Universitat Autònoma de Barcelona
Facultat de Ciències
Departament d'Informàtica

TESI DOCTORAL

Systematic and Local Search Algorithms for Regular-SAT

Memòria de treball presentada per en **Ramón Béjar Torres** a la Universitat Autònoma de Barcelona per a l'obtenció del títol de Doctor en Informàtica. El treball contingut en aquesta memòria ha estat realitzat a la Universitat de Lleida sota la direcció del Dr. **Felip Manyà Serres**.

Lleida, setembre de 2000

Contents

List of Figures

List of Tables

Abstract

Regular-SAT is the problem of deciding the satisfiability of a class of many-valued propositional formulas called regular CNF formulas. A *regular CNF formula* is a classical propositional conjunctive clause form based on a generalised notion of literal, called regular literal. Given a truth value set N ($|N| \geq 2$) equipped with a total ordering $\leq$, a *regular literal* is an expression of the form $S : p$, where p is a propositional variable and S is a subset of N which is either of the form $\uparrow i = \{j \in N \mid j \geq i\}$ or of the form $\downarrow i = \{j \in N \mid j \leq i\}$ for some $i \in N$. A given many-valued interpretation (i.e., a mapping that assigns to every propositional variable an element of the truth value set N) satisfies a regular literal $S : p$ iff it assigns to p a truth value that appears in S, and satisfies a regular clause iff it satisfies at least one of its literals. A regular CNF formula is satisfiable iff there exists an interpretation that satisfies all its clauses; otherwise, it is unsatisfiable.

In this thesis we focus on the design, implementation and analysis of systematic and local search algorithms for *Regular-SAT*, and define a new and competitive generic problem solving approach which consists in modeling hard combinatorial problems as *Regular-SAT* instances and then solving the resulting encodings with algorithms for *Regular-SAT*. In particular, we pay special attention to the definition of suitable data structures for representing formulas, heuristics for efficiently exploring the search space, strategies for escaping from local minima in local search algorithms, and suitable *Regular-SAT* encodings for modeling combinatorial problems.

Concerning systematic search algorithms we have designed and implemented a variant of Regular-DP, which is a Davis-Putnam-style procedure for regular CNF formulas. Concerning local search algorithms we have designed and implemented two new families of algorithms, Regular-GSAT and Regular-WalkSAT, which are natural generalizations of two prominent families —the GSAT and WalkSAT architectures— of local search algorithms for SAT. These are the first local search algorithms described for regular CNF formulas. Our implementations of Regular-DP, Regular-GSAT and Regular-WalkSAT are the only implementations developed so far for solving *Regular-SAT*.

We have also conducted a comprehensive experimental investigation for (i) analysing and comparing Regular-DP, Regular-GSAT, and Regular-WalkSAT; (ii) studying the existence of phase transitions in *Regular Random 3-SAT*; and (iii) providing experimental evidence of the practical usefulness of the generic problem solving approach which consists in modeling hard combinatorial problems as *Regular-SAT* instances and then solving the resulting encodings using algorithms for *Regular-SAT*.

The results of the experimental investigation for *Regular Random 3-SAT* instances provide experimental evidence of the existence of phase transitions in *Regular Random 3-SAT*. The results also indicate that the location of the threshold increases logarithmically in the cardinality of the truth value set. We give a theoretical explanation of this fact by providing upper bounds on the location of the unsatisfiability threshold.

The results of the experimental investigation for realistic problems (k-colourability of graphs, round robin, and all-interval-series) provide experimental evidence of the practical usefulness of our generic problem solving approach, and highlight the importance of defining suitable encodings. At least for the combinatorial problems studied, our approach can outperform or compete with state-of-the-art problem solving approaches.

As a byproduct of the experimental investigation, we have constructed the first benchmark library, formed a representative sample of hard instances, for analysing and comparing systematic and local search algorithms for *Regular-SAT*.

Acknowledgements

This work is a collaborative achievement with my Ph.D. advisor, Felip Manyà. I am really grateful for all the help that I have received from Felip. I cannot imagine doing this thesis without him.

I thank the support given by my parents during all this time, and also the support obtained from Magda. She really knows how difficult can be to be with a person doing a Ph.D. thesis, and she has spent a lot of her patience with me. Thanks for your love Magda. I can also not forget a long list of friends that have made the years of the thesis easier to carry on: Primo, Jaume, Marc, Nacho, Itziar, Ruben, Hambo, Javi & Montse, Jordi Planes, Jordi Fo, ...

I thank Gonzalo Escalada-Imaz, Francesc Esteva, Carla P. Gomes, Reiner Hähnle, and Alvaro del Val for accepting to join my examining committee and for valuable comments and suggestions.

I thank my colleagues and friends Teresa Alsinet, Alba Cabiscol, Cèsar Fernàndez, Carles Mateu and Fermín Molina for their technical assistance and moral support. I hope all the best for the Ph.D. thesis of Alba, the new SAT girl.

I thank Javier Chavarriga for his support for obtaining the doctoral fellowship of the Universitat de LLeida.

I thank the Department of Computer Science of the Universitat de Lleida for providing a very special work environment.

This work was supported by a doctoral fellowship of the Universitat de LLeida and a doctoral fellowship of the Comissionat per a Universitats i Recerca (1998FI-00326). The CICYT (project TIC96-1038-C04-03) and la *La Paeria* (projects X080 and X090) partially supported this research.

Chapter 1

Introduction

1.1 Motivation

The satisfiability problem, or *SAT*, is the problem of determining if there is an assignment of truth values to the variables in a classical propositional formula in conjunctive normal form (CNF) that makes the formula true according to the usual rules of interpretation. In this thesis we investigate *Regular-SAT*, which is the satisfiability problem of a particular class of many-valued propositional formulas called *regular CNF formulas.*

We focus on the design, implementation and analysis of systematic and local search algorithms for *Regular-SAT*, and define a new and competitive generic problem solving approach which consists in modeling hard combinatorial problems as *Regular-SAT* instances and then solving the resulting encodings with algorithms for *Regular-SAT*.

A *regular CNF formula* is a classical propositional conjunctive clause form based on a generalised notion of literal, called regular literal. Given a domain N equipped with a total ordering $\leq$, a *regular literal* is an expression of the form $S:p$, where p is propositional variable and S, its sign, is a subset of N which is either of the form $\uparrow i = \{j \in N \mid j \geq i\}$ or of the form $\downarrow i = \{j \in N \mid j \leq i\}$ for some $i \in N$. The informal meaning of $S:p$ is "p takes one of the values in S".

Let N be the set $\{0, 1, 2\}$ with the standard order on natural numbers. An example of regular CNF formula is

$$(\downarrow 0 : p_1 \vee \downarrow 1 : p_2 \vee \uparrow 2 : p_3) \wedge (\uparrow 1 : p_1 \vee \downarrow 0 : p_2),$$

which is represented in clause form as

$$\{\{\downarrow 0 : p_1, \downarrow 1 : p_2, \uparrow 2 : p_3\}, \{\uparrow 1 : p_1, \downarrow 0 : p_2\}\}.$$

When N is considered as a truth value set, there is a natural generalization of the classical concept of satisfiability. A truth assignment, or many-valued propositional interpretation, is a mapping that assigns to every propositional variable an element of N. It satisfies a regular literal $S : p$ iff it assigns to p a truth value that appears in S. It satisfies a regular clause iff it satisfies at least one of its literals. A regular CNF formula is satisfiable iff there exists a truth assignment that satisfies all its clauses; otherwise, it is unsatisfiable.

Regular CNF formulas turn out to be a generic representation for finite-valued logics: given any finite-valued formula, one can derive a satisfiability equivalent regular CNF formula in polynomial time using the methods described in (Hähnle, 1994b; Beckert et al., 2000b). Regular CNF formulas offer a formalism for representing and solving the satisfiability problem of any finite-valued logic, and avoid the development of specific clausal forms, logical calculi and satisfiability algorithms for every finite-valued logic. Thus, the algorithms described in this thesis are of particular interest to researchers in the area of automated theorem proving in many-valued logics.

When $S : p$ is interpreted as "p is constrained to the values in S", *Regular-SAT* turns out to be a constraint satisfaction problem (CSP). A CSP consists of a finite set of variables $X = \{X_1, X_2, \ldots, X_n\}$, an associated set of finite domains $D = \{D_1, D_2, \ldots, D_n\}$ such that each X_i takes values from domain D_i, and a finite set of constraints $C = \{C_1, C_2, \ldots, C_t\}$. Each C_i is a relation over some subset of $D_i's$, i.e., $C_i \subseteq D_{i_1} \times D_{i_2} \times \ldots \times D_{i_{j(i)}}$, where the tuples in C_i are valid assignments of values to the variables $X_{i_1}, X_{i_2}, \ldots, X_{i_{j(i)}}$. A solution for a CSP is a complete assignment that satisfies every constraint.

In recent years, *SAT* has seen growing interest and activity, and has become increasingly popular. This is due to different reasons: (i) it is the first problem for which NP-completeness was established (Cook, 1971); (ii) it is easy to think about (Freeman, 1995); (iii) it captures the essence of many difficult problems in disciplines such as artificial intelligence, computer science, electrical engineering and operations research (Warners, 1999a); (iv) the discovery of new systematic and local search procedures, as well as the memory size and speed of contemporary computers, has extended the range and size of *SAT* instances that can be efficiently solved; and (v) many real-world problems can be efficiently solved by encoding them into *SAT* (e.g. circuits (Warners and vanMaaren, 1999b; Li, 2000), cryptography (Massacci and Marraro, 2000), planning (Kautz and Selman, 1996; Kautz and Selman, 1999), and scheduling (Crawford and Baker, 1994; Béjar and Manyà, 2000)).

Motivated by the success of the results obtained for *SAT*, we will investigate whether similar, or even superior, results can be obtained for *Regular-SAT*. There are several reasons to believe that we can obtain successful results for regular CNF formulas:

- *SAT* is a special case of *Regular-SAT*. If we take $N = \{0, 1\}$ as truth value set, we can transform any *SAT* instance Γ into a logically equivalent *Regular-SAT* instance Γ' by replacing every positive (negative) literal p ($\neg p$) occurring in Γ with $\uparrow 1 : p$ ($\downarrow 0 : p$).

- Proof methods such as resolution have been extended to regular CNF formulas in a straightforward way (Hähnle, 1994b; Hähnle, 1996; Manyà, 1996).

- Satisfiability algorithms and heuristics for classical CNF formulas have been generalized to regular CNF formulas quite naturally (Escalada-Imaz and Manyà, 1994; Hähnle, 1996; Manyà, 1996; Manyà et al., 1998). As we will see, the good properties of the classical algorithms remain in regular algorithms. This implies that for designing new algorithms for regular CNF formulas we do not have to start from scratch, we can take advantage of the techniques that have proven to be successful in the classical setting.

