1, u_2\rangle, \langle i_3\rangle, \langle o_1, o_2, o_3\rangle) =$
$(\overline{i_3} \vee o_1) \wedge (u_2 \vee \overline{o_3}) \wedge (\overline{u_2} \vee o_2) \wedge (i_3 \vee \overline{o_3}) \wedge (u_1 \vee \overline{o_2}) \wedge (\overline{u_1} \vee o_1) \wedge$
$(\overline{u_1} \vee \overline{i_3} \vee o_2) \wedge (\overline{u_2} \vee \overline{i_3} \vee o_3) \wedge (i_1 \vee i_3 \vee \overline{o_1}) \wedge (u_2 \vee i_3 \vee \overline{o_2})$

Now suppose that the variables i_1 and i_3 are fixed to 1*. Unit propagation will set u_1 to* 1 *through the clause $(\overline{i_1} \vee u_1)$, u_2 to* 0 *through the clause $(\overline{u_2} \vee \overline{i_3} \vee o_3)$, then i_2 to* 0 *through the clause $(\overline{i_1} \vee \overline{i_2} \vee u_2)$.*
This last deduction restores the generalized arc consistency on Q.

This encoding requires $\Theta(n^2)$ clauses and $\Theta(n \log n)$ additional variables. Without questioning the correctness and filtering properties, its effective size can be optimized by bounding to $\lambda + 1$ the number of output variables of any adder in the totalizer.

6. Empirical Evaluation

This section is divided into two parts. In the first part, we report on an empirical evaluation of three algorithms for DISTANCE-SAT: `DLL-distance` and `DLL-lasso`, both based on the same, naive, non-optimized implementation of a DLL algorithm, and `BerkMin-full-CNF-encoding`, the full-CNF approach using a state-of-the-art DLL algorithm, the `BerkMin561` solver (Goldberg and Novikov, 2002). In the second part, we report on some additional, yet less systematic, experiments obtained using three other SAT solvers: `Minisat` (Eén and Sörensson, 2004), `JeruSat` (Nadel, 2002), and `dll-basic`, i.e., the naive, non-optimized DLL implementation that is the core of `DLL-distance` and `DLL-lasso`. These experiments concern DISTANCE-SAT instances related to a non-classical planning problem, the parity

learning problem and slightly larger random 3-CNF formulas than the ones considered in the first part.

All the experiments were achieved on the same desktop computer Pentium® 4 3Ghz under Linux OS.

6.1. Statistical results on random instances

All the results presented in the following are about random 3-CNF formulas Σ generated under the "fixed-length clause" model (Chvátal and Szemerédi, 1988): literals are picked up under uniform conditions and clauses with redundant variables are rejected. Without loss of generality, the variables of $Dom(PI)$ are the first ones w.r.t. the lexicographic order, and they are assigned to 0. Every DISTANCE-SAT instance can be turned into an instance for which this assumption is satisfied, through a simple renaming of its literals.

Three solvers are considered: `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding`, the full-CNF approach using the state-of-the-art `BerkMin561` SAT solver.

The computational difficulty of a DISTANCE-SAT instance w.r.t. `DLL-distance` and `DLL-lasso` is quantified as the size of the corresponding search tree, where both unary and binary nodes are taken into account; in other words, it is evaluated as the number of variable assignments that are required to solve the instance. This difficulty measure depends neither on the implementation of the algorithms nor on the computer used to perform the experiments.

In the same way, the difficulty of a DISTANCE-SAT instance w.r.t. a SAT algorithm using the full-CNF encoding is quantified as the number of assignments required by the SAT algorithm to solve the corresponding SAT instance.

Figure 2 gives the proportion of satisfiable 100 variable instances, as a function of both the number of clauses of Σ and the distance d, given a complete interpretation PI ($|PI| = 100$). 1000 instances per point have been considered. A sharp transition appears between the satisfiable and the unsatisfiable regions. When d is large, the transition appears at the well-known satisfiability threshold for 3-SAT, i.e., when the ratio number of variables / number of clauses is equal to 4.25 (Cheeseman *et al.*, 1991) (Crawford and Auton, 1996). When d decreases, less clauses are required to produce unsatisfiable instances.

Figures 3 and 4 give difficulty of 100 variables instances with a complete interpretation PI w.r.t. `DLL-distance` and `DLL-lasso`. Each point corresponds to the mean difficulty over 200 instances.

With the `DLL-lasso` solver, for each distance under consideration, the difficulty is maximum at the sat/unsat transition. The global dif-

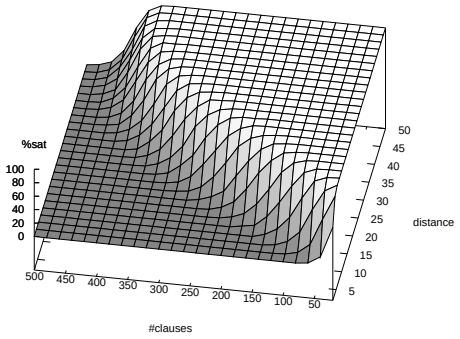

Figure 2. Proportion of satisfiable 100 variable 3-CNF instances of DISTANCE-SAT, as a function of both the number of clauses and the distance d, given a complete interpretation PI.

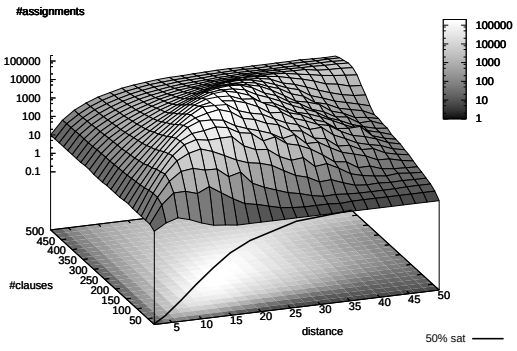

Figure 3. Difficulty of solving 100 variable 3-CNF instances of DISTANCE-SAT with `DLL-distance`, as a function of both the number of clauses and the distance d, given a complete interpretation PI.

ficulty peak of the surface is at the sat/unsat transition of distance 24 (280 clauses). An average of 120000 assignments is required to solve the corresponding random DISTANCE-SAT instances.

