Multi–Domain Logic and its Applications to SAT

Tudor Jebelean
RISC – Linz, Austria
Tudor.Jebelean@risc.uni-linz.ac.at

Gábor Kusper
Eszterházy Károly College
gkusper@aries.ektf.hu

Abstract

We describe a new formalism and special proof methods for a novel generalization of propositional logic, which is especially suitable for solving the satisfiability problem (SAT). A Multi–Domain Logic (MDL) formula is quantifier–free and contains only atoms of the form $x \in A$, where x is a variable and A is a constant set. For formulae in conjunctive normal form, we are interested in finding solutions (assignments to variables which satisfy all clauses). Classical propositional logic corresponds to the special case when each set is either $\{\mathbb{T}\}$ or $\{\mathbb{F}\}$.

The union of all the sets occurring for a certain variable can be seen as "the domain" of that variable, thus MDL is also a generalization of multi-valued logic, but with different domains for variables. The most distinctive feature is, however, the indication of the sub-domain in each clause.

The notions of resolution, subsumption, as well as the basic steps of the DPLL method generalize in an elegant and straightforward way. As a novel MDL specific technique, variable clustering *consists in creating a new variable ranging over the cartesian product of the domains of several "clustered" variables. This allows the transformation of classical SAT problems in MDL problems with less literals, and in which the propagation of information can be performed more efficiently than classical unit propagation.*

The basic idea of MDL originates from the earlier work of the second author on "hyper-unit" propagation (that is simultaneous propagation of several unit clauses) and on the representation and propagation of "k-literals" (generalized literals containing information on several propositional variables).

Preliminary experiments with a prototype Java implementation exhibit speed–ups of up to 30 times.

1. Introduction

The basic algorithmic step of most efficient SAT solvers is *unit propagation*, which can be seen as simplification of the SAT problem by propagating the information provided by one unit clause. The research presented in this paper is motivated by the idea of propagating simultaneously the information provided by *several clauses*.

A first approach, introduced by the second author [5, 4], consists in simultaneous propagation of *several unit clauses*. Later the same author extended this approach to propagation of the information provided by *several clauses* [6], by using a special representation of this information into a so called *"k-literal"*. In more detail, if $\mathcal{C}$ is the set of clauses whose information we want to propagate, and $\mathcal{V}$ is the set of all propositional variables occurring in them, then the conjunction of these clauses defines a boolean function F on $\mathcal{V}$. This function contains all the information provided by $\mathcal{C}$, in the same way as a classical literal contains all the information provided by one unit clause. If the size of $\mathcal{V}$ is k, then this function can be represented (as suggested by the first author) as a string of 2^k bits, which is called a *k-literal* (generalizing the classical 1–literals). Each bit in the string of 2^k bits corresponds (in a certain conventional order) to a certain assignment of $\mathbb{T}/\mathbb{F}$ to the variables in $\mathcal{V}$. Thus, by partitioning the original set of variables into subsets of a certain size k, one can transform the original problem into a set of clauses of k–literals, and all the classical operations on clauses and literals can be generalized and implemented in a natural way, because conjunction and disjunction of k–literals correspond to the same operations on bit–strings (or computer words).

Later the first author noticed that such a k–literal (or just *multi-literal* – because they do not have to be all of the same size) can be represented by the *set of assignments* which make F true, and in fact, from the algorithmic point of view, the nature of the elements of the set does not matter. Each multi-literal can be seen as an atom stating that a new variable x (associated to the subset $\mathcal{V}$) belongs to a certain set. Again all the operations on literals and clauses can be generalized in a natural way, because conjunction and disjunction of literals correspond now to intersection and union of sets, respectively.

Obviously any solution for a certain variable x has to be from the union of the sets associated to x (one set in each clause) – thus this union can be called *the domain* of

x. Since different variables can have different domains, we call this *multi–domain logic* (of course this is just a subset of first order predicate logic – see more details below). Moreover, this name intents to suggest the fact that *a variable is associated to different sets of values in different clauses*, which constitutes *the* essential difference between this logic and multi-valued logic.

