Journal on Satisfiability, Boolean Modeling and Computation 1 (2007) 169–186

Using SAT Encodings to Derive CSP Value Ordering Heuristics

Christophe Lecoutre lecoutre@cril.fr
Lakhdar Saïs sais@cril.fr
Julien Vion vion@cril.fr
CRIL-CNRS FRE 2499, Université d'Artois
Lens, France

Abstract

In this paper, we address the issue of designing from SAT new value ordering heuristics for CSP. We show that using the direct and support SAT encodings of CSP instances, such heuristics can be naturally derived from the well-known two-sided *Jeroslow-Wang* heuristic. These heuristics exploit the bi-directionality of constraint supports to give a more comprehensive picture in terms of domain reduction when a given value is assigned to (resp. removed from) a given variable. Interestingly, in the context of a backtracking search algorithm that exploits binary branching and the adaptive variable ordering heuristic *dom/wdeg*, we experimentally observed that the new heuristics yielded the best results on satisfiable and unsatisfiable instances when following the *promise* and the *fail-first* policies, respectively.

KEYWORDS: *heuristics, SAT encodings, CSP*

Submitted November 2006; revised April 2007; published May 2007

1. Introduction

For solving instances of the Constraint Satisfaction Problem (CSP), backtracking search algorithms are commonly used. To limit their combinatorial explosion, various improvements have been proposed (e.g. ordering heuristics, filtering techniques and conflict analysis). It is well known that the ordering used to perform search decisions has a great impact on the size of the search tree. At each stage, one needs to decide the *value* to assign to a *variable*. So far, such decisions have been performed by choosing the variable in a first step (vertical selection) and the value to assign in a second step (horizontal selection).

Many works have been devoted to the first selection step. Variable ordering heuristics that have been proposed can be conveniently classified as static (e.g. *deg*), dynamic (e.g. *dom* [15], *brelaz* [6], *dom/ddeg* [4]) and adaptive (e.g. *dom/wdeg* [5]). The heuristic *dom/wdeg* has been shown to be the most robust generic heuristic [5, 20, 16, 28]. However, value ordering (the second step of the decision) has clearly been considered for a long time as potentially of marginal effect to search improvements. The arguments behind this can be related to the fact that selecting a given value is computationally more difficult than selecting a given variable, particularly when one considers dynamic selection. The second reason for considering value ordering as useless is that, when facing unsatisfiable instances or when searching all solutions, one needs to consider all values for each variable. As clearly

shown by Smith and Sturdy [25], these arguments hold when search is based on d-way branching but not on 2-way branching. In fact, d-way branching means that, at each node of the search tree, a variable X is selected and d branches are considered where d is the current size of the domain of X: the i^{th} branch corresponds to $X = v_i$ where v_i denotes the i^{th} value of the domain of X. On the other hand, with binary (or 2-way) branching, at each node of the search tree, a pair (X, a) is selected where X is an unassigned variable and a a value in the domain of X, and two branches are considered: the first one corresponds to the assignment $X = a$ and the second one to the refutation $X \neq a$. These two schemes are not equivalent as it has been shown that binary branching is more powerful than non-binary branching [17].

Traditionally, two principles are considered during search: at each step, select the variable which is the most constrained and select then the least constrained value (e.g. *min-conflicts* [12]). These principles respectively correspond to two policies called $fail\text{-}first$ and *promise*, and one interesting issue is the adherence assessment of heuristics to both policies [2, 30].

Considering the fact that, for some types of constraints, good value ordering can significantly reduce the search effort [25], we decided to further investigate value ordering heuristics (assuming, of course, an underlying 2-way branching scheme). In particular, our attention was attracted by the fact that 2-way branching is the basic scheme in SAT solvers. We thought that this might be very helpful to map SAT heuristics to CSP ones. Indeed, considering any SAT encoding of a CSP instance, selecting a pair composed of a variable and a value corresponds to the selection of a literal in SAT.

In this paper, we show a direct correspondence between *min-conflicts* (resp. *max-conflicts*) and the maximum number of literal occurrences in the SAT formula obtained using support (resp. direct) encoding of CSP instances. Also, we propose new value ordering heuristics derived from the well known *Jeroslow-Wang (JW)* heuristic [18]. The obtained heuristics exploit the bi-directionality of constraints to give a more comprehensive picture in terms of domain reduction when a given value is assigned to a given variable and also when a given value is removed from the domain of a given variable. Let us illustrate this with the following example.

Example 1. *Let C be the binary constraint depicted by Figure 1. Note that any value in the domain of X occurs in two allowed tuples and is in conflict with two values in the domain of Y. Consequently, applying a classical value ordering heuristic such as min-conflicts does not discriminate between the different values of X since all the values of X have the same number of supports in Y.*

This example shows that it is not always sufficient to only consider the number of conflicts in order to choose the most (or least) promising value. Indeed, considering a binary branching scheme, when the value a is assigned to X (decision corresponding to the first branch), two values are removed from $dom(Y)$, and when a is removed from $dom(X)$ (decision corresponding to the second branch) two values are also removed from $dom(Y)$. On the other hand, when the value b or c is assigned to X, two values are removed from $dom(Y)$, and when b or c is removed from $dom(X)$ no value is removed from $dom(Y)$. So, the value a is more constrained than b or c. This illustration shows that it can be

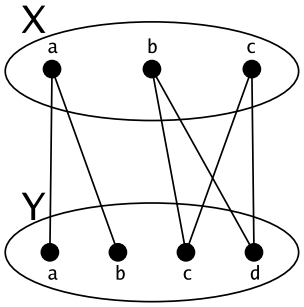

Figure 1. A constraint C between X and Y. Edges correspond to allowed pairs of values.

important to consider the impact on both branches when evaluating values to be selected by an heuristic.

In fact, the estimation of the number of removed values when eliminating a given value from the domain of a given variable (the refutation labelling the second branch of a binary search) has not been considered so far when devising generic value ordering heuristics. Interestingly enough, our approach can be used to derive in more general way a suitable value ordering with respect to any type of constraint. It is then related to a recent independent work by Szymanek and O'Sullivan [26].

The rest of the paper is organized as follows. After some technical background about CSP and SAT, SAT encodings of CSP instances are recalled. Our approach is then presented. Experimental results conducted on a wide range of CSP instances are described and discussed before concluding.