- *Regular-SAT*, like *SAT*, is one of the syntactically and conceptually simplest NP-complete problems. The design, implementation and analysis of algorithms for *Regular-SAT* tend to be easier than for other NP-complete problems such as CSPs.

- As regular CNF formulas are a more expressive representation formalism than classical CNF formulas, some problems encoded as *Regular-SAT* instances give rise to more compact encodings, which usually use a smaller number of propositional variables. It is expected that this fact will extend the range and size of problems that can be solved using *Regular-SAT* encodings and will have positive effects in the search for a solution.

One of the objectives of this thesis is the design, implementation and analysis of *efficient* algorithms for solving *Regular-SAT*. Since *Regular-SAT* is NP-complete, we have that *Regular-SAT* is a computationally *intractable* problem; the known complete algorithms for solving *Regular-SAT* require exponential time in the worst case, and it is considered unlikely that an algorithm exists that can solve all instances of *Regular-SAT* in polynomial time. We can obtain computationally *tractable* problems if we restrict the language of regular CNF formulas: *Regular-SAT* can be solved in polynomial time (i) when the clauses contain at most two literals, *Regular 2-SAT* (Manyà, 1996; Manyà, 2000), and (ii) when the clauses contain at most one positive literal (i.e., a regular literal containing a sign of the form $\uparrow i$), *Regular Horn SAT* (Hähnle, 1996; Manyà, 1996). Thus, a *good* algorithm for solving any given *Regular-SAT* instance might mean an algorithm performing well on average, or with high probability, or on a wide range of instances agreed to be important (Cook and Mitchell, 1997).

In this thesis we will consider both *systematic search* and *local search* satisfiability algorithms. Systematic search algorithms perform a search through the space of all possible truth assignments, in a systematic manner, to prove that either a given formula is satisfiable (the algorithm finds a satisfying truth assignment) or unsatisfiable (the algorithm explores the entire search space without finding any satisfying truth assignment). By contrast, local search algorithms usually do not explore the entire search space, and a given truth assignment can be considered more than once. They start typically with some randomly generated truth assignment and try to find a satisfying interpretation by iteratively changing the assignment of one propositional variable. Such changes are repeated until either a satisfying interpretation is found or a pre-set maximum number of changes is reached. This process is repeated as needed, up to a pre-set maximum number of times. The main difference among local search algorithms is the manner of selecting the variable whose assignment has to be changed.

One difficulty with local search algorithms is that they can get stuck in local minima. Another difficulty is that they are incomplete and cannot prove unsatisfiability: if a solution is found, the formula is declared satisfiable and the algorithm terminates successfully; but if the algorithm fails to find a solution, no conclusion can be drawn. However, if a formula is satisfiable, it is often determined much faster using a local search algorithm than using a systematic search algorithm.

Concerning systematic search algorithms we will deal with variants of Regular-DP (Hähnle, 1996; Manyà et al., 1998), which is a Davis-Putnam-style procedure for regular CNF formulas. In particular, we will define the data structures for representing formulas that we used in our implementations, and a new branching heuristic for exploring more efficiently the search space.

Concerning local search algorithms we will describe two new families of algorithms, Regular-GSAT and Regular-WalkSAT, which are natural generalizations of two prominent families —the GSAT (Selman et al., 1992) and WalkSAT (Selman et al., 1994) architectures— of local search algorithms for *SAT*. In particular, we will give a detailed description of the data structures that we have defined and used to represent formulas in our implementations, and we will define several strategies for escaping from local minima.

Regular-GSAT and Regular-WalkSAT are, to our best knowledge, the first families of local search algorithms described for regular CNF formulas. Moreover, our efficient implementations of Regular-DP, Regular-GSAT and Regular-WalkSAT are the only implementations developed so far for solving *Regular-SAT*.

Another of the objectives of this thesis is the analysis and comparison of satisfiability algorithms for Regular CNF formulas. Unfortunately, there is no satisfactory theoretical method for analysing algorithms: worst-case analysis does not provide a realistic view of the behavior of the algorithms, and probabilistic analysis is too complex to handle, for satisfiability algorithms, with the currently available techniques. The more suitable form of analysing and comparing satisfiability algorithms remains experimental evaluation (Warners, 1999a).

We will perform an experimental investigation for analysing and comparing Regular-DP, Regular-GSAT and Regular-WalkSAT. The results of this experimental investigation, for realistic problems, will provide experimental evidence of the practical usefulness of the generic problem solving approach which consists in modeling hard combinatorial problems as *Regular-SAT* instances and then solving the resulting encodings using algorithms for *Regular-SAT*.

There are publicly available sets of benchmark instances for *SAT* which are widely used to conduct experimental investigations.[1] As no set of bench-

[1] One of the oldest is provided by the benchmark library developed for the Second DIMACS International Algorithm Implementation Challenge in 1993: `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf/`

Another is the benchmark library created for the International Competition and Symposium on Satisfiability Testing held in Beijing in 1996: `http://www.cirl.uoregon.edu/crawford/beijing/`

SATLIB (Hoos and Stützle, 2000b) is the most complete and recently created benchmark library: `http://www.informatik.tu-darmstadt.de/AI/SATLIB/`

mark instances for *Regular-SAT* has been created so far, we will start our experimental investigation by constructing a benchmark library formed by a representative sample of hard *Regular-SAT* instances.

On the one hand, we will conduct the experimental investigation with instances of *Regular Random 3-SAT*, which is a subproblem of *Regular-SAT*. A *Regular Random 3-SAT* instance contains a given number of regular clauses, and each clause is produced by choosing uniformly at random three non-tautological regular literals, with different propositional variable, from the set of literals that can be formed from a given set of propositional variables and a given truth value set.

We will investigate the occurrence of phase transitions in *Regular Random 3-SAT* in order to identify a class of instances which are difficult to solve for both systematic and local search algorithms. Moreover, we will perform a series of experiments to study how the location of the threshold varies as we increase the cardinality of the truth value set, and how Regular-DP, Regular-GSAT and Regular-WalkSAT behave on the region of the phase transition that contains the more computationally difficult instances.

On the other hand, we will conduct the experimental investigation with instances of more realistic problems: the k-colourability problem of graphs, the round robin problem, and the all-interval-series problem. These problems have computationally difficult instances which are standard benchmarks in the communities of constraint programming, operations research, and *SAT*. We will consider the same set of benchmark instances that were used in previous empirical studies in order to compare our experimental results with other published results. We will also define suitable encodings for modeling these hard combinatorial problems as *SAT* and as *Regular-SAT* instances.

The ultimate goal of the experimental investigation is to have a clear picture of the strengths and weaknesses of our generic problem solving approach; improve our implementations, and design better heuristics and algorithms; and provide experimental evidence that, at least for the combinatorial problems studied, our problem solving approach can outperform or compete with state-of-the-art problem solving approaches.

1.2 Contributions

Some of the results presented in this thesis have already been published as journal articles or in conference proceedings:

(1) F. Manyà, R. Béjar, and G. Escalada-Imaz. The satisfiability problem in regular CNF-formulas. *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, 2(3):116–123, 1998.

(2) R. Béjar and F. Manyà. Phase transitions in the Regular Random 3-SAT problem. In *Proceedings of Primer Congrés Català d'Intel.ligència Artificial, CCIA'98, Tarragona, Spain*, pages 39–45, 1998.

(3) R. Béjar and F. Manyà. A comparison of systematic and local search algorithms for regular CNF formulas. In *Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'99, London, England*, pages 22–31. Springer LNAI 1638, 1999.

(4) R. Béjar and F. Manyà. Phase transitions in the Regular Random 3-SAT problem. In *Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems, ISMIS'99, Warsaw, Poland*, pages 292–300. Springer LNAI 1609, 1999.

(5) R. Béjar and F. Manyà. Solving combinatorial problems with regular local search algorithms. In *Proceedings of the Sixth International Conference on Logic for Programming and Automated Reasoning, LPAR'99, Tbilisi, Republic of Georgia*, pages 33–43. Springer LNAI 1705, 1999.

(6) R. Béjar and F. Manyà. Solving the round robin problem using propositional logic. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence, AAAI-2000, Austin, USA*, pages 262–266, 2000.

The main contributions of this thesis, and partly reported in the above publications, can be summarized as follows:

- Design and implementation of a variant of Regular-DP, equipped with a new branching heuristic.
- Design and implementation of two new families of local search algorithms for *Regular-SAT*: Regular-GSAT and Regular-WalkSAT.
- Construction of the first benchmark library, formed a representative sample of hard instances, for analysing and comparing systematic and local search algorithms for *Regular-SAT*.
- Experimental evidence of the existence of phase transitions in *Regular Random 3-SAT*, as well as study of the change of the location of the threshold as the cardinality of the truth value set is varied.
- Development of a new generic problem solving approach, which consists in using the logic of regular CNF formulas to model combinatorial problems and then solving the resulting encodings using regular satisfiability algorithms.
- The first experimental evaluation of algorithms for *Regular-SAT*. Within the empirical study, our original results include:
  - evidence that our branching heuristic for Regular-DP outperforms previous branching heuristics on *Regular Random 3-SAT* instances of the hard region of the phase transition.
  - evidence that local search algorithms (Regular-GSAT and Regular-WalkSAT) outperform systematic search algorithms (Regular-DP) on satisfiable *Regular Random 3-SAT* instances of the hard region of the phase transition.
  - evidence that our problem solving approach is a very competitive approach. For the round robin problem, the performance

of our *Regular-SAT*-based approach is similar to the performance of the *SAT*-based approach. For the k-colourability problem of graphs and the all-interval-series problem, our *Regular-SAT*-based approach outperforms the *SAT*-based approach.

– evidence that the round robin problem can be efficiently solved by reducing it to a satisfiability problem. We have provided the first solutions of this problem using both a *SAT*-based and a *Regular-SAT*-based problem solving approach.

1.3 Thesis overview

The thesis is organized in six chapters and one appendix, whose contents are summarized below.

Chapter 1: Introduction

In this chapter we first discuss and motivate the importance of designing, implementing and evaluating satisfiability algorithms for regular CNF formulas, as well as of encoding hard combinatorial problems as *Regular-SAT* instances. Then, we describe the objectives, main results and structure of the thesis.

Chapter 2: Regular CNF formulas

In this chapter we present regular CNF formulas as a subclass of signed CNF formulas, define formally their syntax and semantics, and summarize the main complexity results of the satisfiability problem for both kinds of formulas.

Chapter 3: Satisfiability Algorithms

In this chapter we give a detailed description of the systematic and local search algorithms for *Regular-SAT* that we have designed and implemented in this thesis. Concerning systematic search algorithms we describe Regular-DP, which is a Davis-Putnam-style procedure for regular

CNF formulas. Concerning local search algorithms we describe two new families of algorithms, Regular-GSAT and Regular-WalkSAT, which are natural generalizations of two prominent families —the GSAT and WalkSAT architectures— of local search algorithms for *SAT*. In particular, we pay special attention to the definition of clever heuristics for exploring the search space and suitable data structures for representing formulas.