The resolution method [8], Davis–Putnam (variable elimination) [3], and DPLL [2] generalize in an elegant and straightforward way. Resolution can be performed on an arbitrary number of clauses and it is realized by intersection and union of sets. Subsumption reduces to testing set inclusion.

A crucial novel technique in MDL is *variable clustering*, which consists in creating a new variable ranging over the cartesian product of the domains of several "clustered" variables. The transformation of a classical set of clauses (which is MDL with all domains being $\{\mathbb{T},\ \mathbb{F}\}$) is just merging of the original propositional variables, using one or more clustering factors. Merging of variables can be also further applied during the solving process, when the size of some domains decreases sufficiently.

The MDL approach exhibits a good potential for efficient implementations, because sets can be efficiently represented as bit strings and set operations can be performed by logical operations on computer words. In the present paper we report on a Java implementation which extends previous "k-literal" implementations of the second author. The experiments demonstrate the effectiveness of the method and reveal the potential for further research towards more efficient SAT solvers.

2. Multi–Domain Logic

A *formula* in *multi–domain logic (MDL)* is a conjunction of disjunctions (clauses) of literals of the form $x \in A$, where x is a variable and A is a set. We assume that a variable occurs at most once in a clause (since literals over the same variable merge by joining the sets).

An *interpretation* is a set of *assignments* of values to variables. A literal $x \in A$ is satisfied only by those interpretations which contain the assignment $x = a$ with $a \in A$. Truth values of composite formulae, as well as validity, satisfiability, etc. are defined like in classical propositional logic. For a given formula, we define the *domain* D_x of a variable x as the union of all sets occurring in literals with x. Note that it is sufficient to consider assignments with values from the respective domains.

An interpretation satisfying a formula will be called *solution* of the formula, and proving reduces to finding solutions or showing their absence. *The MDL algorithmic problem consists in finding solutions to formulae.*

For brevity, clauses will be denoted by $Ax \vee By \vee Cz \vee \ldots$, and interpretations by $\{x = a,\ y = b,\ z = c,\ \ldots\}$ (where $a \in D_x,\ b \in D_y,\ c \in D_z,\ \ldots$). Literals having the empty set can be missing from the clause, since they do not influence the truth value. Likewise, literals having the domain of the respective variable are allways true, thus the corresponding clause becomes irrelevant.

MDL is a generalization of classical propositional logic. The latter corresponds to the special case when each set is included in $\{\mathbb{T},\ \mathbb{F}\}$. Then $x \in A$ corresponds to $\mathbb{F},\ x,\ \neg x,\ \mathbb{T}$ when A is $\{\}$, $\{\mathbb{T}\}$ $\{\mathbb{F}\}$, $\{\mathbb{T},\ \mathbb{F}\}$, respectively.

Also, it is clear that MDL statements can be expressed in first order predicate logic, and their semantics is the usual one, by using an appropriate part of set theory.

It is straightforward to extend MDL to arbitrary propositional constructs, although some care is required for defining the domains corresponding to formulae containing negation. Moreover one can generalize the atoms into $x \leq a$ with a from a lattice. This paper, however, focuses on the normal form syntax and semantics as previously defined, since this appears to be more interesting for applying MDL to SAT solving. Moreover, since MDL formulae can be transformed into normal form and, furthermore, lattices can be expressed as subset sets, these generalizations do not essentially enhance the notion.

3. Classical Proof Methods

All proof methods from classical propositional logic generalize in a very natural way to MDL.

Truth Table and Equivalence Rewriting are straightforward, but less interesting for formulae in normal form, thus we will not discuss them here.