2. Technical Background

2.1 Constraint Satisfaction Problem

A Constraint Network (CN) P is a pair $(\mathscr{X},\mathscr{C})$ where $\mathscr{X}$ is a finite set of variables and $\mathscr{C}$ a finite set of constraints. Each variable $X \in \mathscr{X}$ has an associated domain, denoted $dom(X)$, which represents the set of values allowed for X. Each constraint $C \in \mathscr{C}$ involves a subset of variables of $\mathscr{X}$, called the scope and denoted $vars(C)$, and has an associated relation, denoted $rel(C)$, which contains the set of tuples allowed for the variables of its scope. $\Gamma(X)$ denotes the set of constraints involving X. From now on, to simplify and without any loss of generality, we will only consider binary networks, i.e. networks involving binary constraints. The number of variables of a CN will be denoted by n, the number of constraints by e and the greatest domain size by d. Also, for any given set E, $|E|$ will denote the number of elements in E. Note that in the case of binary networks, $\Gamma(X)$ is equal to the number of neighbors of X.

A solution to a constraint network is an assignment of values to all the variables such that all the constraints are satisfied. A constraint network is said to be satisfiable iff it admits at least one solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given constraint network is satisfiable. A CSP instance is then defined by a constraint network, and solving it involves either finding one

Algorithm 1: MAC(P = $(\mathscr{X},\mathscr{C})$: CN): Boolean

```
1 if 𝒳 = ∅ then return true
2 select (X, v) | X ∈ 𝒳 ∧ a ∈ dom(X)
3 P' ← AC(P|_{X=a})
4 if P' ≠ ⊥ ∧ MAC(P'\X) then return true
5 P' ← AC(P|_{X≠a})
6 return P' ≠ ⊥ ∧ MAC(P')
```

(or more) solution(s) or determining its unsatisfiability. To solve a CSP instance, one can modify the constraint network by using inference or search methods [8]. Usually, domains of variables are reduced by removing inconsistent values, i.e. values that can not occur in any solution. Indeed, it is possible to filter domains by considering some properties of constraint networks. Arc Consistency (AC), which remains the central one, guarantees the existence of a support for each value in each constraint.

Definition 1. *Let $P = (\mathscr{X},\mathscr{C})$ be a CN, $C \in \mathscr{C}$ s.t. $vars(C) = \{X, Y\}$ and $a \in dom(X)$.*

- *The set of supports of (X, a) in C, denoted $\mathcal{S}_p(C, X, a)$, corresponds to the set $\{b \in dom(Y) \mid (a, b) \in rel(C)\}$.*
- *The set of conflicts of (X, a) in C, denoted $\mathcal{C}_f(C, X, a)$, corresponds to the the set $\{b \in dom(Y) \mid (a, b) \notin rel(C)\}$.*

In Example 1, we have $\mathcal{S}_p(C, X, a) = \mathcal{S}_p(C, X, b) = \{(Y, a), (Y, b)\}$, $\mathcal{S}_p(C, X, c) = \{(Y, c), (Y, d)\}$, $\mathcal{C}_f(C, X, a) = \mathcal{C}_f(C, X, b) = \{(Y, c), (Y, d)\}$ and $\mathcal{C}_f(C, X, c) = \{(Y, a), (Y, b)\}$.

Definition 2. *Let $P = (\mathscr{X},\mathscr{C})$ be a CN. A pair (X, a), with $X \in \mathscr{X}$ and $a \in dom(X)$, is arc consistent (AC) iff $\forall C \in \mathscr{C} | X \in vars(C)$, $\mathcal{S}_p(C, X, a) \neq \emptyset$. P is AC iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, (X, a) is AC.*

Notation. *$AC(P)$ denotes the constraint network obtained from P after enforcing arc consistency on P. When P cannot be made arc consistent, we will note $AC(P) = \bot$.*

Let us now briefly describe the well known MAC algorithm [24]. This algorithm aims at solving a CSP instance and performs a depth-first search with backtracking while maintaining arc consistency. More precisely, at each step of the search, a variable assignment is performed followed by a filtering process called constraint propagation which corresponds to enforcing arc-consistency. Algorithm 1 corresponds to a recursive version of the MAC algorithm (using binary branching). It returns *true* iff the given constraint network P is satisfiable. It involves selecting a pair (X, a) and trying first $X = a$ and then $X \neq a$ (if no solution has been found with $X = a$). After any consistent assignment, the assigned variable is eliminated from the network and search is continued (line 4)[1.]. When the current constraint network has no more variables (line 1), it means that a solution has been found.

1. $P|_{X=a}$ denotes the constraint network obtained from P by restricting the domain of X to the singleton $\{a\}$, whereas $P|_{X\neq a}$ denotes the constraint network obtained from P by removing the value a from the domain of X. $P\backslash X$ denotes the constraint network obtained from P by removing the variable X.

Instead of directly selecting a pair (X, a), most CSP solvers select first a variable and then a value belonging to the domain of this variable. It can be partly explained by the fact that the number of elements to consider for the selection is then $O(n+d)$ instead of $O(nd)$. Classically, CSP solvers use $fail\text{-}first$ oriented variable ordering heuristics such as *dom* [15], *brelaz* [6] or *dom/wdeg* [5]. Value ordering is usually considered as less important, this is why lexicographic (*lexico*) value ordering is still the most used one. Let us note that when the values of the domains are shuffled, *lexico* and *random* value selection are equivalent. More informed value heuristics exploit the set of allowed tuples $rel(C)$ associated with each constraint C. This leads to the well known heuristic called *min-conflicts*, which involve selecting the value with the lowest number of conflicts (i.e., disallowed tuples) [16, 13, 12]. $min-conflicts$ (respectively $max-conflicts$) follows the *promise* (respectively $fail\text{-}first$) policy.

One advantage of using *random* or *lexico* is that selecting a value is $O(1)$ whereas it is usually $O(d\eta)$ for the other heuristics with η denoting the complexity of evaluating a value according to the heuristic. In fact, this is true when the heuristic is dynamic. In practice, it is usually far more efficient to consider a static version of the heuristics. It means that the order of values is computed for each domain in a preprocessing step. It is even possible to rearrange the values in their respective domains prior to search in order to be able to select a value in $O(1)$ [23]. From now on, we will consider all value ordering heuristics as being static.

2.2 Encoding CSP into SAT

Propositional satisfiability (SAT) is the problem of deciding whether a Boolean formula in conjunctive normal form (CNF) is satisfiable. A *CNF formula* Σ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as a disjunction) of *literals*. A literal is a positive or negated propositional variable. A *truth assignment* of a boolean formula is an assignment of truth values $\{true, false\}$ to its variables. A *model* of a formula is a truth assignment that satisfies the formula. SAT is one of the most studied NP-Complete problems because of its theoretical and practical importance. Encouraged by the impressive progress in practical solving of SAT, various applications ranging from formal verification to planning are encoded and solved using SAT. CSP instances can also be reformulated as SAT instances.