Chapter 4: Phase Transition in *Regular Random 3-SAT*

In this chapter we first provide experimental evidence of the existence of phase transitions in *Regular Random 3-SAT* when the satisfiability of the instances is determined with Regular-DP. Then, we report a series of experiments that indicate that the location of the threshold increases logarithmically in the cardinality of the truth value set. We also give a theoretical explanation of this fact by providing upper bounds on the location of the unsatisfiability threshold. Finally, we compare the performance of Regular-DP, Regular-GSAT, and Regular-WalkSAT on instances of the hard region of the phase transition.

Chapter 5: Experimental Investigation

In this chapter we describe a comprehensive experimental investigation conducted for evaluating the algorithms for *Regular-SAT* that we have designed and implemented, as well as for providing experimental evidence of the practical usefulness of the generic problem solving approach which consists in modeling hard combinatorial problems as *Regular-SAT* instances and then solving the resulting encodings using algorithms for *Regular-SAT*. The combinatorial problems considered in the experimental investigation were the k-colourability problem of graphs, the round robin problem, and the all-interval-series problem.

Chapter 6: Conclusions

In this chapter we briefly summarize the main contributions of the thesis, and point out some open problems and future research perspectives that we plan to tackle in the near future.

Appendix A: Format of Regular CNF Formulas

In this appendix we describe the format that a file containing a regular CNF formula must have. This file format is accepted by the regular satisfiability algorithms described in this thesis, and is an adaptation of the file format defined by the DIMACS Challenge Committee for classical CNF formulas.

Finally, the reader can find the references appearing in the text and the index.

Chapter 2

Regular CNF Formulas

2.1 Introduction

In this chapter we present regular CNF formulas as a subclass of signed CNF formulas, define formally their syntax and semantics, and summarize the main complexity results of the satisfiability problem for both kinds of formulas. The main difference between signed and regular CNF formulas lies in the notion of literal. In signed CNF formulas, a literal is an expression of the form $S : p$, where p is a propositional variable and S, its sign, is any subset of a non-empty, finite and not necessarily ordered truth value set.

In our informal definition of the logic of regular CNF formulas we assumed that a total ordering was associated with the truth value set (cf. page 11). However, regular CNF formulas over truth value sets equipped with a partial ordering have also been considered in the literature (e.g. (Beckert et al., 1999; Beckert et al., 2000a; Lu et al., 1993; Lu et al., 1998; Sofronie-Stokkermans, 2000)). Thus, in this chapter we will use the term *totally-ordered regular CNF formulas* for formulas over domains equipped with a total ordering, but as all the contributions of this thesis are for this particular class of formulas, we do not make a distinction between totally-ordered and partially-ordered formulas in the remaining chapters.

The reader interested in a recent survey on signed and regular CNF formulas is invited to consult (Beckert et al., 2000b). Surveys on deduction methods for many-valued logics that contain results of signed and regular CNF formulas are (Hähnle, 1994a; Hähnle and Escalada-Imaz, 1997; Hähnle, 2001).

This chapter is organized as follows. In Section 2.2 we define formally the logic of signed and regular CNF formulas. In Section 2.3 we review the main complexity results of the satisfiability problem for signed CNF formulas and their subclasses. In the sections below we follow closely the presentation of (Beckert et al., 2000b; Beckert et al., 2000a).

2.2 Syntax and semantics of signed and regular CNF formulas

We assume that a signature, i.e., a denumerable set of propositional variables, is given. To form signed literals, the propositional variables are adorned with a sign that consists of a finite set of truth values.

Definition 2.1 *A* truth value set *is a non-empty, finite set* $N = \{i_1, i_2, \ldots, i_n\}$ *where* $n \in \mathbb{N}$. *The cardinality of* N *is denoted by* $|N|$ *and* $|N| \geq 2$. *A partial order* $\leq$ *is associated with* N, *which may be the empty order.*

Definition 2.2 *A* sign *is a set* $S \subseteq N$ *of truth values. A* signed literal *is an expression of the form* $S : p$, *where* S *is a sign and* p *is a propositional variable. The* complement *of a signed literal* $S : p$, *denoted by* $\overline{S} : p$, *is* $(N \setminus S) : p$.

A signed clause *is a finite set of signed literals. A signed clause containing exactly one literal is called a* signed unit clause*; and a signed clause containing exactly two literals is called a* signed binary clause. *The empty signed clause is denoted by* $\Box$.

A signed CNF formula *is a finite set of signed clauses. A signed CNF formula whose clauses are binary is called a* signed 2-CNF formula.

The clauses of a signed CNF formula are implicitly conjunctively connected; and the literals in a signed clause are implicitly disjunctively connected. Sometimes we will use $S_1 : p_1 \vee \cdots \vee S_k : p_k$ to represent a signed clause of the form $\{S_1 : p_1, \ldots , S_k : p_k\}$.

Example 2.1 *Let $N = \{1, 2, 3\}$ be ordered using the standard order on natural numbers. The following expression is an example of signed CNF formula:*

$$\begin{aligned}&\{\{\{2,3\}{:}p_3, \{1,2\}{:}p_4\}, \{\{1\}{:}p_3, \{2,3\}{:}p_3, \{3\}{:}p_2\},\\ &\ \{\{1\}{:}p_3, \{3\}{:}p_3, \{2\}{:}p_2, \{2\}{:}p_4\}, \{\{2,3\}{:}p_3, \{3\}{:}p_4\},\\ &\ \{\{2,3\}{:}p_3, \{1,3\}{:}p_4\}, \{\{3\}{:}p_3, \{1,2\}{:}p_2\}\}\end{aligned}$$

Definition 2.3 *A signed literal $S{:}p$* subsumes *a signed literal $S'{:}p'$, denoted by $S{:}p \subseteq S'{:}p'$, iff $p = p'$ and $S \subseteq S'$.*

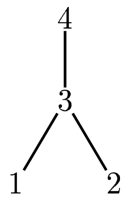
Definition 2.4 *The* size *of a signed clause C, denoted by $|C|$, is its cardinality. The size of a signed formula Γ, denoted by $|\Gamma|$, is the sum of the sizes of its signed clauses.*

Definition 2.5 *For each element i of the truth value set N, let $\uparrow i$ denote the sign $\{j \in N \mid j \geq i\}$ and let $\downarrow i$ denote the sign $\{j \in N \mid j \leq i\}$, where $\leq$ is the (non-empty) partial order associated with N. A sign S is* regular *if it is identical to $\uparrow i$ or to $\downarrow i$ for some $i \in N$.*

A signed literal $S : p$ is a regular literal *if (a) its sign S is regular or (b) its sign $S = \overline{S'}$ is the complement of a regular sign S'. A signed clause (a signed CNF formula) is a* regular clause *(a* regular CNF formula*) if all its literals are regular.*

Observe that the signs of the regular literals occurring in totally-ordered regular CNF formulas are always regular signs. This is due to the fact that the complement of a regular sign is also a regular sign in case N is equipped with a total order. Because of this, in the remaining chapters we assume that the signs of regular literals are either of the form $\uparrow i$ or of the form $\downarrow i$ for some $i \in N$. As shown in Example 2.2, this does not happen when N is equipped with a partial order.

Example 2.2 *Let the truth value set $N = \{1, 2, 3, 4\}$ be ordered as shown below, i.e., we use the standard order on natural numbers except that* 1 *and* 2 *are incomparable.*



Then, the signs $\uparrow 1 = \{1, 3, 4\}$ and $\downarrow 1 = \{1\}$ are regular; and $\overline{\uparrow 1} = \{2\}$ and $\overline{\uparrow 3} = \{1, 2\}$ are complements of regular signs. The signs $\{3\}$ and $\{1, 4\}$ are neither regular nor complements of regular signs.

The complement $\overline{\uparrow 3}$ of the regular sign $\uparrow 3$ is not *regular as it cannot be represented as $\uparrow i$ or $\downarrow i$ for any $i \in N$. Thus, a regular literal can have a sign that is not regular (but is the* complement *of a regular sign).*

Definition 2.6 *A regular sign S is of* positive *(resp.* negative*)* polarity *if it can be represented as $\uparrow i$ (resp. $\downarrow i$) for some $i \in N$. A regular literal is of* positive *(*negative*)* polarity *if its sign is of positive (negative) polarity.*

A regular clause is a regular Horn clause *if it contains at most one literal of positive polarity and the signs of all its other literals are complements of signs with positive polarity. A regular CNF formula is a* regular Horn formula *if all its clauses are regular Horn clauses.*

Example 2.3 *Using the truth value set N and the associated ordering from the previous example, the clause $\uparrow 1 : p$, the clause $\uparrow 2 : p \vee \overline{\uparrow 3} : q$, and the clause $\uparrow 4 : q$ are Horn clauses. The regular clause $\uparrow 1 : p \vee \uparrow 2 : q$ is* not *a Horn clause as it contains more than one literal of positive polarity.*

The polarities assigned to regular literals give rise to the generalized notion of Horn clauses introduced in Definition 2.6. The particular case where N is lattice-ordered and S is an order filter is investigated in annotated logic programming (Kifer and Subrahmanian, 1992) (there, S is called an *annotation*); therefore, annotated logic programs can be considered to be a particular class of signed CNF formulas.

Definition 2.7 *An* interpretation *is a mapping that assigns to every propositional variable an element of the truth value set. An interpretation I* satisfies *a signed literal $S : p$ iff $I(p) \in S$. It* satisfies *a signed clause C iff it satisfies at least one of the literals in C; and it* satisfies *a signed CNF formula Γ iff it satisfies all clauses in Γ.*

A signed CNF formula (a signed clause) is satisfiable *iff it is satisfied by at least one interpretation; otherwise it is* unsatisfiable.

By definition, the empty signed clause is unsatisfiable and the empty signed CNF formula is satisfiable.

The main semantic difference between classical and signed CNF formulas arises in the definition of interpretation, where the number of truth values is greater than two. Otherwise, the concept of satisfiability of signed clauses and CNF formulas is the same as the classical one, but with respect to many-valued interpretations.

2.3 Satisfiability problems of signed and regular CNF formulas

It is well-known that *SAT* (the problem of deciding whether an arbitrary classical CNF formula is satisfiable) is NP-complete (Cook, 1971). It is, however, polynomially solvable under certain restrictions. Two important and well investigated subproblems that admit linear-time algorithms are *Horn SAT* (*SAT* in case all clauses of the formula have at most one positive literal) (Dowling and Gallier, 1984; Minoux, 1988; Escalada-Imaz, 1989a; Scutellà, 1990; Ghallab and Escalada-Imaz, 1991), and *2-SAT* (*SAT* in case all clauses of the formula have at most two literals) (Even et al., 1976; Aspvall et al., 1979; Escalada-Imaz, 1989b; del Val, 2000).