As for the `DLL-lasso` solver, for each distance under consideration, the average number of assignments used by the `DLL-distance` solver is maximum at the sat/unsat transition. However, when compared with the empirical behaviour of `DLL-lasso` on those instances, the behaviour of `DLL-distance` presents two salient differences. First, the difficulty peak for `DLL-distance` is larger than the one for `DLL-lasso`, especially for satisfiable instances. Second, the global difficulty peak for `DLL-distance` is at distance 16 (180 clauses) instead of 24 for `DLL-lasso`.

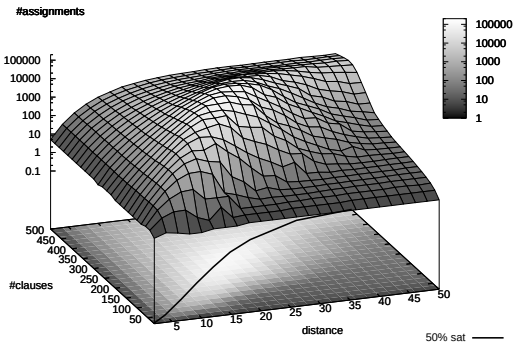

Figure 4. Difficulty of solving 100 variable 3-CNF instances of DISTANCE-SAT with `DLL-lasso`, as a function of both the number of clauses and the distance d, given a complete interpretation PI.

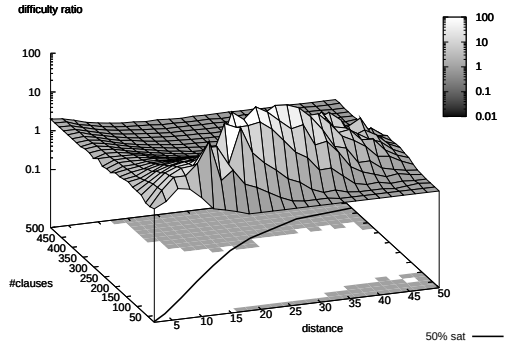

Figure 5. Ratio of the average difficulty of solving 100 variable 3-CNF instances of DISTANCE-SAT using `DLL-distance` to the average difficulty of solving the same instances using `DLL-lasso` as a function of both the number of clauses and the distance d, given a complete interpretation PI.

Figure 5 reports on a comparison of the two algorithms in the following way: the bottom surface shows (in white) the values of the distance and the number of clauses for which `DLL-lasso` outperforms `DLL-distance`; the top surface gives the ratio efficiency of `DLL-lasso` / efficiency of `DLL-distance`. Clearly, `DLL-lasso` performs much better than `DLL-distance` for the smallest distances, especially at the sat/unsat transition. At its global difficulty peak, `DLL-lasso` is slightly more efficient than `DLL-distance` (120000 assignments per run versus 140000 assignments per run). At the global difficulty peak of `DLL-distance`, `DLL-lasso` is about 5 times more efficient than `DLL-distance` (45000 assignments per run versus 247000 assignments

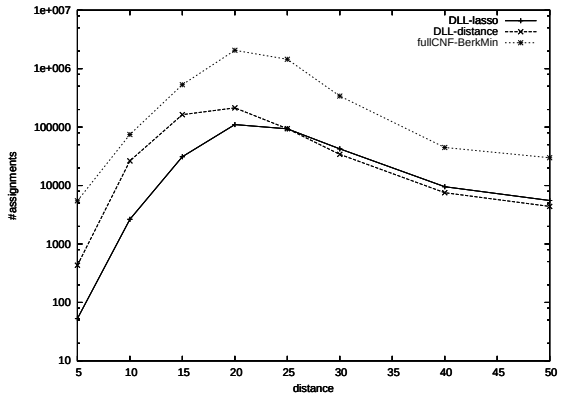

Figure 6. Comparison of the difficulty of solving 100 variable 3-CNF instances of DISTANCE-SAT with `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding` approaches, as a function of the distance, at the sat / unsat transition, given a complete interpretation PI.

per run). For some instances, like the (mostly satisfiable) ones with distance 16 and 80 clauses, `DLL-lasso` outperforms `DLL-distance` by two orders of magnitude (57 assignments per run versus 6600 assignments per run).

Since, in essence, the full-CNF approach to DISTANCE-SAT can be improved by using an "up-to-date" SAT solver (including lazy data structures and state-of-the-art backjumping techniques, which are not employed in our basic DLL algorithm), we found valuable to compare `DLL-distance`, `DLL-lasso` and the full-CNF approach coupled with a state-of-the-art SAT solver, namely `BerkMin561` (Goldberg and Novikov, 2002).

Figure 6 gives the mean number of variable assignments required by the three algorithms to solve random 100 variable DISTANCE-SAT instances, given a complete interpretation PI. For each distance, only the DISTANCE-SAT instances at the sat / unsat transition are considered, i.e., with a number of clauses such that 50% of the instances are satisfiable. Each point corresponds to 500 runs.

Clearly, `DLL-lasso` outperforms `DLL-distance` on the instances with distances lower than 25. On the other instances, `DLL-lasso` and `DLL-distance` perform similarly. Using the full-CNF approach, the `BerkMin561` solver requires much more variable assignments than the two other algorithms.

Yet, because they are based on different implementations, `BerkMin561` and the basic DLL solver used in `DLL-lasso` and `DLL-distance` *do not* achieve the same number of variable assignments per second. So we report on Figure 7 a comparison of the run times of

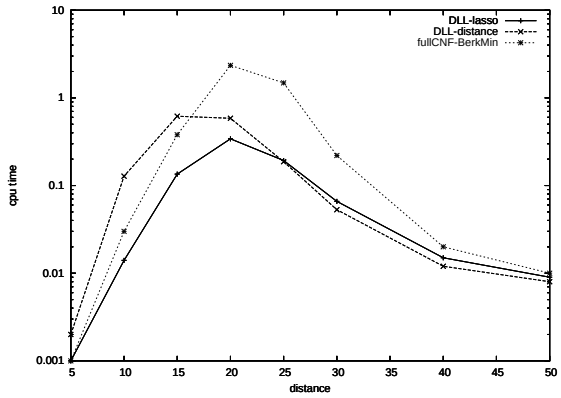

Figure 7. Cpu time required for solving 100 variable 3-CNF instances of DISTANCE-SAT with `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding` approaches, as a function of the distance, at the sat / unsat transition, given a complete interpretation PI.

`DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding` on the same instances as the ones considered on Figure 6.

Interestingly, one can observe that, even if it achieves much more assignments, `BerkMin-full-CNF-encoding` is faster than `DLL-distance` for the lowest distances (i.e., below 15). But `DLL-lasso` outperforms `BerkMin-full-CNF-encoding` on the whole range of distances.

Similarly to what has been reported on Figures 6 and 7 where complete interpretations have been considered, Figures 8, 9, 10, and 11 report on a comparison of the three algorithms on DISTANCE-SAT instances where PI is a partial interpretation.

Figures 8, 9 give the number of assignments and the cpu time required for solving 150 variable 3-CNF instances where 50% of the variables are fixed in PI.

For the smallest distances (i.e., $d \leq 10$), the best algorithm is `BerkMin-full-CNF-encoding`, while `DLL-distance` performs better (at least in terms of cpu time) on the remaining instances.

Figures 10, 11 give the number of assignments and the cpu time required for solving 200 variable 3-CNF instances where 25% of the variables are fixed in PI.

As for instances where 50% of variables are fixed in PI, `BerkMin-full-CNF-encoding` performs better than the two other algorithms on the smallest distances. For the other distances, `DLL-distance` is the best performer.

From the whole experiments, it is clear that none of the three algorithms outperforms the two others for any kind of DISTANCE-SAT instances. As to randomly generated instances, `DLL-lasso` is the best

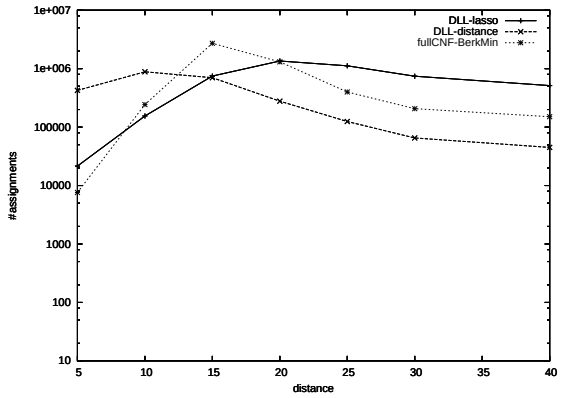

Figure 8. Comparison of the difficulty of solving 150 variable 3-CNF instances of DISTANCE-SAT, 75 variables fixed in PI, using `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding`, as a function of the distance, at the sat / unsat transition.

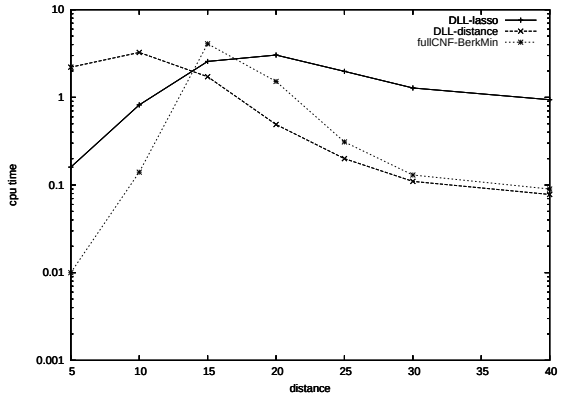

Figure 9. Cpu time required for solving 150 variable 3-CNF instances of DISTANCE-SAT, 75 variables fixed in PI, using `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding`, as a function of the distance, at the sat / unsat transition.

solver, in a large range of distances, whenever PI is complete. When only 25% or even 50% of variables are fixed in PI, the best algorithm is `BerkMin-full-CNF-encoding` for the smallest distances, and `DLL-distance` for the greatest ones.

6.2. Some additional results

In this section, we report on a second series of experiments, concerning three families of DISTANCE-SAT instances:

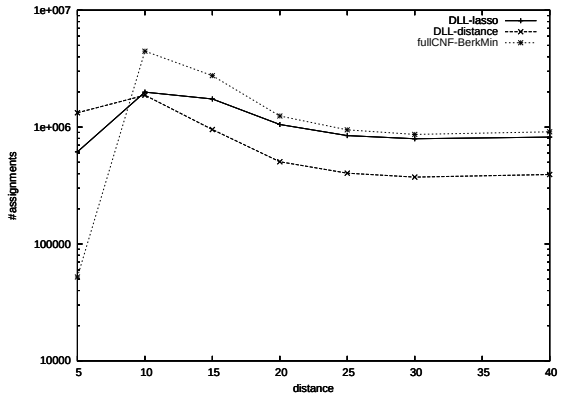

Figure 10. Comparison of the difficulty of solving 200 variable 3-CNF instances of DISTANCE-SAT, 50 variables fixed in PI, using `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding`, as a function of the distance, at the sat / unsat transition.

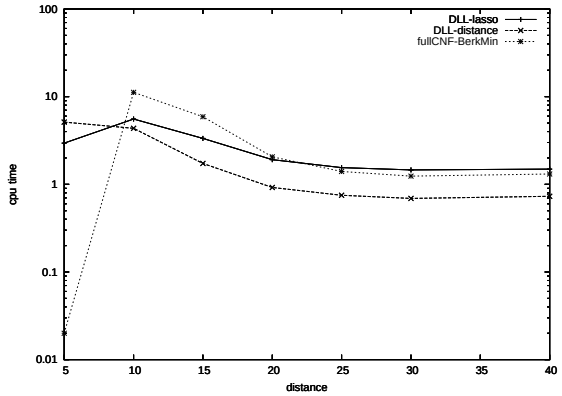

Figure 11. Cpu time required for solving 200 variable 3-CNF instances of DISTANCE-SAT, 50 variables fixed in PI, using `DLL-distance`, `DLL-lasso`, and `BerkMin-full-CNF-encoding` approaches, as a function of the distance, at the sat / unsat transition.