In this paper, we consider the most commonly used encodings of CSP into SAT, namely, the direct encoding [7] and the support encoding [14]. In both SAT encodings of a constraint network $P = (\mathscr{X}, \mathscr{C})$, a propositional variable x_a is associated to each pair (X, a) of P with $X \in \mathscr{X}$ and $a \in dom(X)$. The correspondence is the following: x_a is true if X is assigned the value a (i.e. $X = a$) and x_a is false if a is removed from $dom(X)$ (i.e. $X \neq a$).

Before introducing direct and support encodings, we first present the two particular sets of clauses expressing that a given variable must be assigned to only one value. Given a variable $X \in \mathscr{X}$ with $dom(X) = \{v_1, v_2, \ldots, v_d\}$,

- one *at least one* clause expresses that the variable X must be assigned at least one value from its domain. Such constraint is encoded by the clause of the form $x_{v_1} \vee x_{v_2} \vee \cdots \vee x_{v_d}$. For a given constraint network P, $\Sigma^l(P)$ denotes the set of *at least one* clauses encoding all the domains of P.

- *at most one* clauses express that the variable X must be assigned at most one value from its domain. Such constraint is encoded by the following set of clauses:

  $\{(\neg x_{v_1} \vee \neg x_{v_2}), \ldots, (\neg x_{v_1} \vee \neg x_{v_d}), \ldots,$
  $(\neg x_{v_i} \vee \neg x_{v_{i+1}}), \ldots, (\neg x_{v_i} \vee \neg x_{v_d}), \ldots, (\neg x_{v_{d-1}} \vee \neg x_{v_d})\}$.

  For a given constraint network P, $\Sigma^m(P)$ denotes the set of *at most one* clauses encoding all the domains of P.

To simplify the presentation, for a given constraint network P, we define $\Sigma^{lm}(P) = \Sigma^l(P) \cup \Sigma^m(P)$. Let us note that direct and support encoding differ mainly in the way the constraint are encoded. Both encoding share the same set of clauses encoding the domains (*at least one* and *at most one* clauses).

2.2.1 Direct Encoding

The direct encoding of a constraint network $P = (\mathscr{X}, \mathscr{C})$ [7] involves an additional set of clauses, called *conflict* clauses, encoding $\mathscr{C}$. The conflict clauses encode for each constraint the incompatible tuples i.e. tuples not satisfying the constraint. Each constraint $C \in \mathscr{C}$ such that $vars(C) = \{X, Y\}$ is encoded as a set of *conflict* clauses:

$$\bigcup_{v \in dom(X), w \in \mathcal{C}_f(C,X,v)} \{(\neg x_v \vee \neg y_w)\}$$

From the constraint C given in example 1, we obtain the following set of *conflict* clauses:
$\{(\neg x_a \vee \neg y_c), (\neg x_a \vee \neg y_d), (\neg x_b \vee \neg y_a), (\neg x_b \vee \neg y_b), (\neg x_c \vee \neg y_a), (\neg x_c \vee \neg y_b)\}$

For a given constraint network P, $\Sigma^{\mathcal{C}_f}(P)$ denotes the set of *conflict* clauses encoding all the constraints of P. The direct encoding of P is then defined as a formula $\Sigma^D(P) = \Sigma^{lm}(P) \cup \Sigma^{\mathcal{C}_f}(P)$. The set of *at most one* clauses $\Sigma^m(P)$ is only necessary to guarantee the equivalence between P and $\Sigma^D(P)$, i.e. the same set of models. They can be omitted to obtain an equivalence w.r.t. SAT [31, 14].

2.2.2 Support Encoding

The idea of encoding supports has been first introduced by Kasif in [19] and expanded on by Gent [14]. The support encoding [7] of a constraint network $P = (\mathscr{X}, \mathscr{C})$ involves an additional set of clauses, called *support* clauses, encoding $\mathscr{C}$. Each constraint $C \in \mathscr{C}$ such that $vars(C) = \{X, Y\}$ is encoded using both supports $\mathcal{S}_p(C, X, v)$ for all values $v \in dom(X)$ and $\mathcal{S}_p(C, Y, w)$ for all values $w \in dom(Y)$. The set of *support* clauses encoding C is the union of the two following sets of clauses:

1. $\displaystyle\bigcup_{v_i \in dom(X)} \{(\neg x_{v_i} \vee y_{w_1} \vee \cdots \vee y_{w_k}) \mid \{w_1, \ldots, w_k\} = \mathcal{S}_p(C, X, v_i)\}$

2. $\displaystyle\bigcup_{w_i \in dom(Y)} \{(\neg y_{w_i} \vee x_{v_1} \vee \cdots \vee x_{v_k}) \mid \{v_1, \ldots, v_k\} = \mathcal{S}_p(C, Y, w_i)\}$

The following set of *support* clauses is obtained from Example 1 using the support encoding:

$\{(\neg x_a \vee y_a \vee y_b), (\neg x_b \vee y_c \vee y_d), (\neg x_c \vee y_c \vee y_d), (\neg y_a \vee x_a), (\neg y_b \vee x_a), (\neg y_c \vee x_b \vee x_c), (\neg y_d \vee x_b \vee x_c)\}$

For a given constraint network P, $\Sigma^{\mathcal{S}_p}(P)$ denotes the set of *support* clauses encoding all the constraints of P. The support encoding of P is then defined as a formula $\Sigma^{S}(P) = \Sigma^{lm}(P) \cup \Sigma^{\mathcal{S}_p}(P)$.

Support encoding admits interesting features. In [14], it is shown that encoding supports enables Arc Consistency in the original CSP instance to be established by unit propagation in the translated SAT instances. Last but not least, applying the well known DPLL algorithm using the same ordering to the obtained SAT instance behaves exactly like the MAC algorithm on the original CSP instance. We can also mention that support encoding has been extended to encode non-binary constraints in SAT [3]. Interestingly enough, it has been proved in [9] that support clauses can be inferred from direct encoding using HyperBin resolution introduced by Bacchus [1]. These nice results open new interesting perspectives for establishing strong connections between SAT and CSP. The results that we present below on value ordering can be seen as a step in this direction.

3. Value Ordering Heuristics from SAT to CSP

3.1 SAT Branching Heuristics

Many branching heuristics have been proposed in SAT. One can cite the most recent ones, namely the VSIDS and UP heuristics used in Zchaff [32] and Satz solvers [22] respectively. The first one uses literal occurrences in the set of learned no-goods with dynamic decay policies, whereas the second one measures the effect of unit propagation on the formula when a literal is assigned a truth value. Previously, CSAT [10] and POSIT [11], among other solvers, used simpler heuristics. Most of them are variants of the well-known Jeroslow-Wang (JW) heuristic [18], and evaluate a given literal according to syntactical properties (e.g. occurrence number of literals, clause length).