In this section we first introduce the satisfiability problem of arbitrary signed and regular CNF formulas. Then, we summarize the known results on the complexity of *Horn SAT* and *2-SAT* in the signed setting. Since arbitrary signed literals do not have the concept of polarity, we will consider *Horn SAT* only for regular CNF formulas.

Satisfiability problems of signed and regular CNF formulas have as input parameters both the formula Γ to be tested for satisfiability and the truth value set N. Thus, *Signed-SAT* (*Regular-SAT*) is the problem of deciding for an arbitrary signed (regular) CNF formula Γ over an arbitrary truth value set N, whether there is an interpretation over N satisfying Γ. One also considers decision problems where N is not an input parameter but fixed, which is denoted by attaching the fixed truth value set N as an index to the name of the decision problem. Thus, given a fixed truth value set N, *Signed-SAT*$_N$ (*Regular-SAT*$_N$) is the problem of deciding, for a signed (regular) CNF formula Γ over N, whether there is an interpretation over N satisfying Γ. In case N is not an input parameter, there is the additional computational cost of determining the finite set of different truth values that occur in Γ.

Signed-SAT, *Signed-SAT*$_N$, *Regular-SAT*, and *Regular-SAT*$_N$ are NP-complete: they clearly belong to NP —a non-deterministic algorithm can guess a satisfying interpretation and check that it satisfies the formula in polynomial time—, and they can be proved to be NP-hard by reducing *SAT* to them.

2.3.1 The *Regular Horn SAT* problem

Horn SAT for totally-ordered regular CNF formulas was investigated first: it is known that *Regular Horn SAT*$_N$ can be solved in time linear in the size n of the formula, and *Regular Horn SAT* can be solved in time proportional to $n \log n$ in the worst case (Hähnle, 1996; Manyà, 1996). An algorithm for deciding the satisfiability of a particular subclass of regular Horn formulas appeared before (Escalada-Imaz and Manyà, 1994).

If N is a finite lattice, *Regular Horn SAT* is solvable in time linear in the size of the formula and polynomial in the cardinality of N (Beckert et al., 1999). For distributive lattices, the more precise bound $n \cdot |N|^2$ was found independently (Sofronie-Stokkermans, 1998).

2.3.2 The *Signed 2-SAT* problem

Signed 2-SAT$_N$ for $|N| \geq 3$ and, thus, *Signed 2-SAT* are both NP-complete (Manyà, 1996; Beckert et al., 1999). However, *Signed 2-SAT* is polynomially solvable in case all the signs occurring in the formula are singletons (Manyà, 1996; Manyà, 2000).

Regular 2-SAT$_N$ and *Regular 2-SAT* are NP-complete (Beckert et al., 2000a). However, *Regular 2-SAT*$_N$ and *Regular 2-SAT* are polynomially solvable in case the ordering of N is total (Manyà, 1996; Manyà, 2000). Recently, the complexity of *Regular 2-SAT*$_N$ for non-total orderings has been investigated in more detail, and it is known that: (i) *Regular 2-SAT*$_N$ and *Regular 2-SAT* are NP-complete even if the truth value set N is a complete, distributive or modular lattice; and (ii) these problems are polynomially solvable in case N is a lattice and all signs occurring in the formulas are regular (signs that are complements of regular signs but are not regular themselves are excluded).

Chapter 3

Satisfiability Algorithms

3.1 Introduction

In this chapter we give a detailed description of the systematic and local search algorithms for *Regular-SAT* that we have designed and implemented in this thesis. Concerning systematic search algorithms we describe Regular-DP (Hähnle, 1996; Manyà et al., 1998), which is a Davis-Putnam-style procedure for regular CNF formulas. Concerning local search algorithms we describe two new families of algorithms, Regular-GSAT and Regular-WalkSAT, which are natural generalizations of two prominent families —the GSAT and WalkSAT architectures— of local search algorithms for SAT. In particular, we pay special attention to the definition of clever heuristics for exploring the search space and suitable data structures for representing formulas, because these factors are decisive to get efficient implementations. The programming language used in our implementations is C++.

Systematic search algorithms perform a search through the space of all possible truth assignments, in a systematic manner, to prove that either a given formula is satisfiable (the algorithm finds a satisfying truth assignment) or unsatisfiable (the algorithm explores the entire search space without finding any satisfying truth assignment). By contrast, local search algorithms usually do not explore the entire search space, and a given truth assignment can be considered more than once. They start typically with some randomly generated interpretation and try to find a satisfying interpretation by iteratively changing the assignment of one propositional variable. Such changes are repeated until either a satisfying interpretation is found or a pre-set maximum number of changes is reached. This process is repeated as needed, up to a pre-set maximum number of times. The main difference among local search algorithms is the manner of selecting the variable whose assignment has to be changed.

One difficulty with local search algorithms is that they can get stuck in local minima. Another difficulty is that they are incomplete and cannot prove unsatisfiability: if a solution is found, the formula is declared satisfiable and the algorithm terminates successfully; but if the algorithm fails to find a solution, no conclusion can be drawn. However, if a formula is satisfiable, it is often determined much faster using a local search than using a systematic search algorithm.

As far as we know, Regular-GSAT and Regular-WalkSAT are the first families of local search algorithm described for regular CNF formulas. Moreover, our efficient implementations of Regular-DP, Regular-GSAT and Regular-WalkSAT are the first implementations developed so far for solving *Regular-SAT*.

The algorithms for *Regular-SAT* described in this thesis are generalizations of well-known algorithms for *SAT*. In case $|N| = 2$, our regular algorithms work exactly as their classical counterparts and exhibit identical behaviour.

This chapter is organized as follows. In Section 3.2 we describe Regular-DP and define a new branching heuristic; in Section 3.3 we describe the data structures used in our implementation of Regular-DP; in Section 3.4 we describe the Regular-GSAT architecture; in Section 3.5 we describe the Regular-WalkSAT architecture; and in Section 3.6 we first describe the data structures of the latest versions of (classical) GSAT and WalkSAT, and then the data structures of Regular-GSAT and Regular-WalkSAT used in our implementations; and in Section 3.7 we discuss which local search algorithms for *Regular-SAT* are probabilistically approximately complete.

3.2 The Regular-DP procedure

In this section we first describe Regular-DP (Hähnle, 1996) and then a new branching heuristic. Regular-DP is based on the following rules:

Regular one-literal rule: given a regular CNF formula Γ containing a regular unit clause $\{S:p\}$,

1. remove all clauses containing a literal subsumed by $\{S:p\}$; i.e., all clauses containing a literal $S':p$ such that $S \subseteq S'$;
2. delete all occurrences of literals $S'':p$ such that $S \cap S'' = \emptyset$.

Regular branching rule: reduce the problem of determining whether a regular CNF formula Γ is satisfiable to the problem of determining whether $\Gamma \cup \{S:p\}$ is satisfiable or $\Gamma \cup \{\overline{S}:p\}$ is satisfiable, where $S:p$ is a regular literal occurring in Γ.

The pseudo-code of Regular-DP is shown in Figure 3.1. It returns true (false) if the input regular CNF formula Γ is satisfiable (unsatisfiable). First, it applies repeatedly the regular one-literal rule and derives a simplified formula Γ'. Once the formula cannot be further simplified, it selects a regular literal $S:p$ of Γ', applies the branching rule and solves recursively the problem of deciding whether $\Gamma' \cup \{S:p\}$ is satisfiable or $\Gamma' \cup \{\overline{S}:p\}$ is satisfiable. As such subproblems contain a regular unit clause, the regular one-literal rule can be applied again. Regular-DP terminates when some subproblem is shown to be satisfiable by deriving the regular empty CNF formula or all the subproblems are shown to be unsatisfiable by deriving the regular empty clause in all of them. In the pseudo-code, $\Gamma_{S:p}$ denotes the formula obtained

```
procedure Regular-DP
Input: a regular CNF formula $\Gamma$
Output: true if $\Gamma$ is satisfiable and false if $\Gamma$ is unsatisfiable
 begin
   if $\Gamma = \emptyset$ then return true;
   if $\Box \in \Gamma$ then return false;
   /* regular one-literal rule*/
   if $\Gamma$ contains a unit clause $\{S':p\}$ then Regular-DP($\Gamma_{S':p}$);
   let $S:p$ be a regular literal occurring in $\Gamma$;
   /* regular branching rule */
   if Regular-DP($\Gamma_{S:p}$) then return true;
   else return Regular-DP($\Gamma_{\overline{S:p}}$);
 end
```

Figure 3.1: The Regular-DP procedure

after applying the regular one-literal rule to a regular CNF formula Γ using the regular unit clause $\{S:p\}$.

Regular-DP can be viewed as the construction of a proof tree using a depth-first strategy. The root node contains the input formula and the remaining nodes represent applications of the regular one-literal rule. Otherwise stated, there is one node for each recursive call of Regular-DP. When all the leaves contain the regular empty clause, the input formula is unsatisfiable. When some leaf contains the regular empty formula, the input formula is satisfiable.

The (classical) Davis-Putnam procedure (Davis and Putnam, 1960; Davis et al., 1962) is a particular case of Regular-DP. In order to obtain the Davis-Putnam procedure, we take $N = \{0,1\}$ and represent a classical positive (negative) literal p ($\neg p$) by the regular literal $\uparrow 1 : p$ ($\downarrow 0 : p$).