- DISTANCE-SAT instances modeling non-classical planning instances,
- DISTANCE-SAT instances modeling parity learning instances,
- DISTANCE-SAT instances based on randomly generated 3-CNF formulas.

All those instances were submitted to six DISTANCE-SAT solvers:

- the three solvers considered in Section 6.1, namely `DLL-distance`, `DLL-lasso` and `BerkMin-full-CNF-encoding`,

– three other solvers coupled with the full-CNF approach, namely `Minisat` version 1.14, `Jerusat` version 1.3, and a naive implementation of the DLL procedure, named `DLL-basic`, which is the core of the `DLL-distance` and `DLL-lasso` solvers.

Since the solvers are very dissimilar, only the cpu time is considered as a matter of comparison.

While performed in a less systematic way than the first series of experiments (reported in the previous section), the purpose of the second series was threefold: showing the feasibility of solving "realistic" instances of DISTANCE-SAT, showing the impact of the SAT solver used downstream to the encoding in the full-CNF approach, and finally, showing the efficiency of the two specialized algorithms `DLL-distance` and `DLL-lasso` in solving some instances, despite the fact they are based on a quite inefficient implementation of the DLL procedure.

6.2.1. *Non-classical planning*

We have first considered a few DISTANCE-SAT instances coming from a generalization of classical (propositional) planning, without uncertainty but with a more gradual preferential structure on goals. Formally, we have focused on instances of the classical planning problem, given by a description of a finite set A of (deterministic) actions (in a STRIPS-like language), the description of a (complete) initial state I, the description of the goal (as a (usually incomplete) state G), plus the planning horizon h (a nonnegative integer). An instance is positive whenever there exists a classical plan (a sequence of actions from A) whose length is bounded by h, allowing to go from I to a goal state (i.e. a state where G is satisfied). In the classical planning setting, a state (a complete interpretation over the fluent variables) is either a goal state (i.e. it contains all the literals of the goal) or it is not a goal state. Now, the set of goal states can be viewed as a fuzzy set, where the Hamming distance of a state to the goal (once normalized, and up to a scale inversion) expresses how much the given state is a goal state. Accordingly, instances of a (non-classical) planning problem can be easily obtained by considering an additional nonnegative integer d representing the distance to the goal, and a positive instance is one for which there exists a plan whose length is bounded by h, allowing to go from I to a state which is at distance at most d from G. Classical planning corresponds to the case when $d = 0$. Interestingly, relaxing this constraint by considering $d > 0$ can prove sufficient to get plans for values of h for which no classical plan exist. Thus, a trade-off between the quality of a plan (measured as the number of literals from G occurring in the final state) and the efficiency of the plan (the number of steps) can be looked for.

It is well-known that the classical planning problem can be reduced to SAT[3], and solved as such. Several reductions can be found in the literature. We slightly modified a reduction proposed in (Kautz *et al.*, 1996), based on explanatory frame axioms. The difference is that parallel actions are allowed when their preconditions and effects do not interfere. We took advantage of a translator from instances of classical planning (in PDDL format) to SAT which implements such a reduction[4] and is such that, for each input instance with a goal G, the literals of G are the first (unit) clauses of the corresponding SAT instance. Thanks to this property, it was easy to turn this translator into a translator from non-classical planning into DISTANCE-SAT. We generated some DISTANCE-SAT encodings of instances of non-classical planning (corresponding to well-known instances of classical planning) letting the distance d to vary as well as the horizon h[5]. The blocks world instance (blockaips15), the logistics one (logisticsaips10) and the miconic one (miconic30) come from IPC2 (International Planning Competition), while the depot instance (depot03) comes from IPC3 (Bacchus, 2001; Long and Fox, 2003).

In Table III, the first column gives the name of the starting instance of classical planning, the second column gives the number n of literals in the goal, the third value gives the horizon h under consideration, the fourth column gives the number $\#v$ of variables in the corresponding SAT encoding, while the last column gives the number $\#c$ of clauses in the corresponding SAT encoding.

Note that the size of each DISTANCE-SAT instance is very close to the size of the associated SAT instance (the size of the binary representation of d can be typically neglected). Despite the (quite huge) size of the instances, they have been solved easily using the full-CNF approach, showing thus the feasibility of solving "realistic" non-classical planning instances through DISTANCE-SAT.

The results are reported in Tables IV, VII, VI and V. The cpu time required by each solver is given in seconds, ns meaning that the instance was not solved within 3600 seconds.

Those experiments have shown the feasibility of solving instances of DISTANCE-SAT coming from a non-classical planning problem. The best approach clearly is the full-CNF one. This corroborates the conclusion drawn after the first series of experiments (when only few variables are assigned in the reference (partial) interpretation, the specialized solvers are not efficient). Furthermore, those experiments have clearly

[3] The membership of the classical planning problem (with bounded horizon) to NP ensures it.

[4] We would like to thank Vincent Vidal for providing us with this translator.

[5] Those instances are available from the authors on demand.

Table III. Non-classical planning instances used in the experiments.

name	n	h	$\#v$	$\#c$
blocksaips15	7	15	3216	46378
blocksaips15	7	14	3007	43292
blocksaips15	7	13	2798	40206
logisticsaips10	7	11	3102	13108
logisticsaips10	7	10	2829	11926
logisticsaips10	7	9	2556	10744
logisticsaips10	7	8	2283	9562
miconic30	6	17	2880	21552
miconic30	6	16	2712	20286
miconic30	6	15	2544	19020
depot03	6	11	5064	128226
depot03	6	10	4614	116580
depot03	6	9	4164	104934
depot03	6	8	3714	93288

Table IV. Results on Miconic30 planning instances. h is the horizon, d is the distance to the goal. The cpu time required by each solver is given in seconds. ns means that the corresponding instance was not solved within 3600 seconds.

Miconic30	$h = 17, d = 1$	$h = 16, d = 1$
#variables	2880	2712
#clauses	21552	20286
#fixed variables	6	6
satisfiable	yes	yes
specialized solvers		
`DLL-lasso`	ns	ns
`DLL-distance`	ns	ns
full CNF solvers		
`DLL-basic`	ns	ns
`Berkmin`	ns	ns
`Minisat`	335	2981
`Jerusat`	257	268

Table V. Results on Logisticaips10 planning instances. h is the horizon, d is the distance to the goal. The cpu time required by each solver is given in seconds. ns means that the corresponding instance was not solved within 3600 seconds.