Resolution. If $\mathcal{C}_1, \mathcal{C}_2$ are disjunctions of literals, then the clause $(A_1 \cap A_2)x \vee \mathcal{C}_1 \vee \mathcal{C}_2$ is a logical consequence of $A_1x \vee \mathcal{C}_1$ and $A_2x \vee \mathcal{C}_2$. By induction, the clause $R = (\cap_{i=1}^{p} A_i)x \vee \vee_{i=1}^{p} \mathcal{C}_i$ is the *resolvent* of the clauses $\{A_ix \vee \mathcal{C}_i\}_{i=1}^{p}$. Note that $\vee_{i=1}^{p} \vee_{k=1}^{n} A_{ik}x_k$ reduces to $\vee_{k=1}^{n}(\cup_{i=1}^{p} A_{ik})x_k$.

If $\cap_{i=1}^{p} A_i$ is empty, then we say R is an *eliminating* resolvent. Moreover R is *minimally eliminating* if the intersection of any subset of $\{A_i\}_{i=1}^{p}$ is not empty.

Completeness. By repeated exhaustive application of minimally eliminating resolution to an unsatisfiable clause set one allways obtains the empty clause. This can be shown e. g. by generalizing the semantic tree used in [1].

Example: Let $D_x = D_y = D_z = \{1,\ 2,\ 3\}$. If we do resolution on these two clauses:

$$\{3\}x \vee \{1,\ 2\}y \vee \{1,\ 3\}z$$

$$\{1\}x \vee \{2,\ 3\}y \vee \{1,\ 2\}z$$

then their resolvent on variable y is $\{1,\ 3\}x \vee \{2\}y \vee \{1,\ 2,\ 3\}z$. Note, that here the resolvent is a tautology, because the value set of z is the same as its domain.

Davis–Putnam and DPLL. *Variable Elimination (Davis Putnam).* After performing all possible minimally eliminating resolutions on a certain variable, the clauses containing that variable can be eliminated without changing the satisfiability of the clause set.

Subsumption. We say that a clause $\vee_{k=1}^{n} A_k x_k$ (where some A_k might be empty) *subsumes* the clause $\vee_{k=1}^{n} A'_k x_k$ iff $A_k \subseteq A'_k$ for any $k \in \overline{1,n}$. Subsumed clauses can be eliminated without changing the set of solutions.

Unit resolution. For any variable x we may include in the clause set the unit clause $x \in D_x$. If a variable occurs in several unit clauses, then these *collapse* into one by intersecting the respective sets. (The intersection becomes the new domain.) In any atom $x \in A$, the set A can be replaced by $A \cap D_x$. (This may remove some occurrences of x.)

Pure literal. If the intersection of all nonempty sets occurring with a variable x is nonempty, then any element of the intersection will satisfy all the clauses where x occurs, thus the satisfiability of the set depends only on the remaining clauses.

Unit propagation (DPLL). Whenever a new unit clause L is created:

- eliminate clauses subsumed by L (if all clauses are eliminated then the problem is satisfiable);
- perform unit resolution with L (this may create new unit clauses, or the empty clause and then this branch is unsatisfiable).

When no unit clauses are present, split using a clause or a variable – see below.
Example: The clause set is:

$$\{1,\ 2\}x$$

$$\{2,\ 3\}x$$

$$\{1,\ 2,\ 3\}x \vee \{1\}y \vee \{3\}z$$

$$\{1,\ 3\}x \vee \{2\}z$$

The initial domains are: $D_x = \{1,\ 2,\ 3\}$, $D_z = \{1\}$, $D_z = \{2,\ 3\}$. The first two unit clauses on x collapse to $\{2\}x$, thus D_x becomes $\{2\}$. The second clause is removed by unit subsumption. In the last clause the literal on x is removed by unit resolution. The remaining clause set contains only $\{2\}z$. Therefore, the solution of the set is $x = 2,\ y = 1,\ z = 2$.