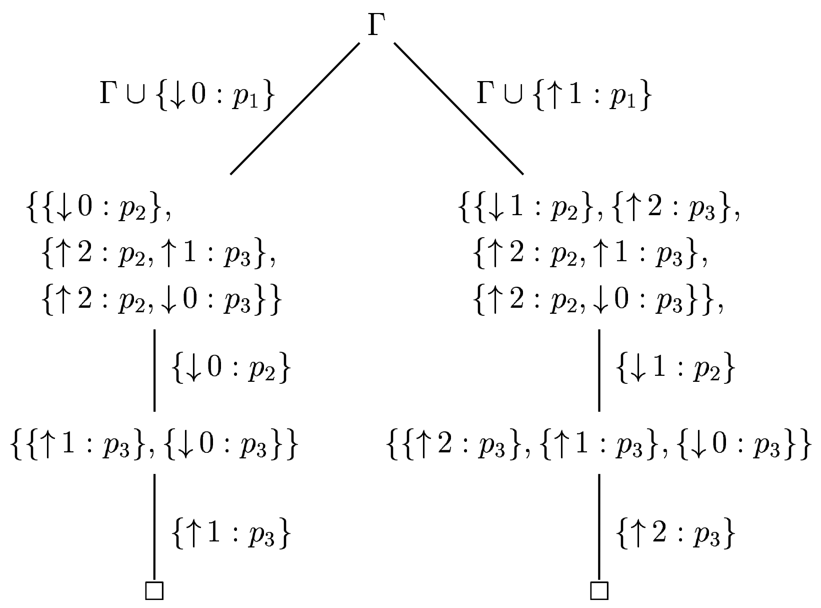

Γ
$\Gamma \cup \{\downarrow 0 : p_1\}$
$\Gamma \cup \{\uparrow 1 : p_1\}$
$\{\{\downarrow 0 : p_2\}, \{\uparrow 2 : p_2, \uparrow 1 : p_3\}, \{\uparrow 2 : p_2, \downarrow 0 : p_3\}\}$
$\{\{\downarrow 1 : p_2\}, \{\uparrow 2 : p_3\}, \{\uparrow 2 : p_2, \uparrow 1 : p_3\}, \{\uparrow 2 : p_2, \downarrow 0 : p_3\}\},$
$\{\downarrow 0 : p_2\}$
$\{\downarrow 1 : p_2\}$
$\{\{\uparrow 1 : p_3\}, \{\downarrow 0 : p_3\}\}$
$\{\{\uparrow 2 : p_3\}, \{\uparrow 1 : p_3\}, \{\downarrow 0 : p_3\}\}$
$\{\uparrow 1 : p_3\}$
$\{\uparrow 2 : p_3\}$
$\Box$
$\Box$

Figure 3.2: A proof tree created by Regular-DP

Example 3.1 *Let $N = \{0, 1, 2\}$, and let Γ be the following regular CNF formula:*

$$\{\{\downarrow 0 : p_1, \downarrow 1 : p_2\}, \{\uparrow 1 : p_1, \downarrow 0 : p_2\}, \{\downarrow 0 : p_1, \uparrow 2 : p_3\},$$
$$\{\uparrow 2 : p_2, \uparrow 1 : p_3\}, \{\uparrow 2 : p_2, \downarrow 0 : p_3\}\}$$

Figure 3.2 shows the proof tree created by Regular-DP when the input is Γ. Edges labelled with a regular CNF formula indicate the application of the regular branching rule and the regular one-literal rule using the regular unit clause in the label. The nodes that contain a regular CNF formula resulting of first applying the regular branching rule and then applying the regular one-literal rule are called branching nodes*. Edges labelled with a regular unit clause indicate the application of the regular one-literal rule using that clause.*

In our experimental investigation we will use both the run time and the number of branching nodes as a measure of the computational difficulty of solving *Regular-SAT* instances with Regular-DP. We prefer branching nodes to arbitrary nodes because the worst-case time complexity of applying the regular one-literal is linear in the length of the regular CNF formula.

The performance of Regular-DP depends dramatically on the heuristic that selects the next literal to which the branching rule is applied. A branching heuristic for Regular-DP, which is an extension of the two-sided Jeroslow-Wang rule, was defined in (Hähnle, 1996): given a regular CNF formula Γ, such a heuristic selects a regular literal L occurring in Γ that maximizes $J(L) + J(\overline{L})$, where

$$J(L) = \sum_{\substack{\exists L' : L' \subseteq L \\ L' \in C \in \Gamma}} 2^{-|C|}, \tag{3.1}$$

$\overline{L}$ is the complement of literal L, and $L' \subseteq L$ denotes that literal L' subsumes literal L.

Taking into account the work of (Hooker and Vinay, 1995) on the Jeroslow-Wang rule in the classical setting, we propose the following definition of $J(L)$:

$$J(L) = \sum_{\substack{\exists L' : L' \subseteq L \\ L' \in C \in \Gamma}} \left(\prod_{S:p \in C} \frac{|N| - |S|}{2(|N| - 1)} \right). \tag{3.2}$$

Hähnle's definition of $J(L)$ assigns a bigger value to those regular literals L subsumed by regular literals L' that appear in many small clauses. This way, when Regular-DP branchs on $\overline{L}$, the probability of deriving new regular unit clauses is bigger. Our definition of $J(L)$ takes into account the length of regular signs as well. This fact is important because regular literals with small signs have a bigger probability of being eliminated during the application of the regular one-literal rule. Observe that in the case that $|N| = 2$ we get the same equation.

In the following we will refer to the branching heuristic that uses (3.1) to calculate $J(L) + J(\overline{L})$ as RH-1, and the branching heuristic that uses (3.2) as RH-2. In Section 4.6 we provide experimental evidence that Regular-DP with RH-2 (Regular-DP/RH-2) outperforms Regular-DP with RH-1 (Regular-DP/RH-1) on a class of randomly generated instances.

3.3 Data structures for Regular-DP

In this section we describe the data structures that we have used in our implementation of Regular-DP. These data structures, inspired by the data structures defined in (Manyà, 1996), are the following ones:

- A regular CNF formula is represented as a doubly-linked list of its clauses (clause list).
- A regular clause is represented by a clause head, which has a counter of its length, and a doubly-linked list of its literals (clause literal list). Each literal has a pointer to the clause head.
- We maintain a list of regular literals occurring in regular unit clauses (unit-clause literal list).
- We maintain a doubly-linked list of the different variables occurring in the formula (variable list), but we also provide direct access to such variables using an array of pointers.
- For each variable p we have a doubly-linked list of the different positive signs (positive sign list) and a doubly-linked list of the different negative signs (negative sign list) occurring in regular literals with the variable p. For each sign S we have in turn a doubly-linked list of all the occurrences of literals of the form $S:p$ (literal occurrence list). Sign lists are sorted by cardinality of signs.

Example 3.2 *Figure 3.3 shows the data structure defined above for the fol-*

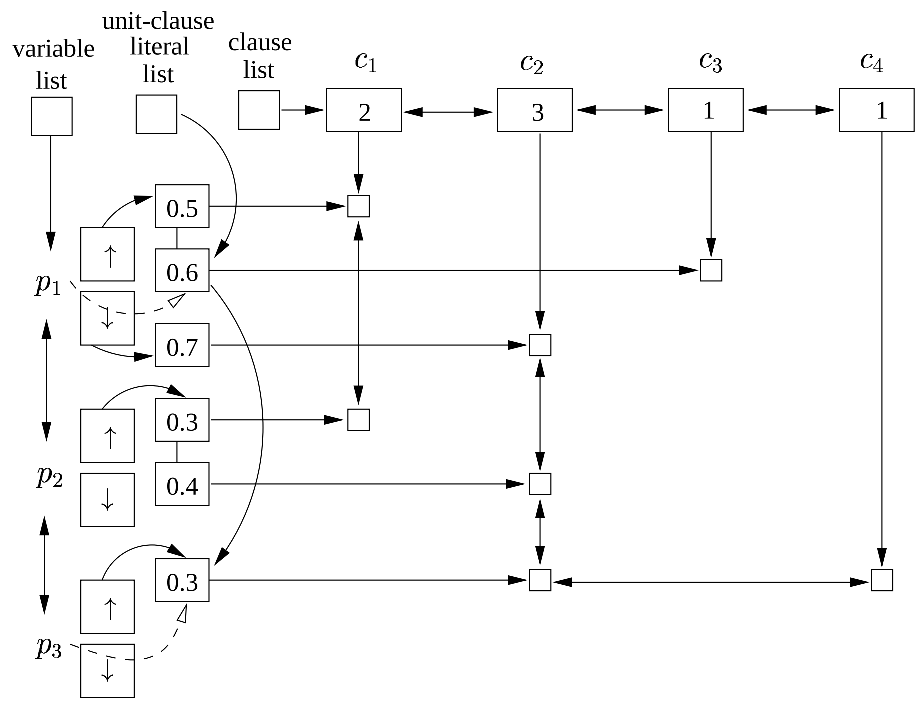


Figure 3.3: Data structures for Regular-DP

lowing regular CNF formula:

$$\{\{\uparrow 0.5 : p_1, \uparrow 0.3 : p_2\}, \{\downarrow 0.7 : p_1, \uparrow 0.4 : p_2, \uparrow 0.3 : p_3\},$$
$$\{\uparrow 0.6 : p_1\}, \{\uparrow 0.3 : p_3\}\}$$

Observe that if we see the representation of the formula as a matrix, rows are literal occurrence lists and columns are clause literal lists.

In the following we will examine the worst-case time complexity of the main operations appearing in procedure Regular-DP.

operation	complexity
$\Gamma =^? \emptyset$	$\mathcal{O}(1)$
$\Box \in^? \Gamma$	$\mathcal{O}(1)$
RH-1(Γ)	$\mathcal{O}(\vert\Gamma\vert)$
RH-2(Γ)	$\mathcal{O}(\vert\Gamma\vert)$
regular-one-literal-rule(Γ)	$\mathcal{O}(\vert\Gamma\vert)$

We maintain a global counter of clauses and so we can decide in constant time if $\Gamma = \emptyset$.

For each variable p, we have a pointer to the positive (resp. a pointer to the negative) literal $S:p$ of the unit-clause literal list, where S is the sign with smallest cardinality among the positive (resp. negative) literals $S':p$ of regular unit clauses of the form $\{S':p\}$. When a regular unit clause $\{S:p\}$ is deduced, and before inserting $S:p$ in the unit-clause literal list, we check if the intersection of S and the sign of the literal, in the unit-clause literal list, of opposite polarity with smallest cardinality is empty. As we have direct access to the latter literal and to p, the test $\Box \in \Gamma$ needs constant time.

The election of the next literal to which the branching rule is applied with heuristic RH-1 (RH-2) is done in such a way that every regular literal occurring in Γ is visited exactly once. Thus, the worst-case time complexity of heuristic RH-1 (RH-2) is in $\mathcal{O}(|\Gamma|)$.

The worst-case time complexity of applying the regular one-literal rule to a regular CNF formula Γ using a regular unit clause $\{S:p\}$ is in $\mathcal{O}(|\Gamma|)$:

- Removing regular clauses of the form $\{\cdots, S':p, \cdots\}$, where $S \subseteq S'$, is proportional to the length of the clauses removed: direct access (i) to the variable p is provided by the array of pointers, (ii) from p to the literal occurrence lists of literals of the form $S':p$ is provided by the ordered sign lists, and (iii) to the clauses with literal $S':p$ is provided by literal occurrence lists; each literal in the clause, except for $S':p$, is unlinked from its literal occurrence list in constant time and the clause head is unlinked from the clause list in constant time; and S' is unlinked from the corresponding sign list in constant time.

- Removing regular literals of the form $S'' : p$, where $S'' \cap S = \emptyset$, is proportional to the number of occurrences of this literal: each occurrence of a literal $S'' : p$ is unlinked from its clause literal list in constant time and S'' is unlinked from the corresponding sign list in constant time and the counter of the clause is decremented in constant time since each literal has a pointer to its clause head. When removing regular clauses and literals, if both sign lists become empty we also unlink the variable from the variable list.