Logisticaips10							
h	11	10	10	9	9	8	8
d	1	1	2	2	3	2	3
#variables	3102	2829	2829	2556	2556	2283	2283
#clauses	13108	11926	11926	10744	10744	9562	9562
#fixed variables	7	7	7	7	7	7	7
satisfiable	yes	no	yes	no	yes	no	yes
specialized solvers							
`DLL-lasso`	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*
`DLL-distance`	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*
full CNF solvers							
`DLL-basic`	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*
`Berkmin`	0.07	0.05	0.06	0.04	0.04	0.03	0.04
`Minisat`	0.01	0.02	0.01	0.02	0.01	0.02	0.01
`Jerusat`	0.09	0.03	0.05	0.06	0.02	0.04	0.05

shown the impact of the SAT solver used in the full-CNF approach: the encoding itself is not sufficient to ensure an efficient resolution (`DLL-basic` appears as a bad performer in the experiments, compared to the state-of-the-art SAT solvers).

6.2.2. *Parity learning*

We have also focused on the parity learning problem, which consists in guessing a parity function, given noisy input/output samples. The famous par32 SAT instance that was one of the ten IJCAI'97 challenges proposed in (Selman *et al.*, 1997) is an instance of the parity learning problem. In our experiments, we have considered parity learning instances obtained through the encoding proposed in (Bailleux and Boufkhad, 2003). Each of them includes a distance constraint which specifies the maximum number of samples with a wrong output value. To be more precise, the instances we have used have been generated as follows. A 32 bit parity function is first randomly generated. This parity function is characterized by a 32 bit string, called a *mask*. A bit of the input string is said to be *unmasked* if and only if the corresponding bit is set to 1 in the mask. The output value of the parity function for the

Table VI. Results on Depot03 planning instances. h is the horizon, d is the distance to the goal. The cpu time required by each solver is given in seconds. ns means that the corresponding instance was not solved within 3600 seconds.

Depot03							
h	11	11	10	10	9	9	8
d	1	2	1	2	2	3	3
#variables	5064	5064	4614	4614	4164	4164	3714
#clauses	128226	128226	165580	165580	104934	104934	93288
#fixed variables	6	6	6	6	6	6	6
satisfiable	no	yes	no	yes	no	yes	yes
specialized solvers							
`DLL-lasso`	ns	ns	ns	ns	ns	ns	ns
`DLL-distance`	ns	ns	ns	ns	ns	ns	ns
full CNF solvers							
`DLL-basic`	ns	ns	ns	ns	ns	ns	ns
`Berkmin`	3.17	0.45	0.50	0.58	0.28	0.40	0.20
`Minisat`	1.25	0.55	0.39	0.44	0.26	0.15	0.16
`Jerusat`	27.8	2.15	4.66	3.07	1.88	1.19	0.47

Table VII. Results on Blocsaips10 planning instances. h is the horizon, d is the distance to the goal. The cpu time required by each solver is given in seconds. ns means that the corresponding instance was not solved within 3600 seconds.

Blocsaips10	$h = 15, d = 1$	$h = 14, d = 1$	$h = 13, d = 1$
#variables	3216	3007	2798
#clauses	46378	43292	40206
#fixed variables	7	7	7
satisfiable	yes	yes	yes
specialized solvers			
`DLL-lasso`	ns	ns	ns
`DLL-distance`	ns	ns	ns
full CNF solvers			
`DLL-basic`	1714	ns	1903
`Berkmin`	0.58	0.60	0.40
`Minisat`	0.19	0.14	0.11
`Jerusat`	3.33	1.06	0.64

Table VIII. Results on parity learning instances based on two randomly generated 32 bit parity functions. d is the distance, which represent the maximum number of wrong samples. The cpu time required by each solver is given in seconds. ns means that the corresponding instance was not solved within 3600 seconds.

Parity function	$F_1, d = 6$	$F_1, d = 7$	$F_2, d = 6$	$F_2, d = 7$
#variables	518	518	552	552
#clauses	1944	1944	2080	2080
#fixed variables	64	64	64	64
satisfiable	no	yes	no	yes
specialized solvers				
`DLL-lasso`	50.5	88.6	55.7	97.4
`DLL-distance`	50.5	88.6	58.5	103.5
full CNF solvers				
`DLL-basic`	673	1295	102	195
`Berkmin`	5.89	32.1	12.0	41.9
`Minisat`	7.48	0.01	13.0	30.8
`Jerusat`	14.5	0.10	34.2	41.0

instance is 1 if and only if there is an odd number of unmasked bits in the input string. 64 input samples have been randomly generated, and the corresponding outputs processed. Then, 7 of the output values have been flipped. The data are encoded as a CNF formula according to the method proposed in (Bailleux and Boufkhad, 2003). By construction, the resulting DISTANCE-SAT instance is satisfiable for any distance value greater than 6.

Table VIII gives the cpu times required for solving two satisfiable instances ($d = 7$) and the two corresponding unsatisfiable instances ($d = 6$).

Again, the best performers for those experiments are the state-of-the art SAT solvers used in the full-CNF approach. Specialized solvers behave not so bad (the cpu time they required is "only" an order of magnitude larger then the cpu time required by the state-of-the art SAT solvers, despite the fact they are based on an inefficient DLL procedure), and better than the full-CNF approach based on `DLL-basic`.

Table IX. Results on random instances with 100% variables fixed in PI. The cpu time required by each solver is given in seconds. ns means that the corresponding instance was not solved within 3600 seconds.