Split creates new unit clauses on several branches. One possibility is to use a [short] clause and create a branch for each literal. In the present paper we will investigate in more detail another possibility, which is analogous to the classical method, that is split using a variable. If D_x is the domain of the variable x, then a partition $\{D_k\}_{k=1}^{n}$ of D_x can be used for creating n branches, each containing a new unit clause $x \in D_k$. It is of course necessary that in each of the new branches, the variable x disappears, that is, each literal containing x is either subsumed or canceled by the new unit clause. One way to insure this is to partition D_x in subsets of one element each – which of course increases the number of branches. More efficient partitions can be determined as described in the next section.

4. Specific Methods

The interest of MDL lays in its novel specific concepts and methods, from which we present here two important ones. The first is *variable clustering*, which allows a compact representation of SAT problems and propagation of more information (instead of just units). Moreover, clustering can be used also during the MDL proof, for further compacting the representation. The second is *weak assignments*, which allows to reduce the number of branches in the split step.

Clustering of variables. Let x, x' be two variables and D, D' be their domains. One can transform the formula by replacing x, x' by a new *multi–variable* y, which (intuitively) represents $\langle x, x' \rangle$ and ranges over $D \times D'$. A disjunction $Ax \vee A'x'$ will become $((A \times D') \cup (D \times A'))y$. By induction, clustering extends to an arbitrary number of variables.

Example: Consider the set of propositional clauses:

$$\neg a \vee \neg b,\ \ \neg a \vee b,\ \ a \vee \neg b.$$

These can be represented in MDL as:

$$\{0\}a \vee \{0\}b,\ \ \{0\}a \vee \{1\}b,\ \ \{1\}a \vee \{0\}b.$$

(The domains are $D_a = D_b = \{0,\ 1\}$.) Now we can cluster a and b into a new multivariable y with domain $D_y = D_a \times D_b = \{00,\ 01,\ 10,\ 11\}$. Then the clause set becomes:

$$\{\mathbf{0}0,\ \mathbf{0}1\}y \cup \{0\mathbf{0},\ 1\mathbf{0}\}y = \{00,\ 01,\ 10\}y,$$
$$\{\mathbf{0}0,\ \mathbf{0}1\}y \cup \{0\mathbf{1},\ 1\mathbf{1}\}y = \{00,\ 01,\ 11\}y,$$
$$\{\mathbf{1}0,\ \mathbf{1}1\}y \cup \{0\mathbf{0},\ 1\mathbf{0}\}y = \{00,\ 10,\ 11\}y.$$

These three unit clauses collapse by intersection to $\{00\}y$ (which is equivalent with $\neg a \wedge \neg b$).

A propositional formula (or SAT problem) is a special case of MDL, in which all the domains are $\{\mathbb{T}, \mathbb{F}\}$ (or any other binary set). One can cluster any number of propositional variables.

If all the variables are clustered into one, then all clauses become units and the proof reduces to intersecting all the sets. This basically corresponds to the trivial algorithm of trying exhaustively all possible assignments to propositional variables, and therefore is unfeasible for a large number of variables.

However, if the representation of the literals is efficient for a certain size of clusters (e. g. because it is supported by the computer hardware – see next section), then clustering of some number of variables may lead to a significant speed-up. Moreover, the clusters do not need to be of the same size, and also further clustering can be used during the proof, when certain domains become relatively small. A special interest and also a challenge of MDL consists in finding good criteria for efficient clustering.

Weak assignments. As previously mentioned, in case there are no more unit clauses, one must split the problem into several ones. In propositional DPLL there are allways two branches, analogously in MDL the number of branches may equal the number of the elements of the domain of the variable which is used for branching. However, some elements of the domain may be eliminated – these are the here called *weak assignments*.