Let us explain the main idea behind the JW heuristic. Given a CNF formula Σ with n variables, Σ admits $p = 2^n$ interpretations. Each clause $c \in \Sigma$ removes $v = 2^{n-|c|}$ lines from the truth table where $|c|$ denotes the size of the clause. So, v expresses the number of interpretations that falsifies c. The proportion $v/p = w = 2^{-|c|}$, represents the proportion of the search space falsifying c. The smaller the clause size is, the higher the number of interpretations falsifying it. A clause of size one is falsified by 2^{n-1} interpretations, i.e. half of the search space.

In SAT branching heuristics, the score, denoted $H(\Sigma, x)$, of a variable x from a CNF Σ is generally defined as a function $f[h(x), h(\neg x)]$, called two-sided, of the score associated with its positive ($h(x)$) and negative literals ($h(\neg x)$). The next variable to assign is then chosen among variables with the greatest (max) or the lowest (min) score. Generally, SAT branching heuristics are formulated as follows:

Definition 3. *Let Σ be a propositional formula and x be a propositional variable from Σ, the score, denoted $H_w(\Sigma, x)$, of x in Σ w.r.t. a weighting function w is defined as follows:*

$$H_w(\Sigma, x) = \sum_{c \in \Sigma \mid x \in c} w(|c|) + \sum_{c \in \Sigma \mid \neg x \in c} w(|c|), \text{ with } c \in \Sigma$$

A SAT heuristic, denoted $H_w^{\otimes}$, considers all the variables $x \in vars(\Sigma)$ and selects according to the operator $\otimes$ the variable with the optimal value $H_w(\Sigma, x)$.

Considering a selection operator $\otimes$ and a weighting function w, different heuristics can be derived from the general formulation given in definition 3. For example, the two-sided Jeroslow-Wang (JW) rule corresponds to $H_w^{\otimes}$ where $w(\alpha) = 2^{-\alpha}$ and $\otimes = max$. Many variants of the JW heuristic have been proposed; all of them attempt to choose variables with Maximum Occurrences in clauses of Minimal Size (MOMS) [10]. Another basic heuristic that we will consider in this paper is $H_w^{\otimes}$ where $w(\alpha) = 1$ and $\otimes \in \{min, max\}$. This heuristic, with $\otimes = max$ (resp. $\otimes = min$), selects in priority a variable with the greatest (resp. lowest) number of occurrences in the formula.

3.2 Mapping SAT Heuristics to CSP

Using direct and support encodings, we present now the CSP value ordering heuristics obtained from the instantiation of SAT branching heuristic $H_w^{\otimes}$. We note $D\text{-}H_w$ (resp. $S\text{-}H_w$) the variable value function obtained using direct (resp. support) encoding.

3.2.1 Mapping $H_w^{\otimes}$ using Direct Encoding

Let us first show how the SAT branching heuristic on direct encoding can correspond to the CSP value ordering heuristic $max\text{-}conflicts$ or $min\text{-}conflicts$.

Definition 4. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, $X \in \mathscr{X}$ and $a \in dom(X)$, the score, denoted $D\text{-}H_w[P,(X,a)]$, of (X,a) in P is defined as follows:*

$$D\text{-}H_w[P,(X,a)] = W[dom(X)] + w(2) \times \sum_{C \in \mathscr{C} \mid X \in vars(C)} |\mathcal{C}_f(C,X,a)|$$

such that

$$W[dom(X)] = w(|dom(X)|) + w(2) \times (|dom(X)| - 1)$$

Property. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, $X \in \mathscr{X}$ and $a \in dom(X)$*
We have $D\text{-}H_w[P,(X,a)] = H_w[\Sigma^D(P), x_a]$

Proof. Each positive literal x_a appears exactly one time in *at least one* clauses. The size of the clause containing x_a corresponds to the size of the domain of the variable X. The negative literal $\neg x_a$ occurs in $(|dom(X)| - 1)$ *at most one* binary clauses. Hence, we obtain the term $W[dom(X)]$. On the other hand, $\neg x_a$ also appears in each *conflict* binary clause corresponding to incompatible tuples involving a in constraints binding X. □

If the CSP value ordering heuristic $D\text{-}H_w^{\otimes}$ is applied only on the values of a given variable X (horizontal choice), then all the values appear exactly the same number of times in *at least one* and *at most one* clauses. Consequently, $W[dom(X)]$ becomes useless, and we obtain, also discarding the constant term $w(2)$:

$$D\text{-}H_w[P,(X,a)] = \sum_{C \in \mathscr{C} \mid X \in vars(C)} |\mathcal{C}_f(C,X,a)|$$

Only the number of conflicts is still relevant. In this case $D\text{-}H_w^{\otimes}$ with $\otimes = max$ (respectively $\otimes = min$) thus delivers the same ordering as $max\text{-}conflicts$ (respectively $min\text{-}conflicts$).

3.2.2 Mapping $H_w^{\otimes}$ using Support Encoding

For support encoding, the length of the clauses depends on the number of supports of a value with respect to a given constraint. Consequently, considering $H_w^{\otimes}$ on $\Sigma^S(P)$ the CNF formula obtained using support encoding of the constraint network P, we derive new interesting value orderings.

Definition 5. *Let $P = (\mathscr{X}, \mathscr{C})$ be a constraint network, $X \in \mathscr{X}$ and $a \in dom(X)$.*

$$S\text{-}H_w[P,(X,a)] = W[dom(X)] + \sum_{C \in \mathscr{C} \mid X \in vars(C)} [W_{\downarrow}(C,X,a) + W_{\uparrow}(C,X,a)]$$

such that

$$W[dom(X)] = w(|dom(X)|) + w(2) \times (|dom(X)| - 1)$$
$$W_{\downarrow}(C,X,a) = w[1 + |\mathcal{S}_p(C,X,a)|]$$
$$W_{\uparrow}(C,X,a) = \sum_{(Y,b) \in \mathcal{S}_p(C,X,a)} w[1 + |\mathcal{S}_p(C,Y,b)|]$$

Property. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, $X \in \mathscr{X}$ and $a \in dom(X)$. We have:*
$S\text{-}H_w[P,(X,a)] = H_w[\Sigma^S(P), x_a]$

Proof. As for direct encoding, the first factor $W[dom(X)]$ corresponds to *at least one* and *at most one* clauses. On the other hand, for each constraint C involving the variable X, each literal x_a corresponding to the value a will appear in *support* clauses,

- only one time negatively. The size of the clause is 1 (the negative literal $\neg x_a$) plus the number of supports of a according to the constraint C. This corresponds to the term $W_{\downarrow}(C,X,a)$.
- positively in all support clauses corresponding to values $b \in \mathcal{S}_p(C,Y,b)$. These clauses are of the form $\neg y_b \vee x_{v_1} \cdots \vee x_{v_k}$, with $rel(C) = \{X,Y\}$, $b \in dom(Y)$ and b being a support of $v_1, \ldots, v_k$. The size of these clauses is 1 (the occurrence of the negative literal $\neg y_b$) plus the number of supports of b according to C. This corresponds to the term $W_{\uparrow}(C,X,a)$.