Instance number	1	2	3	4	5	6
#variables	200	200	200	200	150	150
#clauses	200	200	400	400	450	450
#fixed variables	200	200	200	200	150	150
distance	15	15	30	30	35	35
satisfiable	no	yes	no	yes	no	yes
specialized solvers						
`DLL-lasso`	2.13	0.01	2377	48.1	7.63	10.6
`DLL-distance`	72.9	163.6	*ns*	*ns*	6.77	9.01
full CNF solvers						
`DLL-basic`	*ns*	*ns*	*ns*	*ns*	*ns*	*ns*
`Berkmin`	1.62	0.01	*ns*	0.32	802	1.48
`Minisat`	1.14	0.02	1810	0.22	101	0.07
`Jerusat`	3.88	0.03	*ns*	0.71	313	36.1

6.2.3. *Random instances*

Finally, we report on some results obtained with random DISTANCE-SAT instances, significantly larger than the ones considered in Section 6.1. Tables IX, XI, X present the results that were obtained. Globally, the best performer is `Minisat`, but some instances with complete reference interpretations are best solved by `dll-lasso` or `dll-distance`.

Those experiments corroborate the conclusions drawn after the first series of experiments: when the number of variables assigned in the reference (partial) interpretation is sufficiently large, the specialized solvers `DLL-distance` and `DLL-lasso` can prove as much better performers than the state-of-the-art SAT solvers in the full-CNF approach.

7. Conclusion

The main contribution of this paper is the identification of the complexity of DISTANCE-SAT and of several restrictions of it, as well as two algorithms for solving it in the CNF case.

Those two algorithms were empirically evaluated and compared with a full-CNF solving approach which takes advantage of a recent

Table X. Results on random instances with 50% variables fixed in PI. The cpu time required by each solver is given in seconds. *ns* means that the corresponding instance was not solved within 3600 seconds.

Instance number	7	8	9	10	11	12
#variables	400	400	400	400	200	200
#clauses	400	400	800	800	600	600
#fixed variables	200	200	200	200	100	100
distance	5	5	15	15	20	20
satisfiable	no	yes	yes	no	yes	no
specialized solvers						
`DLL-lasso`	0.01	*ns*	*ns*	*ns*	92.0	177
`DLL-distance`	*ns*	*ns*	928	*ns*	108	99.8
full CNF solvers						
`DLL-basic`	68.3	54.1	*ns*	*ns*	*ns*	*ns*
`Berkmin`	0.01	0.01	0.03	81.2	251	*ns*
`Minisat`	0.01	0.01	0.01	13.3	32.8	71.8
`Jerusat`	0.02	0.01	0.01	44.4	662	*ns*

efficient CNF encoding scheme, allowing some conclusions about their respective applicability for being drawn. Especially, in the light of the experiments, `DLL-lasso` appears as the most efficient algorithm for hard instances with complete PI, at the sat/unsat transition. This stresses the quality of $branching_{lasso}$ as a branching heuristic, since it is the only difference between `DLL-lasso` and a naive DLL procedure.

On the other hand, the `BerkMin-full-CNF-encoding` and `DLL-distance` algorithms appear more efficient for solving DISTANCE-SAT instances with incomplete PI.

This work calls for many perspectives. A first one consists in evaluating other branching rules, based on corresponding branching rules used in SAT solvers. Another one consists in incorporating some of the features used by state-of-the-art SAT solvers (like watched literals, clause learning and so on) into `DLL-lasso` so as to improve it; this is not an obvious extension since exploiting lazy data structures prevents from computing in an efficient way sophisticated branching rules, like $branching_{lasso}$.

Because instances of DISTANCE-SAT can be easily encoded as instances of the satisfiability problem for propositional cardinality formulas, it would be interesting to extend our algorithms so as to make

Table XI. Results on random instances with 25% variables fixed in PI. The cpu time required by each solver is given in seconds. *ns* means that the corresponding instance was not solved within 3600 seconds.

instance number	13	14	15	16
#variables	400	400	400	400
#clauses	800	800	1200	1200
#fixed variables	100	100	100	100
distance	4	4	10	10
satisfiable	yes	no	yes	no
specialized solvers				
`DLL-lasso`	*ns*	*ns*	1678	*ns*
`DLL-distance`	*ns*	*ns*	*ns*	*ns*
full CNF solvers				
`DLL-basic`	0.04	1.12	93.6	2618
`Berkmin`	0.01	0.01	0.01	4.0
`Minisat`	0.01	0.01	0.07	2.50
`Jerusat`	0.01	0.01	0.08	4.89

them able to take simultaneously several distance constraints into account. In addition, it would be interesting to study whether additional structural properties (e.g., how the clauses of Σ are satisfied by the reference interpretation) could be exploited in the design of more efficient DISTANCE-SAT solvers. These are two issues for further research.

(Hebrard *et al.*, 2005) study the complexity of several problems, related to the issue of determining how diverse (resp. close) some solutions of a CSP are. The decision problem corresponding to the optimization problem called MOSTCLOSE in (Hebrard *et al.*, 2005) is related to SAT-DISTANCE-SAT, a variant of DISTANCE-SAT: given a propositional formula Σ, a model I of it and and integer d, determine whether Σ has another model, which is at distance at most d from I. It is not difficult to prove that SAT-DISTANCE-SAT is NP-complete. A deeper investigation of the connections between MOSTCLOSE and DISTANCE-SAT is a topic for further research.

Some other perspectives include the comparison of the full-CNF approach with other encodings of cardinality constraints (especially the one reported in (Sinz, 2005)), and the optimization version of DISTANCE-SAT (which can be easily reduced to the optimization prob-

lem WEIGHTED-MAX-SAT). An issue to be investigated is the adaptation of the specialized solvers to such optimization problems, and the evaluation of their performances.