In a given clause set, let $\{A_i x \vee C_i\}_{i=1}^{p}$ be all the clauses containing x, and let $D = \cup_i A_i$ be the domain of x. For two elements d, d' of D, we say that $d \preceq d'$ (d is *weaker* than d', d' is *stronger* than d) iff, for any $i \in \overline{1,p}$, d' occurs in A_i whenever d occurs in A_i (that is, there is no A_i containing d but not containing d'). The weaker assignment d can be eliminated from D, because any solution using d can be transformed in a solution using d'. In other words, if there exists no solution using d', then there is also no solution using d. On the other hand, any solution using d' may also lead to a solution using d – these additional solutions can be checked in a straightforward way after satisfiability has been decided upon.

If $d \preceq d'$ and $d' \preceq d$ than we say d is *idempotent* to d' and one of them can be eliminated because they give analogous sets of solutions.

For each element d, the set of elements which are weaker than d (and thus can be eliminated) is $D - \cup\{A_i \mid d \notin A_i\} - \{d\}$.

Elimination of weak assignments is equivalent to partitioning the domain into ideals with respect to the reverse of $\preceq$ (each subset contains together with any element all its inferiors). Such a partitioning *eliminates* x from all generated branches, because every x-literal will be either subsumed or cancelled.

Moreover, this is optimal, in the sense that it generates the smallest possible branching. Indeed, two assignments which are not comparable via $d \preceq d'$ must be in different partitions, otherwise the elimination property does not hold.

5. Implementation and Experiments

Data representation. A MDL formula $\wedge_{j=1}^{m} \vee_{k=1}^{n} A_{jk} x_k$ is completely characterized by the matrix of sets: $\langle\langle A_{jk} x_k \rangle_{k=1}^{n}\rangle_{j=1}^{m}$, where each line corresponds to a clause and each column corresponds to a variable. The domain of each variable x_k is $D_k = \cup_j A_{jk}$. The proof (e. g. by DPLL) consists in updating this matrix of sets and the vector of domains.

Therefore, clearly the efficiency of the implementation depends crucially on the representation of the sets A_{jk}, D_k, as well as on the representation of the matrix and of the vector. In the experiments presented in this paper, we use a binary representation of sets: we assume that each domain has its elements listed in a conventional order, and each A_{jk} is represented as a string of bits whose length equals the cardinality of D_k. The $h-$th bit in the string is 1 if the $h-$th element of D_k is present in A_{jk} and 0 otherwise. We illustrate the idea by exhibiting the representation of clusters of two propositional variables a, b. The table below lists the bit strings, the corresponding sets, and the formula which is encoded. (The formula which is encoded evaluates to true on the elements of the set, if we consider that a set element represents the binary pair ab.)

0000	$\{\}$	$\mathbb{F}$	1000	$\{0\}$	$\neg a \wedge \neg b$
0001	$\{3\}$	$a \wedge b$	1001	$\{0,3\}$	$a \Leftrightarrow b$
0010	$\{2\}$	$a \wedge \neg b$	1010	$\{1,2\}$	$\neg b$
0011	$\{2,3\}$	a	1011	$\{0,2,3\}$	$a \vee \neg b$
0100	$\{1\}$	$\neg a \wedge b$	1100	$\{0,1\}$	$\neg a$
0101	$\{1,3\}$	b	1101	$\{0,1,3\}$	$\neg a \vee b$
0110	$\{1,2\}$	$a \times b$	1110	$\{0,1,2\}$	$\neg a \vee \neg b$
0111	$\{1,2,3\}$	$a \vee b$	1111	$\{0,1,2,3\}$	$\mathbb{T}$

These strings of bits can be represented as a list (or array) of computer words, and then union and intersection can be computed using hardware logical operations on words: union becomes *bitwise or* and intersection becomes *bitwise and*. For instance, if we merge 5 propositional variables into a multi-variable, then its domain has $2^5 = 32$ elements, so we can represent it on a 32-bit computer word. For higher clustering factors, however, the length of the representation in words grows exponentially, thus from a certain value the positive effect of clustering will be cancelled by the higher cost of multi–word operations. Our experiments confirm the expected behavior: the running time first decreases until a certain clustering factor, and then it grows again.