- Detecting regular unit clauses is done in constant time, since the clause head has a counter of the clause length.

Notice that when we apply the regular one-literal rule to a formula Γ, we modify the data structures of Γ for obtaining a formula Γ'. We maintain a stack where we insert the literals of the regular unit clauses used by the regular one-literal rule. This way, we have just one formula in memory and avoid the cost of copying Γ for obtaining Γ'. When the algorithm backtracks, it obtains Γ from Γ' undoing the operations in reverse order using the information in the mentioned stack and recovering the pointers unlinked during the creation of Γ'. Otherwise, the memory space required would be quadratic in the size of the formula in the worst case.

3.4 The Regular-GSAT architecture

In this section we give a detailed description of two local search algorithms —Regular-GSAT/Basic and Regular-GSAT/Tabu— which are based on the Regular-GSAT architecture. Regular-GSAT is an extension of the GSAT architecture (Selman et al., 1992) to the framework of regular CNF formulas.

Given a regular CNF formula Γ, Regular-GSAT starts with a randomly generated interpretation, and tries to reduce the number of unsatisfied clauses by iteratively choosing one propositional variable occurring in Γ and then changing its truth assignment. Such changes are repeated until either a satisfying interpretation is found or a pre-set maximum number of changes (MaxChanges) is reached. This process is repeated as needed, up to a pre-set maximum number of times (MaxTries).

```
procedure Regular-GSAT
Input: a regular CNF formula $\Gamma$, MaxChanges and MaxTries
Output: a satisfying interpretation of $\Gamma$, if found
 begin
  for $i := 1$ to MaxTries
    $I$ := a randomly generated interpretation for $\Gamma$;
    for $j := 1$ to MaxChanges
      if $I$ satisfies $\Gamma$ then return $I$;
      $S$ := select-Regular-GSAT( $\Gamma, I$ );
      $p'$ := pick_variable( $\{p \mid (p,k) \in S\}$ );
      $k'$ := pick_value( $\{k \mid (p',k) \in S\}$ );
      $I$ := $I$ with the truth assignment of $p'$ changed to $k'$;
    end for
  end for
  return "no satisfying interpretation found";
 end
```

Figure 3.4: The Regular-GSAT architecture

The pseudo-code of Regular-GSAT is shown in Figure 3.4. Function select-Regular-GSAT returns a subset of the set of all possible changes that can be applied to the current interpretation. This subset contains the *best* changes that can be applied in a given step; and only one of such changes is chosen and applied in each step. Every change is represented by a pair (p, k), where p is a propositional variable occurring in the input formula Γ and k is a truth value of the set N of truth values occurring in Γ. The meaning of a change (p, k) is: the current assignment of p is changed to k. Function pick-variable chooses, at random, one propositional variable and func-

```
function select-Regular-GSAT
Input: a regular CNF formula $\Gamma$ and an interpretation $I$
Output: a set $S$ of variable-value pairs
 begin
   $u' := max\ (\ \{decrease((p,k),\Gamma,I)\ \mid\ (p,k) \in changes(\Gamma,I)\}\ )$;
   $S := \{(p,k) \mid (p,k) \in changes(\Gamma,I)\ \ and\ decrease((p,k),\Gamma,I) = u'\}$;
   return $S$
 end
```

Figure 3.5: Function select-Regular-GSAT for Regular-GSAT/Basic

tion pick-value chooses, at random, one truth value. The difference between Regular-GSAT/Basic (cf. Section 3.4.1) and Regular-GSAT/Tabu (cf. Section 3.4.2) lies in the strategy followed to construct the set returned by select-Regular-GSAT.

3.4.1 Regular-GSAT/Basic

The algorithm Regular-GSAT/Basic uses the function select-Regular-GSAT whose pseudo-code is shown in Figure 3.5. Function $decrease((p,k),\Gamma,I)$ returns the difference between the total number of regular clauses not satisfied by I' and the total number of regular clauses not satisfied by I, where I' is the interpretation that is identical to I except that the truth assignment of p is changed to k; and $changes(\Gamma,I)$ denotes the set of all possible changes that can be applied to the current interpretation; i.e., the set of the pairs (p,k) such that (i) p is a propositional variable occurring in Γ, and (ii) k is a truth value of the set N of truth values occurring in Γ such that $I(p) \neq k$. Thus, select-Regular-GSAT returns the subset of $changes(\Gamma,I)$ formed by those changes that produce the maximum decrease in the total number of unsatisfied clauses.

Observe that the maximum decrease may be zero, or even negative, and therefore the algorithm can get stuck in local minima. To cope with this problem, the most basic strategy is to restart the algorithm after a fixed number of changes if the algorithm does not find a solution. This is the strategy used by Regular-GSAT/Basic.

3.4.2 Regular-GSAT/Tabu

Regular-GSAT/Tabu is an extension of Regular-GSAT/Basic which incorparates a tabu search mechanism (Glover, 1989; Glover and Laguna, 1997) for preventing the algorithm to get stuck in local minima. Regular-GSAT/Tabu maintains a list of a fixed size t, called tabu list, that contains the last t propositional variables whose assignment has been modified. The way of preventing the algorithm to get stuck in local minima is to forbid changing the assignment of the variables of the tabu list.

Function select-Regular-GSAT for Regular-GSAT/Tabu is the same as in Figure 3.5, but now $changes(\Gamma, I)$ denotes the set of the pairs (p, k) such that (i) p is a propositional variable occurring in Γ which is not in the tabu list, and (ii) k is a truth value of the set N of truth values occurring in Γ such that $I(p) \neq k$.

3.5 The Regular-WalkSAT architecture

In this section we give a detailed description of a family of local search algorithms —Regular-WalkSAT/Basic, Regular-WalkSAT/G, Regular-WalkSAT/G+Tabu, Regular-WalkSAT/Novelty, and Regular-WalkSAT/R-Novelty— which are based on the Regular-WalkSAT architecture. Regular-WalkSAT is an extension of the WalkSAT architecture (Selman et al., 1994) to the framework of regular CNF formulas.

Given a regular CNF formula Γ, Regular-WalkSAT starts with a randomly generated interpretation, and tries to find a satisfying interpretation by iteratively choosing, at random, one clause C of Γ not satisfied by the

```
procedure Regular-WalkSAT
Input: a regular CNF formula $\Gamma$, MaxChanges, MaxTries, and $\omega \in [0,1]$
Output: a satisfying interpretation of $\Gamma$, if found
begin
  for $i := 1$ to MaxTries
    $I$ := a randomly generated interpretation for $\Gamma$;
    for $j := 1$ to MaxChanges
      if $I$ satisfies $\Gamma$ then return $I$;
      Pick one unsatisfied clause $C$ from $\Gamma$;
      $S$ := select-Regular-WalkSAT( $\Gamma$, $C$, $I$, $\omega$ );
      $p'$ := pick_variable( $\{p \mid (p,k) \in S\}$ );
      $k'$ := pick_value( $\{k \mid (p',k) \in S\}$ );
      $I$ := $I$ with the truth assignment of $p'$ changed to $k'$;
    end for
  end for
  return "no satisfying interpretation found";
end
```

Figure 3.6: The Regular-WalkSAT architecture

current interpretation and then changing the truth assignment of one propositional variable occurring in C. Such changes are repeated until either a satisfying interpretation is found or a pre-set maximum number of changes (MaxChanges) is reached. This process is repeated as needed, up to a pre-set maximum number of times (MaxTries).

The pseudo-code of Regular-WalkSAT is shown in Figure 3.6. Function select-Regular-WalkSAT returns the set of *best* changes that can be applied to the unsatisfied clause C chosen by Regular-WalkSAT; this set is a subset of the set of all changes that let C become satisfied by the new interpretation. Function pick-variable chooses, at random, one propositional variable and function pick-value chooses, at random, one truth value. The parameter

```
function select-Regular-WalkSAT
Input: a regular CNF formula $\Gamma$, a clause $C$, an interpretation $I$, and $\omega \in [0,1]$
Output: a set $S$ of variable-value pairs
begin
  $u := min\,(\ \{broken((p,k),\Gamma,I)\ \mid\ (p,k) \in changes(C,I)\}\ )$;
  if $(u > 0)$ then
    with probability $\omega$  return $changes(C,I)$;
  end if
  $S := \{\ (p,k) \mid broken((p,k),\Gamma,I) = u\ and\ (p,k) \in changes(C,I)\}$;
  return $S$;
end
```

Figure 3.7: Function select-Regular-WalkSAT for Regular-WalkSAT/Basic

$\omega \in [0,1]$ is a noise parameter whose aim is to introduce noise in a controlled way to escape from local minima. This means that, with certain probability, the change applied by the algorithm is not necessarily one of the apparently *best* changes. There is enough experimental evidence that these *random walks* accelerate the search for a solution. The difference among the algorithms based on the Regular-WalkSAT architecture lies in the strategy followed to construct the set returned by select-Regular-WalkSAT.

3.5.1 Regular-WalkSAT/Basic

The algorithm Regular-WalkSAT/Basic uses the function select-Regular-WalkSAT whose pseudo-code is shown in Figure 3.7. Function $broken((p,k),\Gamma,I)$ returns the number of broken clauses; i.e. the number of clauses of Γ that are satisfied by I but that would become unsatisfied if the assignment of p is changed to k; and $changes(C,I)$ denotes the set of variable-value pairs (p,k) such that (i) p is a propositional variable occurring in clause C, and (ii) C is satisfied by the interpretation I' which

```
function select-Regular-WalkSAT
Input: a regular CNF formula $\Gamma$, a clause $C$, an interpretation $I$, and $\omega \in [0,1]$
Output: a set $S$ of pairs
 begin
   with probability $\omega$ return $changes(C, I)$;
   $u' := max\ (\ \{decrease((p,k),\Gamma) \mid (p,k) \in changes(C,I)\}\ )$;
   $S := \{\ (p,k) \mid decrease((p,k),\Gamma,I) = u'\ and\ (p,k) \in changes(C,I)\}$;
   return $S$;
 end
```

Figure 3.8: Function select-Regular-WalkSAT for Regular-WalkSAT/G

is identical to I except that the truth assignment of p is changed to k; i.e., $changes(C, I)$ denotes the set of all changes that let C become satisfied by the new interpretation. If the minimum number of broken clauses in $changes(C, I)$, say u, is greater than zero, then (i) with probability ω, select-Regular-WalkSAT returns $changes(C, I)$, and (ii) with probability $1 - \omega$, select-Regular-WalkSAT returns the subset of $changes(C, I)$ formed by the changes for which the number of broken clauses is u. If $u = 0$, then select-Regular-WalkSAT returns the set of changes for which $u = 0$. Observe that while the GSAT/Basic objective function counts the number of unsatisfied clauses, the Regular-WalkSAT/Basic objective function counts the number of broken clauses.