□

If we consider the choice restricted to the values of a given variable, the first argument $W[dom(X)]$ is the same for all these values. This term may then be dropped.

Depending on the weighting function w and the operator $\otimes$, different new value ordering heuristics that exploit the bi-directionality of constraints (take into account *both* branches of the 2-way branching search scheme) can be derived from $S\text{-}H_w^{\otimes}$ using the $\Sigma^S(P)$.

Instantiation 1 ($S\text{-}H_{occ}^{\otimes} = S\text{-}H_{w(\alpha)=1}^{\otimes}$)**.** *The following CSP value ordering heuristic is obtained using $w(\alpha) = 1$ as a weighting function:*

$$S\text{-}H_{occ}[P,(X,a)] = |dom(X)| + |\Gamma(X)| + \sum_{C \in \mathscr{C} \mid X \in vars(C)} |\mathcal{S}_p(C,X,a)|$$

We then obtain the CSP value ordering corresponding to the SAT heuristic based on the occurrence number of literals in $\Sigma^S(P)$. When the choice is restricted to the values of a given variable, the order in which values will be chosen depends only on the number of supports ($|dom(X)|$ and $|\Gamma(X)|$ are constants), which is inversely proportional to the number of conflicts. $S\text{-}H^{min}_{occ}$ (respectively $S\text{-}H^{max}_{occ}$) thus delivers the same ordering as $max\text{-}conflicts$ (respectively $min\text{-}conflicts$).

Instantiation 2 ($S\text{-}H^{\otimes}_{jw} = S\text{-}H^{\otimes}_{w(\alpha)=2^{-\alpha}}$)**.** *The following CSP value ordering heuristic is obtained using $w(\alpha) = 2^{-\alpha}$ as a weighting function :*

$$S\text{-}H_{jw}[P,(X,a)] = \\ W_{jw}[dom(X)] + \tfrac{1}{2} \times \sum_{C \in \mathscr{C} \mid X \in vars(C)} \left[2^{-|\mathcal{S}_p(C,X,a)|} + W_{\uparrow jw}(C,X,a)\right]$$

with

$$W_{jw}[dom(X)] = 2^{-|dom(X)|} + \tfrac{1}{4} \times (|dom(X)| - 1)$$
$$W_{\uparrow jw}(C,X,a) = \sum_{(Y,b) \in \mathcal{S}_p(C,X,a)} 2^{-|\mathcal{S}_p(C,Y,b)|}$$

Instantiation 3 ($S\text{-}H^{\otimes}_{inv} = S\text{-}H^{\otimes}_{w(\alpha)=\alpha}$)**.** *The following CSP value ordering heuristic is obtained using weighting $w(\alpha) = \alpha$ as a weighting function :*

$$S\text{-}H_{inv}[P,(X,a)] = \\ 3 \times |dom(X)| - 2 + |\Gamma(X)| + \sum_{C \in \mathscr{C} \mid X \in vars(C)} [|\mathcal{S}_p(C,X,a)| + W_{\uparrow inv}(C,X,a)]$$

with

$$W_{\uparrow inv}(C,X,a) = |\mathcal{S}_p(C,X,a)| + \sum_{(Y,b) \in \mathcal{S}_p(C,X,a)} |\mathcal{S}_p(C,Y,b)|$$

3.2.3 New value ordering heuristics for CSP

The second and the third instantiations deliver new value ordering heuristics for CSP. Restricting the evaluation to the values of a given variable leads to the following simple CSP value ordering heuristics H_{jw} (respectively H_{inv}) corresponding to the instantiation 2 (respectively 3) where non-discriminating terms have been discarded.

Definition 6. *Let $P = (\mathscr{X}, \mathscr{C})$ be a constraint network, $X \in \mathscr{X}$ and $a \in dom(X)$.*

$$H_{jw}[P,(X,a)] = \sum_{C \in \mathscr{C} | X \in vars(C)} \left[2^{-|\mathcal{S}_p(C,X,a)|} + W_{\uparrow jw}(C,X,a)\right]$$

In the following, we will refer to the new value ordering heuristic H^{max}_{jw} as $max\text{-}jw$. In Example 1, we obtain $H_{jw}[P,(X,a)] = 1.25$ and $H_{jw}[P,(X,b)] = H_{jw}[P,(X,c)] = 0.75$

Using the selection operator $\otimes = max$, the value a is then chosen. This choice clearly corresponds to the best evaluation of both branches corresponding to $X = a$ and $X \neq a$. Indeed, all values in $dom(X)$, when assigned to X, lead to the removal of 2 values from the

Table 1. Summary: CSP heuristics obtained from $D\text{-}H_w^{\otimes}$ and $S\text{-}H_w^{\otimes}$.

Encoding	$H_w^{\otimes}$		**CSP Value Ordering Heuristic**		**Policy**
			known ?	name	
(D)irect	$\otimes = max$	$w(\alpha) = 1$	Y	$max\text{-}conflicts$	$fail\text{-}first$
	$\otimes = min$	$w(\alpha) = 1$	Y	$min\text{-}conflicts$	$promise$
(S)upport	$\otimes = max$	$w(\alpha) = 1$	Y	$min\text{-}conflicts$	$promise$
	$\otimes = min$	$w(\alpha) = 1$	Y	$max\text{-}conflicts$	$fail\text{-}first$
	$\otimes = max$	$w(\alpha) = \alpha$	N	$max\text{-}inverse$	$promise$
	$\otimes = min$	$w(\alpha) = \alpha$	N	$min\text{-}inverse$	$fail\text{-}first$
	$\otimes = max$	$w(\alpha) = 2^{-\alpha}$	N	$max\text{-}jw$	$fail\text{-}first$

domain of Y, whereas removing a from the domain of X leads to the removal of 2 values from the domain of Y instead of 0 values when b or c are removed from the domain of X.

On the other hand, in Example 1, H_{jw} does not discriminate between the different values of Y ($H_{jw}[P(Y,v)] = 0.75\ \forall v \in \{a,b,c,d\}$) even though assigning Y the value a removes two values from $dom(X)$ and selecting $Y = c$ removes only one value.