This representation of multi-domain literals was in fact already used in [6], under the name of k-literals, and the results reported in that research are already relevant for the approach presented here.

The number of elementary operations on literals in unit propagation may depend significantly on the representation used for the matrix of literals. In the current implementation we use a sparse matrix with pointers, and we also keep track of the length of the clauses – similar to GRASP [7]. (Experiments with lazy data structures – e. g. using watched literals are in progress.)

Overview of the experiments. In order to demonstrate the effectiveness of multi–domain logic and to check the basic implementation principles, we implement a generalization of the DPLL method [2] in multi–domain logic.

We test our implementation on unsatisfiable problems from the SATLIB – Benchmark Problems homepage: http://www.satlib.org/. All instances are CNF formulae encoded in DIMACS (http://dimacs.rutgers.edu/). We use a Java implementation on a Core(TM)2 Duo, 2.2 GHz PC with 2 GB memory.

We experiment with unsatisfiable problems, because this gives more reliable information on the general behavior of the program. (On satisfiable problems the running time may be influenced by the randomness of early found solutions).

We investigate three main aspects of the novel method: initial clustering, clustering size, and weak assignments.

Initial clustering. The initial clustering of the propositional variables can be arbitrary. Therefore, one may choose the clusters based on the analysis of the current problem in order to fulfill certain criteria.

In the experiments presented in this paper, we use fixed-size clusters containing the variables which appear more often in the same clause, because this may lead to several unit clauses for the new multi-variable. (Of course this is just a quick fix for the purpose of demonstrating the main algorithm, and further research is necessary for identifying better clustering methods.)

For instance the problem `uuf50-01` from SATLIB, with a cluster size of 5, has no unit clauses, 25% binary clauses, and 75% ternary clauses. After preprocessing it has 5% units, 38% binaries and 57% ternaries.

This cluster preprocessing of the problem leads to significant time reduction in certain cases. In Fig. 1 one can see the average running time with and without cluster preprocessing, for the first 100 problems in the SATLIB set `uuf50-218` (uniform random 3-SAT problems).

As expected, cluster preprocessing does increase the efficiency as the cluster factor grows. More careful experiments may result better clustering methods. This is an interesting subject for further research.

Clustering size. Compared to propositional logic, MDL allows to propagate more information at once. This decreases the depth of the search space, however the width of

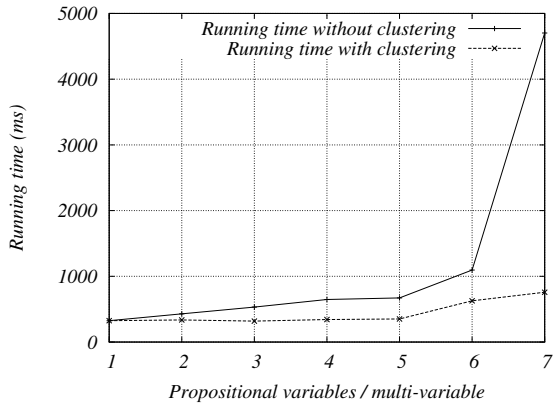

Figure 1. The effect of cluster preprocessing.

it increases because splits create more branches. In extreme cases, the total number of branches is the same, however in concrete examples the efficiency may increase due to substantial unit propagation before split.

Our experiments (Fig. 1, 2, 3) show that the increase of the clustering factor results first in the decrease of the running time, and then in its increase again due to the effect of multi–word operations. One also sees that the effect may be quite different on different kind of problems: it is almost absent on random 3-SAT problems, but quite significant on "pigeonhole" problems. In all cases, the best efficiency is obtained when the clustering factor equals the maximal size of the original clauses.

Weak assignments. The search space may decrease due to deletion of weak assignments, however detection of weak assignments also consumes time. The experiments show that deletion of weak assignments indeed increases efficiency.

The simplification technique based on this observation will be called "*D*eletion *o*f *W*eak *A*ssignments" (**DoWA**). This technique can be used as a preprocessing step, as a simplification step before each unit propagation, or in both ways.