3.5.2 Regular-WalkSAT/G

The algorithm Regular-WalkSAT/G, which is a generalization of the algorithm WalkSAT/G for *SAT* (McAllester et al., 1997), uses the function select-Regular-WalkSAT whose pseudo-code is shown in Figure 3.8. With probability ω, select-Regular-WalkSAT returns $changes(C, I)$, and with probability $1 - \omega$, select-Regular-WalkSAT returns the subset of *changes*(C, I) formed by the changes (p, k) that produce the maximum decrease in the total number of unsatisfied clauses when the truth value of p is changed to k.

3.5.3 Regular-WalkSAT/G+Tabu

The algorithm Regular-WalkSAT/G+Tabu is an extension of Regular-WalkSAT/G which incorporates tabu search for preventing the algorithm to get stuck in local minima by avoiding to repeat earlier changes. Regular-WalkSAT/G+Tabu maintains a list of a fixed size t, called tabu list, that contains the last t changes performed by the algorithm. With probability ω, select-Regular-WalkSAT returns *changes*(C, I), and with probability $1 - \omega$, select-Regular-WalkSAT returns the subset of *changes*(C, I) formed by the changes (p, k) that are not in the tabu list and produce the maximum decrease in the total number of unsatisfied clauses when the truth value of p is changed to k. If all the pairs in *changes*(C, I) are tabu, select-Regular-WalkSAT returns the empty set indicating to the algorithm that a new unsatisfied clause must be chosen instead. If all the possible changes in all the unsatisfied clauses are tabu, then the tabu list is ignored. In contrast to the other algorithms with a tabu list such as Regular-GSAT/Tabu and WalkSAT/Tabu (Selman et al., 1994), the tabu list of Regular-WalkSAT/G+Tabu contains a list of changes instead of a list of propositional variables.

3.5.4 Regular-WalkSAT/Novelty

The algorithm Regular-WalkSAT/Novelty is an extension of the algorithm WalkSAT/Novelty for solving *SAT* (McAllester et al., 1997). Function select-Regular-WalkSAT returns a singleton, i.e., a set containing only one change. For selecting this change, it first sorts the changes of *changes*(C, I) by the decrease in the number of unsatisfied clauses that such changes produce. When more than one change produces the same decrease, ties are broken in favor of the least recently applied change. Then, it considers the best and second best changes under this sort. If the best change is not the most recently applied to the clause under consideration, it selects the best pair. Otherwise, it either selects, with probability ω, the second best pair, or it selects, with probability $1 - \omega$, the best pair.

By taking into consideration the last time a particular change was applied, function select-Regular-WalkSAT tries to avoid to select a same pair more than once within a short period of time.

3.5.5 Regular-WalkSAT/R-Novelty

The algorithm Regular-WalkSAT/R-Novelty is an extension of the algorithm WalkSAT/R-Novelty for solving *SAT* (McAllester et al., 1997). Function select-Regular-WalkSAT also returns a singleton and sorts the changes of $changes(C, I)$ by the decrease in the number of unsatisfied clauses that such changes produce. Let (p_i, k_i) be the best change, let (p_j, k_j) be the second best change, and let n be the difference between the number of unsatisfied clauses produced by (p_i, k_i) and the number of unsatisfied clauses produced by (p_j, k_j). Then, if the best change is not the most recently applied to the clause under consideration, select-Regular-WalkSAT selects (p_i, k_i). Otherwise, there are four cases:

1. When $\omega < 0.5$ and $n > 1$, it selects (p_i, k_i).
2. When $\omega < 0.5$ and $n = 1$, then it either selects, with probability 2ω, (p_j, k_j), or it selects, with probability $1 - 2\omega$, (p_i, k_i).
3. When $\omega \geq 0.5$ and $n = 1$, it selects (p_j, k_j).
4. When $\omega \geq 0.5$ and $n > 1$, then it either selects, with probability $2(\omega - 0.5)$, (p_j, k_j), or it selects, with probability $1 - 2(\omega - 0.5)$, (p_i, k_i).

Moreover, to inhibit loops, select-Regular-WalkSAT randomly selects one change from $changes(C, I)$ every 100 changes. In contrast to Regular-WalkSAT/Novelty, when choosing between the best and second best pair, select-Regular-WalkSAT takes into consideration both the noise parameter ω and the difference n.

3.6 Data structures for local search algorithms

One important point in the design and implementation of local search algorithms is the definition of suitable data structures that allow algorithms to select and apply changes as fast as possible. The selection of changes amounts to implement the selection functions described above (select-Regular-GSAT and select-Regular-WalkSAT). The application of changes amounts to modify the current interpretation and update, in the clauses that are affected by the new interpretation, the *literal counter*. This counter contains the number of literals of the clause that are satisfied by the new interpretation. We associate a literal counter with each clause.

To obtain a low time complexity for the selection of changes, we should avoid to calculate, in each step and for each change, the number of clauses that become unsatisfied by the interpretation resulting of applying the change (broken clauses) and the number of clauses that become satisfied (repaired clauses). To this end, we associate two counters with each change: the *break counter* and the *repair counter*. These counters are updated at the beginning of each try, and after the application of a particular change, only the break and repair counters affected by this change are updated.

3.6.1 Data structures for GSAT and WalkSAT

Before presenting in detail the data structures for Regular-GSAT and Regular-WalkSAT, we first describe the data structures used in the latest implementations of GSAT and WalkSAT, which are publicly available at the SATLIB. The description below was obtained by inspecting the code of GSAT (version 41) and WalkSAT (version 35).

In GSAT and WalkSAT the truth value set is $N = \{0, 1\}$ and, therefore, there are two possible changes —$(p, 0)$ and $(p, 1)$— for each propositional variable p. A break counter and a repair counter is associated with each change, and a literal counter is associated with each clause. Observe that the difference between the repair counter and the break counter gives the increment in the total number of clauses satisfied by the interpretation resulting of applying the change. In the following, we will refer to the interpretation resulting of applying a given change as the interpretation induced by that change.

In GSAT, an array containing the set of best changes is maintained, and this array is updated after each change application. In WalkSAT, there is no data structure for the best changes because we consider a different clause in each change. Thus, the worst-case time complexity of selecting a change is in $\mathcal{O}(1)$ for GSAT and is in $\mathcal{O}(m)$ for WalkSAT, where m is the maximum number of literals per clause.

When a change (p, k) is applied, the literal counter of the clauses that contain the variable p has to be modified. Given a change $(p, 0)$ $((p, 1))$, the literal counter of the clauses that contain the literal p $(\neg p)$ is decremented (incremented) and the literal counter of the clauses that contain the literal $\neg p$ (p) is incremented (decremented). Moreover, every time the literal counter of a clause is modified, there are four situations in which it is also necessary to update break and repair counters:

1. the literal counter goes from 0 to 1. We must decrement the repair counter of all the changes such that the interpretation induced by them satisfy the clause and increment the break counter of the change such that its induced interpretation unsatisfies the clause. If the change is of the form $(p, 0)$ $((p, 1))$, we have to increment the break counter of the change $(p, 1)$ $((p, 0))$.

clause	C_1	C_2	C_3	C_4
literal counter	0	1	2	1

change	$(p,0)$	$(p,1)$	$(q,0)$	$(q,1)$	$(r,0)$	$(r,1)$
repair counter	0	1	1	0	0	1
break counter	0	1	1	0	0	0

Figure 3.9: Literal, break and repair counters before applying the change

2. the literal counter goes from 1 to 2. We must decrement the break counter of the change such that its induced interpretation, previous to the application of the change (p, k), unsatisfied the clause.

3. the literal counter goes from 2 to 1. We must increment the break counter of the change such that its induced interpretation unsatisfies the clause. If the change is of the form $(p, 0)$ $((p, 1))$, we have to increment the break counter of the change $(p, 1)$ $((p, 0))$.

4. the literal counter goes from 1 to 0. We must increment the repair counter of all the changes such that the interpretation induced by them satisfy the clause and decrement the break counter of the change such that its induced interpretation, previous to the application of the change (p, k), unsatisfied the clause.

For each change $(p, 0)$ $((p, 1))$, we have a list containing the clauses with an occurrence of the literal $\neg p$ (p). This way, counters are efficiently updated. If clauses are non-tautological, they never occur more than once in the same list.

Example 3.3 *To illustrate how break and repair counters are updated during the application of changes, we present an example based on the satisfiable CNF formula* $\Gamma = \{C_1, C_2, C_3, C_4\}$, *where*

clause	C_1	C_2	C_3	C_4
literal counter	1	2	1	0

change	$(p, 0)$	$(p, 1)$	$(q, 0)$	$(q, 1)$	$(r, 0)$	$(r, 1)$
repair counter	1	0	1	0	0	0
break counter	1	0	1	0	0	0

Figure 3.10: Literal, break and repair counters after applying the change

$$C_1 = \{p, \neg q, r\} \quad C_2 = \{p, q, r\}$$
$$C_3 = \{\neg p, q\} \quad C_4 = \{\neg p, \neg q\}$$

Assume that the initial interpretation for Γ*, which is randomly generated by a local search algorithm, is:* $I(p) = 0$*,* $I(q) = 1$ *and* $I(r) = 0$*. The initial values of the literal, break and repair counters are shown in Figure 3.9.*

In case we apply the change $(p, 1)$*, we have to update the counters as follows:*

1. *The literal counter of* C_1 *goes from* 0 *to* 1*. Then, we decrement the repair counter of the changes* $(p, 1)$*,* $(q, 0)$ *and* $(r, 1)$ *and increment the break counter of the change* $(p, 0)$*.*

2. *The literal counter of* C_2 *goes from* 1 *to* 2*. Then, we decrement the break counter of the change* $(q, 0)$*.*

3. *The literal counter of* C_3 *goes from* 2 *to* 1*. Then, we increment the break counter of the change* $(q, 0)$*.*

4. *The literal counter of* C_4 *goes from* 1 *to* 0*. Then, we increment the repair counter of the changes* $(p, 0)$ *and* $(q, 0)$ *and decrement the break counter of the change* $(p, 1)$*.*

Figure 3.10 shows the values of the literal, break and repair counters after applying the change $(p, 1)$*.*

With the above data structures, the worst-case time complexity of applying a change in WalkSAT is in

$$\mathcal{O}(|\Gamma'|),$$

where Γ' is the subformula formed by all the clauses whose literal counters have been updated by WalkSAT.