Definition 7. *Let $P = (\mathscr{X}, \mathscr{C})$ be a constraint network, $X \in \mathscr{X}$ and $a \in dom(X)$.*

$$H_{inv}[P,(X,a)] = \sum_{C \in \mathscr{C} \ \mid \ X \in vars(C)} \left[2 \times |\mathcal{S}_p(C,X,a)| + \sum_{(Y,b) \in \mathcal{S}_p(C,X,a)} |\mathcal{S}_p(C,Y,b)| \right]$$

In the following, we will refer to the new value ordering heuristics H_{inv}^{max} (respectively H_{inv}^{min}) as $max\text{-}inverse$ (respectively $min\text{-}inverse$).

Applying H_{inv} on the variables X and Y of Example 1, we obtain:
$H_{inv}[P,(X,a)] = 8$, $H_{inv}[P,(X,b)] = H_{inv}[P,(X,c)] = 10$,
$H_{inv}[P,(Y,a)] = H_{inv}[P,(Y,b)] = 4$ and $H_{inv}[P,(Y,c)] = H_{inv}[P,(Y,d)] = 8$

We can note that using $min\text{-}inverse$ leads to the same value ordering on the variable X as $max\text{-}jw$, whereas on the variable Y, $min\text{-}inverse$ selects the value a or b and $max\text{-}jw$ derives the same score for all the values from the domain of Y.

All heuristics obtained from $S\text{-}H_w^{\otimes}$ and $D\text{-}H_w^{\otimes}$ are summarized in Table 1, assuming that the value selection is restricted to the values from the same domain of a given variable. Indeed, selecting the best pair (X,v) from all possible pairs of a given constraint network is time consuming. In this last case, no mapping is possible between the instantiations presented above and the known CSP value orderings, i.e. all instantiations lead to new CSP value orderings heuristics of pairs (variable, value).

From Table 1, one can see that the derived CSP value ordering heuristics $H_w^{\otimes}$ (with $\otimes = min$ and $w(\alpha) = 1$) using *(D)irect* encoding leads to the *known* (Y) *promise*-oriented CSP value ordering heuristic $min\text{-}conflicts$, whereas $H_w^{\otimes}$ (with $\otimes = min$ and $w(\alpha) = \alpha$) using *(S)upport* encoding leads to the new (N) $fail\text{-}first$-oriented CSP value ordering heuristic called $min\text{-}inverse$.

3.2.4 Computation cost of the heuristics

In the general case i.e. constraint networks involving n-ary constraints, the maximum number of tuples a constraint may admit is in $O(d^r)$ where r denotes the maximal arity of the constraints. To compute the number of supports for each given variable-value, one needs to enumerate for each constraint the set of allowed tuples. This can be done in $O(ed^r)$, which exactly corresponds to the computation cost of H_{occ} (*min-* and *max-conflicts*). Consequently, we only consider static versions of the proposed value heuristics i.e. the number of supports of each value is only computed prior to the search. For each domain, the values are sorted according to their scores. Consequently, at each node the value selection is done in $O(1)$.

Computing H_{occ} at each node of the search tree is very time consuming. For a given variable (chosen using a variable ordering heuristic), the worst-case time complexity of selecting a given value is in $O(\Gamma_{max}d^r)$, Γ_{max} denoting the maximal number of constraints involving one variable.

For H_{jw} and H_{inv} computation, we proceed in two phases. First, the number of supports of each value is computed. Then, the score of each value according to H_{jw} and H_{inv} is computed. Assuming that $\mathcal{S}_p(C, X, a)$ can be obtained in $O(1)$, the worst-case time complexity to compute H_{jw} and H_{inv} is clearly the same as for H_{occ}. Indeed the two phases admit the same worst-case time complexity ($O(ed^r)$).

4. Experiments

To prove the practical interest of our approach, we have implemented the different heuristics described in the previous sections in our platform *CSP4J* [29] and conducted an experimentation with the full set of 3115 instances used as benchmarks of the first CSP Competition[2.] (the Pseudo-Boolean instances have been discarded because they involve constraints of high arity which prevent an enumeration of all supports).

The search algorithm that has been employed is MGAC embedding GAC3^{rm} [21] and equipped with *dom/wdeg*. All value ordering heuristics have been implemented statically: an ordering is established prior to search and remains unchanged during the whole search process [23].

First, we conducted a full comparison of all heuristics instances from the first round of the 2006 CSP Solver Competition. Figure 2 shows the relative difference in terms of proportion of solved instances with respect to *random* value ordering. All heuristics with positive proportion of solved instances lead to solve more problems than *random* within a timeout of 1200 seconds. Overall results show that *max-inverse* is the best heuristic on the whole set of instances. The other *promise*-oriented heuristic, *min-conflicts*, also gives good results. However, remark that most of the academic instances are easily solved by our solver whatever the value ordering heuristic used. The slight advantage of *max-inverse* appears on All Interval and Golomb Ruler series. On random instances, the *promise*-oriented heuristics present a good behavior. As a summary, the new *promise*-oriented CSP value ordering heuristic *max-inverse* seems to be the most robust one.

2. All instances can be downloaded at http://www.cril.univ-artois.fr/~lecoutre.

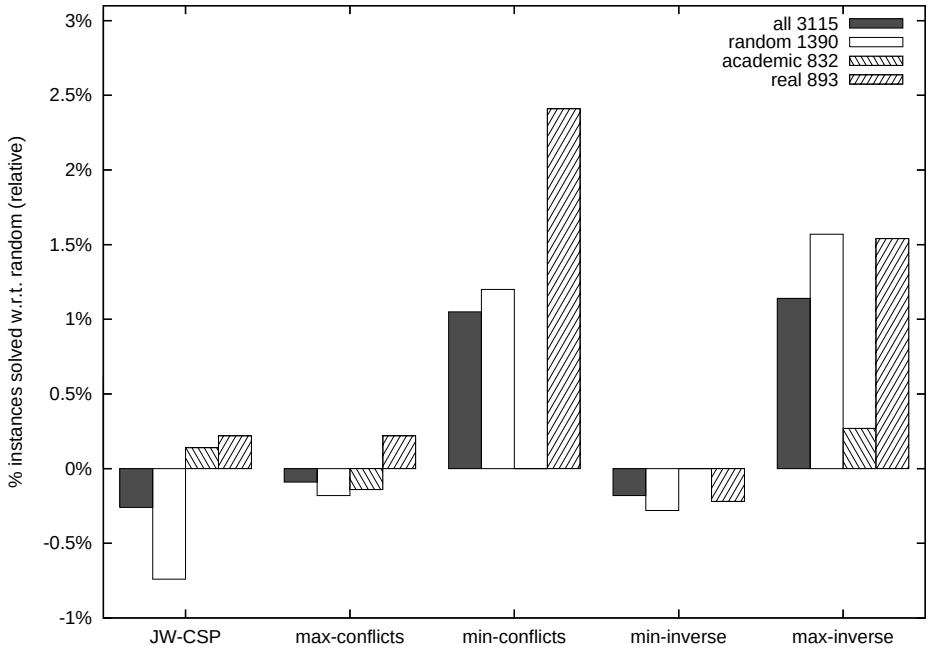

Figure 2. Relative deviation of heuristic performances with respect to *random* in terms of solved instances (timeout = 1200s).