Appendix

Proof of Proposition 1:

– Membership results:

  - to NP: this result is very easy since once a model of Σ has been guessed in nondeterministic polynomial time, it is sufficient to check in deterministic polynomial time that it disagrees with PI on at most d variables.
  - to P: all the membership results come from the tractability of the following cases:
    * a Horn (resp. reverse Horn, Krom, Blake) Σ, a fixed d, any PI: tractability is the consequence of two basic facts. On the one hand, since every variable occurring in PI but not in Σ is irrelevant (i.e., it can be discarded from PI without questioning the membership to DISTANCE-SAT) and since such an elimination process can be done in time linear in $|\Sigma|$, the number m of variables of PI that has to be considered is bounded by the number of variables occurring in Σ, hence by the size of Σ. So only $\sum_{i=1}^{d} \binom{m}{i} = \mathcal{O}(d \times m^d)$ partial interpretations PI' must be checked for being consistent with Σ in the worst case. The quantity $\sum_{i=1}^{d} \binom{m}{i}$ is polynomial in the size of Σ when d is fixed. On the other hand, whenever Σ is a Horn (resp. reverse Horn, Krom, Blake) formula and PI' is a partial interpretation (viewed as a term), it is possible to compute in time polynomial in $|\Sigma|+|PI'|$ a formula from the same fragment that is equivalent to $\Sigma \wedge PI$. This is obvious for the Horn, reverse Horn and Krom class and the case of the Blake class is given by Proposition 36 from (Marquis, 2000). The fact that the satisfiability test can be done in polynomial time for all these fragments concludes the proof.
    * a DNNF Σ, any d, any PI: this is a direct, slight extension of Theorem 10 from (Darwiche, 2001). Let us show how to compute in polynomial time the minimal Hamming distance between (a model of) Σ and (an extension of)

PI. The proof is by induction on the structure of Σ. The base case is as follows: if Σ is $true$ (resp. $false$) then its distance to PI is 0 (resp. $+\infty$); otherwise, if Σ is a literal l, then its distance to PI is 0, except when $\neg l \in PI$; in the latter case, the distance is 1. There are two inductive steps, one for conjunctions and the other one for disjunctions. As to the conjunction case, since the conjuncts do not share any variables, the minimal distance between Σ and PI is given by the sum of the minimal distances between the conjuncts of Σ and PI. As to the disjunction case, since every model of Σ is a model of at least one of its disjuncts (and *vice-versa*), the minimal distance between Σ and PI is given by the smallest distance between the disjuncts of Σ and PI.

  - ∗ any Σ, a fixed d, a complete PI: direct consequence of the fact that only $\sum_{i=1}^{d} \binom{|PI|}{i} = \mathcal{O}(d \times |PI|^d)$ interpretations must be checked for being a model of Σ in the worst case.

– NP-hardness results: all the results come from the NP-hardness of the following restrictions:

  - a CNF Σ, a fixed d, any PI: reduction from SAT. To every CNF formula Σ we associate the instance $\langle \Sigma, \{\}, 0 \rangle$ of DISTANCE-SAT. Obviously enough, we have $\Sigma \in$ SAT if and only if $\langle \Sigma, \{\}, 0 \rangle \in$ DISTANCE-SAT.
  - a Horn (resp. reverse Horn, Krom, Blake) Σ, any d, a complete PI: we consider the restricted fragment monotone-Krom, which is the subset of both the reverse Horn fragment and of the Krom fragment, consisting of the CNF formulas for which each clause contains at most 2 literals and every literal is positive. We exhibit a reduction from the well-known NP-complete problem HITTING SET to DISTANCE-SAT with Σ monotone-Krom. Indeed, an instance of HITTING SET is given by a pair $\langle C, d \rangle$ where C is a finite set of subsets of a finite set S and d is a non-negative integer; the instance is positive if and only if there exists a subset H of S s.t. the cardinal of H is lower than or equal to d and for every $c \in C$, $H \cap c \neq \emptyset$. It is known that the problem is NP-complete even if the case where each $c \in C$ contains at most two elements (Karp, 1972). Our polynomial reduction is as follows: we map $\langle C, d \rangle$ to $\langle \Sigma, I, d \rangle$ such that, each element of $\bigcup_{c \in C} c$ being viewed as a propositional symbol from PS, each subset $c \in C$ can be viewed as a clause containing at most

two positive literals, and the conjunction Σ of these clauses is a monotone-Krom formula. I is the interpretation s.t. each variable from PS is set to 0. Clearly enough, each hitting set H of C can be viewed as an implicant of Σ since Σ is monotone. Every model of Σ is an extension of at least one of its implicant and the converse holds. It remains to observe that H contains d elements if and only if it disagrees with I on d variables.

The proof is similar in the Horn case (just flip every literal in the above reduction and consider I as the interpretation that set every variable to 1). As to the Blake case, the result comes from the fact that to every monotone-Krom formula it is possible to associate an equivalent Blake one in polynomial time (just remove every clause that is entailed by another one).

■

Proof of Proposition 2: Let $\langle \Sigma, PI, d \rangle$, where PI is a complete interpretation, be the input of `DLL-lasso`. Without loss of generality, we assume that PI sets all the variables of Σ to 0. Let K be the number of literals in the largest clause of Σ.

If all the clauses of Σ are satisfied by PI, then a model is found and the procedure stops.

Now, suppose that some clauses of Σ are falsified by PI. Let q denote the number of literals in the smallest clause of Σ that is falsified by PI. Let x denote the branching variable at the root of the search tree. Since $branching_{lasso}$ is used, x belongs to a clause of size q.

If $q = 1$ then unit propagation operates, fixing x to 1 and producing a simplified instance of DISTANCE-SAT with distance $d - 1$; else, in the worst case, two instances of DISTANCE-SAT are successively produced by fixing x to 0 and 1, respectively. Fixing x to 0 produces an instance with distance d and $q - 1$ literals in the shortest clause falsified by PI. Fixing x to 1 produces an instance with distance $d-1$. Given that any clause in Σ has at most K literals, the number of assignments required by `DLL-lasso` to solve $\langle \Sigma, PI, d \rangle$ is bounded by $N_{d,K}$ s.t.:

$$\begin{cases} \forall \kappa \geq 0, & N_{0,\kappa} = 0, \\ \forall \delta > 0, & N_{\delta,1} = 1 + N_{\delta-1,K}, \\ \forall \delta > 0, \forall \kappa > 1, & N_{\delta,\kappa} = 1 + N_{\delta,\kappa-1} + N_{\delta-1,K}. \end{cases}$$

Because $\forall \kappa > 0, \forall \delta > 0$, $N_{\delta,\kappa} \leq \kappa(1 + N_{\delta-1,K})$, we obtain

$$N_{d,K} = \mathcal{O}(K^d).$$

Since the run time needed to propagate a variable assignment onto Σ (including, if applicable, the computation of the branching function) is $\mathcal{O}(|\Sigma|)$, the time complexity of `DLL-lasso` is $\mathcal{O}(|\Sigma| \times K^d)$. ∎

Acknowledgements

Many thanks to Vincent Vidal for his help. The second author has been partly supported by the IUT de Lens, the Université d'Artois, the Nord/Pas-de-Calais Région through the IRCICA Consortium, and by the European Community FEDER Program.