The first variant (called DoWA in preprocessing) consists in deleting all weak assignments before starting the DPLL algorithm. This is very effective for example in case of the pigeonhole problem, which shows that weaker assignments reveal some inner structure which is hidden by CNF form. On the other hand it is not very effective in the case of random problems.

In the second variant, before each unit propagation we delete weaker assignments from the respective domain. The shorter the unit is the more likely is that unit subsumption takes place, i.e., we delete more clauses by unit propagation.

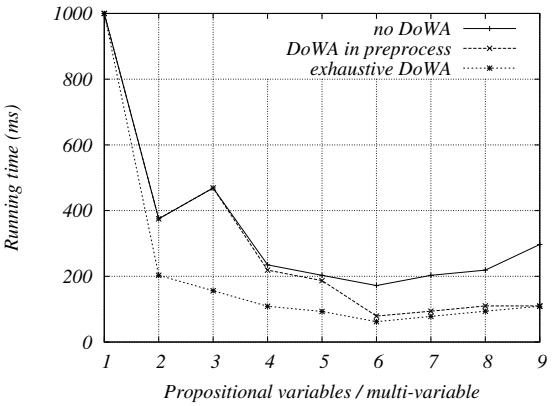

Figure 2. DoWA on pigeonhole 6.

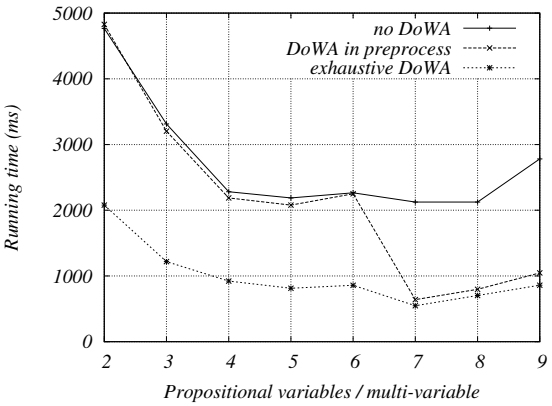

Figure 3. DoWA on pigeonhole 7.

This is a kind of lazy variant of DoWA. The greedy one would eliminate weaker assignments from each new unit. This variant has been not tested alone, but only together with the first one.

The third variant, which we call exhaustive DoWA, consists in applying the previous techniques together. It is really effective on structured problems like pigeonhole or graph coloring.

In Figures 2 and 3 we present the results of DoWA on the pigeonhole problems of size 6 and 7, respectively. The most efficient strategy is exhaustive DoWA.

The best speed-up is about 30 times and it is obtained for pigeonhole 7 with exhaustive DoWA. (For scaling reasons, the running time of 22 781 ms in case of no clustering is not shown in Fig. 3.)

6. Conclusions

This succinct theoretical investigation and the preliminary experiments demonstrate the effectiveness of the MDL approach and suggest that, at least for certain classes of problems, this approach may lead to significant increases in the efficiency of SAT solvers.

Moreover, the novelty of these ideas opens promising lines of research: efficient representations of sets, cluster preprocessing, migration of various classical techniques to MDL, etc.

References

[1] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[2] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[3] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[4] G. Kusper. *Solving and Simplifying the Propositional Satisability Problem by Sub-Model Propagation*. PhD thesis in Informatics, RISC Institute – Johannes Kepler University Linz, 2005.

[5] G. Kusper. Solving the resolution-free sat problem by submodel propagation in linear time. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):129–136, 2005.

[6] G. Kusper and L. Csőke. Better Test Results for the Graph Coloring and the Pigeonhole Problems using DPLL with k-Literal Representation. In *7th International Conference on Applied Informatics*, volume II., pages 127–135, 2007.

[7] J. P. Marques-Silva and K. A. Sakeallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. on Computers*, 48(5):506–521, 1999.

[8] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.