To get a more comprehensive picture on the behavior of *promise* and *fail-first* oriented heuristics, in Figure 3 we give a detailed comparison between the two new proposed CSP value ordering heuristics *max-inverse* (*promise*-oriented) and *min-inverse* (*fail-first*-oriented) on all satisfiable and unsatisfiable instances. Points near the diagonal correspond to instances equally solved by both heuristics. Points under the diagonal are instances for which the algorithm represented on the *y-axis* (*max-inverse* in Figures 3 and 4) is better than the algorithm represented on the *x-axis*. Unsolved instances within a time limit of 1200 seconds are shown on the right hand side and on the top of the two figures. Clearly, on the whole set of instances, *min-inverse* is outperformed by *max-inverse*. This is confirmed on satisfiable instances. Interestingly enough, on unsatisfiable instances, the *fail-first min-inverse* heuristic becomes better that *max-inverse*. To confirm this behavior, we conducted an additional experiment on all random instances, looking for all solutions. In this case, the search space is exhaustively explored for both satisfiable and unsatisfiable instances. Figure 4 shows that *min-inverse* outperforms again *max-inverse*.

Finally, we focused on some Open-Shop scheduling instances. We tested extensively our heuristic against Taillard's Open-Shop Generator [27]. Hardest Open-Shop instances are those that are unsatisfiable but close to the optimal solution. Proving a solution within a makespan t to be optimal involves proving that there is no solution within a makespan of $t-1$. In Table 2, we can observe that *fail-first*-oriented heuristics are the most efficient ones on hard unsatisfiable instances (see median-5-95), whereas the *promise*-oriented heuristics are the best on hard satisfiable instances, where t is equal to the optimal value (see median-

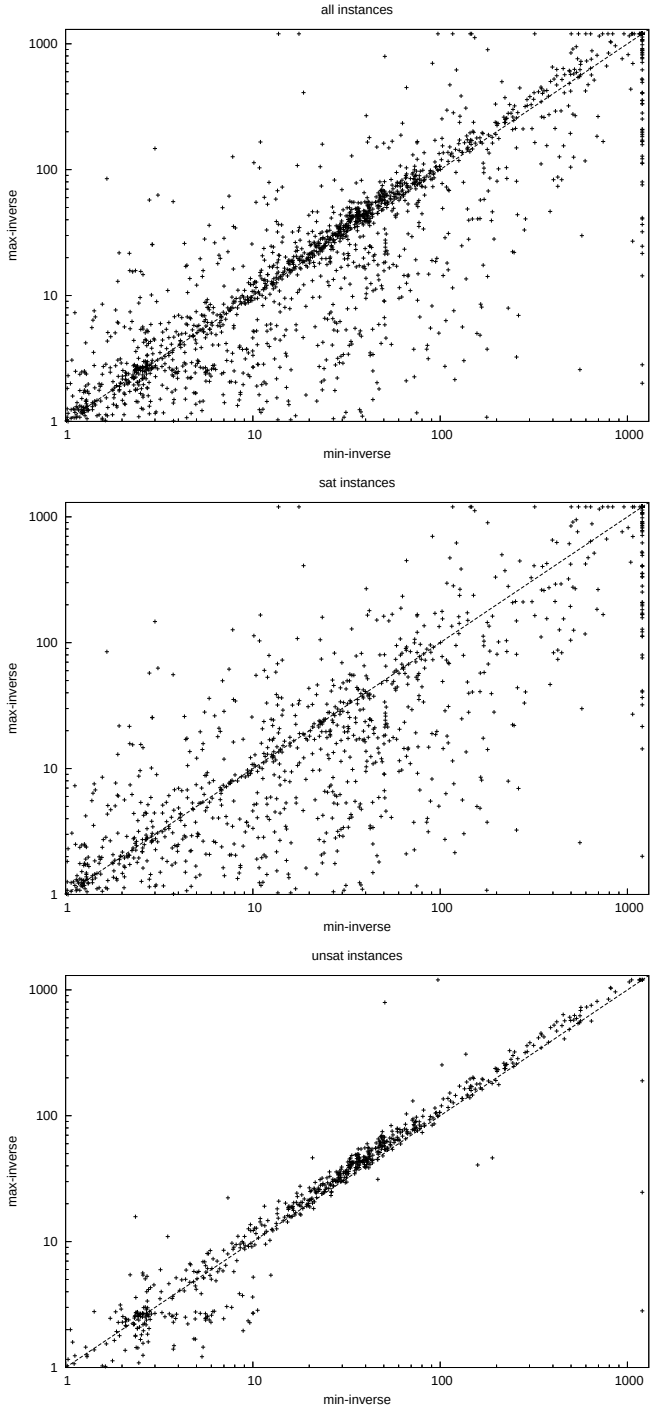

Figure 3. Looking for one solution: comparative performance of $min\text{-}inverse$ and $max\text{-}inverse$ (cpu time against cpu time, in seconds) on respectively all, satisfiable and unsatisfiable competition instances.

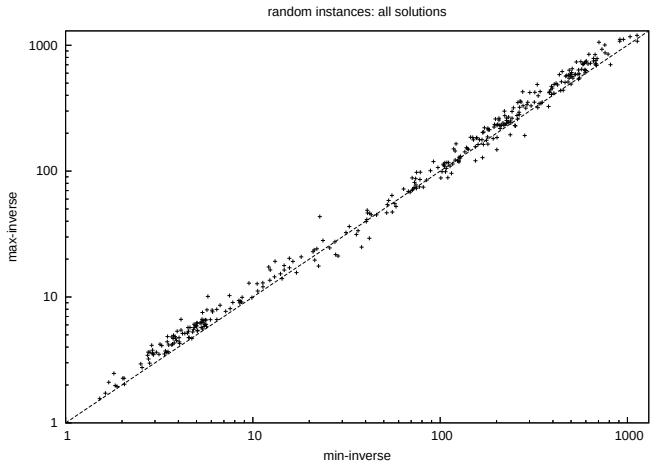

Figure 4. Looking for all solutions: comparative performance of *min-inverse* and *max-inverse*, cpu time against cpu time, in seconds, on random instances.