References

F. Bacchus. The 2000 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 22(3):47–56, 2001.

O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Proc. of the 9^{th} International Conference on Principles and Practice of Constraint Programming (CP'03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122, Kinsale (Ireland), 2003. Springer Verlag.

B. Benhamou, L. Saïs, and P. Siegel. Two proof procedures for a cardinality based langage in propositional calculus. In *Proc. of the 11^{th} Annual Symposium on Theoretical Aspects of Computer Science (STACS'94)*, volume 775 of *Lecture Notes in Computer Science*, pages 71–84, Caen (France), 1994. Springer Verlag.

C. Bessiere and J.-C. Regin. Refining the basic constraint propagation algorithm. In *Proceedings of IJCAI01*, pages 309–315, 2001.

I. Bloch and J. Lang. Towards mathematical morphologics. In *Proc. of the 8^{th} International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'00)*, pages 1405–1412, Madrid (Spain), 2000.

A. Bockmayr. Solving pseudo-boolean constraints. In *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, pages 22–38. Springer Verlag, 1995.

J. Chabrier, V. Juliard, and J.J. Chabrier. SCORE(FD/B): An efficient complete local-based search method for satisfiability problems. In *Proc. of the CP'95 Workshop on Solving Really Hard Problems*, pages 25–30, Cassis (France), 1995.

P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proc. of the 12^{th} International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 331–337, Sydney, 1991.

V. Chvátal and E. Szemerédi. Many hard examples for resolution. *JACM*, 35(4):759–768, 1988.

J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence*, 81:31–57, 1996.

M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In *Proc. of the 7^{th} National Conference on Artificial Intelligence (AAAI'88)*, pages 475–479, St Paul (MN), 1988.

A. Darwiche and P. Marquis. A perspective on knowledge compilation. In *Proc. of the* 17^{th} *International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 175–182, Seattle (WA), 2001.

A. Darwiche. Compiling knowledge into decomposable negation normal form. In *Proc. of the* 16^{th} *International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 284–289, Stockholm (Sweden), 1999.

A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

O. Dubois, P. André, Y. Boufkhad, and J. Carlier. *SAT versus UNSAT*, pages 415–436. Second DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.

D. Dubois, F. Esteva, P. Garcia, L. Godo, and H. Prade. A logical approach to interpolation based on similarity relations. *International Journal of Approximate Reasoning*, 17(1):1–36, 1997.

N. Eén and N. Sörensson. An extensibel sat solver. In *Proc. of the* 6^{th} *International Conference on Theory and Application of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer Verlag, 2004.

K. Forbus. Introducing actions into qualitative simulation. In *Proc. of the* 11^{th} *International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1273–1278, Detroit (MI), 1989.

M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, 1979.

M.L. Ginsberg, A.J. Parkes, and A. Roy. Supermodels and robustness. In *Proc. of the* 15^{th} *National Conference on Artificial Intelligence (AAAI'98)*, pages 334–339, Madison (WI), 1998.

E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver. In *Proc. of the Design, Automation and Test in Europe Conference (DATE'02)*, pages 142–149, Paris, 2002.

E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh. Finding diverse and similar solutions in constraint programming. In *Proc. of the* 20^{th} *National Conference on Artificial Intelligence (AAAI'05)*, pages 372–377, 2005.

R.M. Karp. *Reducibility among combinatorial problems*, chapter Complexity of Computer Computations, pages 85–103. Plenum Press, New York, 1972.

H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proc. of the* 5^{th} *International Conference on Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

S. Konieczny and R. Pino Pérez. On the logic of merging. In *Proc. of the* 6^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 488–498, Trento (Italy), 1998.

C. Lafage and J. Lang. Logical representation of preferences for group decision making. In *Proc. of the* 7^{th} *International Conference on Knowledge Representation and Reasoning (KR'00)*, pages 457–468, Breckenridge (C0), 2000.

C. Lafage and J. Lang. Propositional distances and preference representation. In *Proc. of the* 6^{th} *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01)*, pages 48–59, Toulouse (France), 2001.

ChuMin Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of the* 15^{th} *International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya (Japan), 1997.

P. Liberatore and M. Schaerf. Reducing belief revision to circumscription (and vice versa). *Artificial Intelligence*, 93(1-2):261–296, 1997.

D. Long and M. Fox. The 3^{rd} International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

P. Marquis. *Consequence finding algorithms*, volume 5 of *The Handbook for Uncertain and Defeasible Reasoning*, chapter Algorithms for Uncertain and Defeasible Reasoning, pages 41–145. Kluwer Academic Publishers, 2000.

A. Nadel. The jerusat sat solver. Master's thesis, Hebrew University of Jerusalem, 2002.

R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

A. Roy and Ch. Wilson. Supermodels and closed sets. *Electronic Colloquium on Computational Complexity (ECCC)*, TR00-010, 2000.

B. Selman, H.A. Kautz, and D.A. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 50–54, 1997.

C. Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *Proc. of the 11^{th} International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831, Sitges (Spain), 2005. Springer Verlag.

J. Slaney and T. Walsh. Phase transition behavior: from decision to optimization. In *Proc. of the 5^{th} International Symposium on the Theory and Applications of Satisfiability Testing (SAT'02)*, Cincinnati (OH), 2002.

S. Thiébaux, J. Slaney, and P. Kilby. Estimating the hardness of optimisation. In *Proc. of the 14^{th} European Conference on Artificial Intelligence (ECAI'00)*, pages 123–127, Berlin (Germany), 2000.

P. Van Henteryck and Y. Deville. The cardinality operator: A new logical connective for constraint logic programming. In *Proc. of the 8^{th} International Conference on Logic Programming (ICLP'91)*, pages 745–749, Paris (France), 1991.

J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.