The complexity of applying a change in GSAT is bigger than in WalkSAT. This is due to the fact that we have to insert and remove changes in the array that stores the best changes. As the time complexity of an insertion (deletion) is in $\mathcal{O}(1)$, the time complexity of updating the array usually is in $\mathcal{O}(|\Gamma'|)$. The exception is when the array contains only one change and its counters are modified in such a way that the change is no longer the unique best change. In this case, the array that stores the best changes must be rebuilt from scratch. The cost of rebuilding the array is in $\mathcal{O}(|V|)$, where V is the set of propositional variables occurring in the formula. Thus, the worst-case time complexity of applying a change in GSAT is in

$$\mathcal{O}(|\Gamma'| + |V|).$$

3.6.2 Data structures for Regular-GSAT and Regular-WalkSAT

The complexity of selecting and applying changes in local search algorithms for *Regular-SAT* is bigger due to the following facts: (i) the same propositional variable can occur more than once in a (non-tautological) regular clause, and (ii) a regular literal $S : p$ is associated with as many changes as truth values appear in S.

To reduce the time complexity, there is a preprocessing phase in which tautological clauses are eliminated, and all the occurrences of literals with the same propositional variable in a non-tautological clause are grouped together. Given a regular clause with several occurrences of literals with the same propositional variable, we can obtain a logically equivalent regular clause by eliminating all the positive (negative) literals with this variable except for the positive (negative) literal whose sign has the greatest cardinality; i.e., subsumed literals are eliminated. This way, we have at most one positive ($\uparrow i : p$) and one negative ($\downarrow j : p$) literal with the same propositional variable in each regular clause, which are stored in a common data structure that we call *metaliteral* and that we denote by $\{\downarrow i, \uparrow j\} : p$. The metaliteral $\{\emptyset, \uparrow j\} : p$ ($\{\downarrow i, \emptyset\} : p$) is used when there are only occurrences of positive (negative) literals. Finally, we maintain, for each metaliteral, a list that contains the clauses in which this metaliteral appears. For every different variable p, the lists of clauses in which there are metaliterals with the variable p are accessed through a two-dimensional array such that the list for the metaliteral $\{\downarrow i, \uparrow j\} : p$ is found at row i and column j.

To select changes in Regular-GSAT, fast access to the best changes is provided. To this end, an array containing the propositional variables occurring in the best changes is maintained and, for each propositional variable, an array with its best truth values is maintained. With these data structures we have that the time complexity of selecting a change in Regular-GSAT is in $\mathcal{O}(1)$.

The complexity of applying changes in Regular-GSAT is bigger than in GSAT. This is due to the fact that the cardinality of the truth value set is greater than 2. Every time the assignment of a propositional variable p is changed from a value k to a new value k', Regular-GSAT considers both the metaliterals that become satisfied and the metaliterals that become unsatisfied. The metaliterals $\{\downarrow i, \uparrow j\} : p$ that become unsatisfied are those metaliterals such that either k is in $\downarrow i$ or in $\uparrow j$ but k' is not in any of them, and the metaliterals that become satisfied are those metaliterals such that k is not in any of the two signs but k' is in one of them. Observe that a metaliteral $\{\downarrow i, \uparrow j\} : p$ such that k is in $\downarrow i$ and k' is in $\uparrow j$, or k is in $\uparrow j$ and k' is in $\downarrow i$, remains satisfied and so the literal counter of clauses containing that metaliteral is not modified.

The literal counter of the clauses that contain a metaliteral that has become satisfied is incremented, and the literal counter of the clauses that contain a metaliteral that has become unsatisfied is decremented. As clauses contain at most one metaliteral with the same propositional variable, literal counters of clauses are updated at most once during the application of a given change.

Every time the literal counter of a regular clause is modified, it is necessary to update the break and repair counters like in GSAT (cf. page 51). In Regular-GSAT, the difference is that the number of changes whose break and repair counters are affected by the modification of a given literal counter is bigger than in GSAT. This is due to the fact that the number of changes whose induced interpretations satisfy or unsatisfy a given metaliteral depends on $|N|$. Moreover, the array used to store the variables occurring in the best changes must be also updated every time the counters of a change are modified. The time complexity of updating the counters of all the changes affected by the increase or decrease of a given literal counter is in $\mathcal{O}(m \cdot |N|)$, where m is the maximum number of metaliterals per clause. The time complexity of inserting or deleting variables to/from the array that stores the variables occurring in the best changes is in $\mathcal{O}(1)$. Sometimes, like in GSAT, this array must be rebuilt from scratch after applying a change with a time complexity which is in $\mathcal{O}(|V|)$. Thus, we have that the worst-case time complexity of applying a change in Regular-GSAT is in

$$\mathcal{O}(|\Gamma'| \cdot |N| + |V|),$$

where Γ' is the subformula formed by all the clauses whose literal counters have been updated by Regular-GSAT. Observe that we have direct access to the clauses of Γ' thanks to the lists of metaliterals.

Regular-WalkSAT selects and applies changes in a very different way. In contrast to Regular-GSAT, it does not maintain data structures for providing fast access to the best changes because only the best changes of one clause are considered. Another difference is the way of updating the break and repair counters of changes.

As before, given a change (p, k) we have to update the literal counter of some clauses and the break and repair counters of some changes. First, the literal counter of the clauses that contain a metaliteral that has become satisfied is incremented, and the literal counter of the clauses that contain a metaliteral that has become unsatisfied is decremented. Second, we should update the break and repair counters of the changes like in GSAT (cf. page 51).

Updating the counters of all the changes affected by the increase or decrease of a given literal counter can be done in time $\mathcal{O}(m \cdot |N|)$, where m is the maximum number of metaliterals per clause. Actually, in order to obtain a time complexity in $\mathcal{O}(m)$, we delay updating the break and repair counters until we really need to know the values of such counters. To this end, we associate two auxiliary counters with the repair counter of each change and two auxiliary counters with the break counter. Given a change (p', k'), one of the auxiliary counters associated with the repair counter indicates the increment or decrement that has to be applied to the repair counter of all the changes (p', k_1) such that $k' \leq k_1$, and the other auxiliary counter indicates the increment or decrement that has to be applied to the repair counter of all the changes (p', k_2) such that $k_2 \leq k'$. The two auxiliary counters associated with the break counter are used in a similar way. Regular-WalkSAT updates the break and repair counters of all the changes associated with the variable p' when p' appears in the clause selected by select-Regular-WalkSAT. Updating the break and repair counters of these changes by using the auxiliary counters is in $\mathcal{O}(|N|)$. Therefore, the worst-case time complexity of selecting a change in Regular-WalkSAT is in $\mathcal{O}(m \cdot |N|)$, and the worst-case time complexity of applying a change is in $\mathcal{O}(|\Gamma'|)$, where Γ' is the subformula formed by all the clauses whose literal counters have been updated by Regular-WalkSAT.

Table 3.1 shows the worst-case time complexity of selecting and applying one change in the GSAT, Regular-GSAT, WalkSAT and Regular-WalkSAT families. Observe that there is a trade-off between the complexity of selecting a change and the complexity of applying it. In GSAT and Regular-GSAT, the fact of maintaining the set of best changes is performed at the cost of increasing the complexity of applying changes. By contrast, the complexity of selecting changes in WalkSAT and Regular-WalkSAT is greater than in GSAT and Regular-GSAT, but the complexity of applying changes is smaller.

	classical		**regular**	
	selecting	applying	selecting	applying
GSAT	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\Gamma'\rvert + \lvert V\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\Gamma'\rvert \cdot \lvert N\rvert + \lvert V\rvert)$
WalkSAT	$\mathcal{O}(m)$	$\mathcal{O}(\lvert\Gamma'\rvert)$	$\mathcal{O}(m \cdot \lvert N\rvert)$	$\mathcal{O}(\lvert\Gamma'\rvert)$

Table 3.1: The complexity of selecting and applying changes in local search algorithms for *SAT* and *Regular-SAT*

3.7 Probabilistically approximately complete algorithms

In this section we first define probabilistically approximately complete algorithm (Hoos, 1999a), and then analyse this property for the regular local search algorithms described in this thesis.

Definition 3.1 *Let* Π *be a decision problem. Let* A *be a randomized Las Vegas algorithm for* Π*; i.e.,* A*'s output is always correct and* A*'s run time is a random variable. For a given soluble problem instance* $\pi \in \Pi$*, let* $P_s(RT_{A,\pi} \leq t)$ *denote the probability that* A *finds a solution for* π *in time less than or equal to* t*, and let* $P \subseteq \Pi$*.* A *is probabilistically approximately complete (PAC) for* P *if* $lim_{t\rightarrow\infty}[P_s(RT_{A,\pi} \leq t)] = 1$ *for all soluble instances* $\pi \in P$*.* A *is essentially incomplete for* P *if it is not PAC for* P*; i.e., there is a soluble instance* $\pi \in P$ *for wich* $lim_{t\rightarrow\infty}[P_s(RT_{A,\pi} \leq t)] < 1$*.*

This property has been shown to be important for local search algorithms for SAT (Hoos, 1998; Hoos, 1999a; Hoos and Stützle, 2000a). A local search algorithm with this property is more robust with respect to the setting of the MaxChanges parameter; i.e., the algorithm is always able to find a solution in only one try when the setting of the MaxChanges parameter is infinite.

A sufficient condition for having the PAC property a local search algorithm of the Regular-GSAT and Regular-WalkSAT families is that in every change the algorithm performs a random step with a probability greater than zero. By a random step we mean, for the algorithms of the Regular-GSAT family, to randomly select a change from the set $changes(\Gamma, I)$ and, for the algorithms of the Regular-WalkSAT family, to randomly select a change from $changes(C, I)$. We assume that the probability of choosing one arbitrary change from $changes(\Gamma, I)$ and $changes(C, I)$ is greater than zero. The proof that this is a sufficient condition is the same as the proof in (Hoos, 1999a) for classical local search algorithms. It relies on the fact that, in each change, the algorithm decreases the Hamming distance between the current interpretation and a satisfying interpretation with a probability greater than zero. The Hamming distance decreases if the algorithm can perform arbitrary long sequences of random steps and there is at least one change in $changes(\Gamma, I)$ (or in $changes(C, I)$) that decreases the Hamming distance by one.

The algorithms Regular-WalkSAT/G and Regular-WalkSAT/G+Tabu are PAC because they satisfy the above condition. On the other hand, the algorithms Regular-GSAT/Basic, Regular-WalkSAT/Novelty and Regular-WalkSAT/R-Novelty, like their classical counterparts (Hoos, 1998), are essentially incomplete.

It is an open question to know whether Regular-WalkSAT/Basic and Regular-GSAT/Tabu are PAC. The experimental investigation reported in (Hoos, 1998) suggests that WalkSAT might be PAC. The experimental results described in Chapter 5 indicate that Regular-WalkSAT/Basic is PAC for a particular class of problem instances. Observe that Regular-WalkSAT/Basic does not satisfy the random step condition, because this algorithm does not perform a random step with probability greater than zero in each step; it only considers random steps when all the changes in $changes(C, I)$ break clauses.