Table 2. Results details on 5x5 Open Shop instances. Timeout is 1200 seconds CPU time. 5-95- instances are UNSAT whereas 5-100- instances are SAT.

Instance	*random*	*max-jw*	*max-cfl*	*min-cfl*	*min-inv*	*max-inv*
os-taillard-5-95-0	**50.58**	82.67	74.29	111.75	54.7	83.55
os-taillard-5-95-1	8.31	10.16	**5.82**	7.44	7.16	6.48
os-taillard-5-95-2	850.2	462.8	49.22	558.89	189.97	**46.25**
os-taillard-5-95-3	35.26	23.01	25.95	30.88	**22.34**	26.59
os-taillard-5-95-4	1195.4	112.87	151.11	201.37	**97.33**	*timeout*
os-taillard-5-95-5	376.77	269.18	192.79	**71.93**	137.06	308.33
...						
median-5-95	89.39	78.51	72.03	92.94	**63.07**	107.29
os-taillard-5-100-0	*timeout*	248.64	6.68	45.12	**4.17**	*timeout*
os-taillard-5-100-1	45.07	73.91	24.29	**5.64**	12.87	158.54
os-taillard-5-100-2	*timeout*	**239.67**	*timeout*	*timeout*	*timeout*	*timeout*
os-taillard-5-100-3	*timeout*	*timeout*	**36.69**	692.08	*timeout*	*timeout*
os-taillard-5-100-4	**77.16**	*timeout*	*timeout*	*timeout*	*timeout*	*timeout*
os-taillard-5-100-5	*timeout*	*timeout*	829.96	703.45	*timeout*	**251.02**
...						
median-5-100	> 1200	> 1200	> 1200	**402.33**	> 1200	433.33

5-100). Note that not all instances are shown, the median can thus not be deducted from the lines in the table.

5. Conclusion

In this paper, we have focused on converting SAT heuristics to CSP. Thanks to the Support and Direct encodings, we have shown how to derive new CSP value ordering heuristics. Interestingly, these heuristics enabled us to measure for the first time the impact of both positive and negative decisions of binary branching. Using the two-sided Jeroslow-Wang general formulation, we introduced a clear map about the relationships existing between some SAT and CSP heuristics.

We have also shown that $fail\text{-}first$ (respectively *promise*) are more suitable for solving unsatisfiable (respectively satisfiable) instances. Using simpler variants, so-called *min-inverse* and *max-inverse* of the Jeroslow-Wang heuristic allowed to solve more unsatisfiable and satisfiable instances (from the suite of instances of the first round of the 2006 CSP Solver competition) than classical heuristics.

In fact, we noticed that on unsatisfiable instances or when looking for all solutions, following the $fail\text{-}first$ policy does pay off. Our understanding of this phenomenon is that, as $dom/wdeg$ is able to efficiently refute unsatisfiable subtrees, the overhead of refuting more unsatisfiable sub-trees (as more often than not, we guide search towards unsatisfiable sub-trees) is compensated by the benefit of rapidly reducing the search space.

Finally, our experimental results seems to confirm that the gap existing between the different (tested) value ordering heuristics is rather small. However, we think that it is worthwhile to understand the impact of following different policies. Our opinion is that the most efficient the variable ordering heuristic will be (to respect the $fail\text{-}first$ policy), the most interesting to follow the same principle at the value level it could be.

Acknowledgments This work has been supported by the CNRS and the "IUT de Lens". We would like to thank all reviewers for their useful comments that helped to greatly improve the quality of this paper.

References

[1] F. Bacchus. Enhancing Davis Putnam with extended binary clause reasoning. In *Proceedings of AAAI'02*, pages 613–619, 2002.

[2] J.C. Beck, P. Prosser, and R.J. Wallace. Variable ordering heuristics show promise. In *Proceedings of CP'04*, pages 711–715, 2004.

[3] C. Bessiere, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *Selected revised papers from SAT'03*, pages 299–314, 2003.

[4] C. Bessiere and J. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.

[5] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[6] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, **22**:251–256, 1979.

[7] J. de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of IJCAI'89*, pages 290–296, 1989.

[8] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[9] L. Drake, A. Frisch, I. Gent, and T. Walsh. Automatically reformulating SAT-encoded CSPs. In *Proceedings of the RCSP'02 workshop held with CP'02*, 2002.

[10] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. Sat versus unsat. In *Second DIMACS Challenge*, pages 299–314, 1993.

[11] J.W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[12] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of IJCAI'95*, pages 572–578, 1995.

[13] P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of ECAI'92*, pages 31–35, 1992.

[14] I.P. Gent. Arc consistency in SAT. In *Proceedings of ECAI'02*, pages 121–125, 2002.

[15] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, **14**:263–313, 1980.

[16] T. Hulubei and B. O'Sullivan. Search heuristics and heavy-tailed behaviour. In *Proceedings of CP'05*, pages 328–342, 2005.

[17] J. Hwang and D.G. Mitchell. 2-way vs d-way branching for CSP. In *Proceedings of CP'05*, pages 343–357, 2005.

[18] R.G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, **1**:167–187, 1990.

[19] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, **45**:275–286, 1990.

[20] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *Proceedings of ICTAI'04*, pages 549–557, 2004.

[21] C. Lecoutre and F. Hemery. A study of residual supports in arc consistency. In *Proceedings of IJCAI'07*, pages 125–130, 2007.

[22] C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI'97*, pages 366–371, 1997.

[23] D. Meetah and M.R.C. van Dongen. Static value ordering heuristics for constraint satisfaction problems. In *Proceedings of CPAI'05 workshop held with CP'05*, pages 49–62, 2005.

[24] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP'94*, pages 10–20, 1994.

[25] B.M. Smith and P. Sturdy. Value ordering for finding all solutions. In *Proceedings of IJCAI'05*, pages 311–316, 2005.

[26] R. Szymanek and B. O'Sulivan. Guiding search using constraint-level advice. In *Proceedings of ECAI'06*, pages 158–162, 2006.

[27] E. Taillard. Benchmarks for basic scheduling problems. *European journal of operations research*, **64**:278–295, 1993.

[28] M.R.C. van Dongen, editor. *Proceedings of CPAI'05 workshop held with CP'05*, volume **II**, 2005.

[29] J. Vion. Constraint Satisfaction Problem for Java. http://cspfj.sourceforge.net/, 2006.

[30] R.J. Wallace. Heuristic policy analysis and efficiency assessment in constraint satisfaction search. In *Proceedings of CPAI'05 workshop held with CP'05*, pages 79–91, 2005.

[31] T. Walsh. SAT v CSP. In *Proceedings of CP'00*, pages 441–456, 2000.

[32] L. Zhang, C.F. Madigan, M.W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of ICCAD'01*, pages 279–285, 2001.