Pre-processing in Boolean Satisfiability Using Bounded $(2, k)$-Consistency on Regions with Locally Difficult Constraint Setup

Pavel Surynek

Department of Theoretical Computer Science and Mathematical Logic,
Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Malostranské náměstí 25, 118 00 Praha 1, Czech Republic

Graduate School of Maritime Sciences, Division of Maritime Management Sciences,
Kobe University, Japan

5-1-1 Fukae-minamimachi, Higashinada-ku, Kobe 658-0022, Japan

pavel.surynek@mff.cuni.cz, Tel: +420 221 914 139, Fax: +420 221 914 323

Pre-processing in Boolean Satisfiability Using Bounded $(2, k)$-Consistency on Regions with Locally Difficult Constraint Setup

A new type of partially global consistency derived from $(2, k)$-consistency called bounded $(2, k)$-consistency (B2C-consistency) is presented in this manuscript. It is designed for application in Boolean satisfiability (SAT) as a building block for a pre-processing tool. Together with the new B2C-consistency a special mechanism for selecting regions of the input SAT instance with difficult constraint setup was also proposed. This mechanism is used to select suitable difficult sub-problems of which simplification by the consistency can lead to the significant reduction of the effort needed to solve the instance. A new prototype pre-processing tool `preprocessSIGMA` which is based on proposed techniques was implemented. As a proof of new concepts a competitive experimental evaluation on a set of relatively difficult SAT instances was conducted. It showed that our prototype pre-processor is competitive with respect to existent pre-processing tools `LiVer`, `NiVer`, `HyPre`, and `Shatter`.

Keywords: SAT; CSP; SAT pre-processing; local consistency;
global consistency; $(2, k)$-consistency; probability; difficult instances

1 Introduction and Motivation

Recent works dealing with difficult instances of *Boolean Satisfiability* (*SAT*) [1, 2, 7, 9, 21] indicate that an intelligent pre-processing focused on the structure of the instance can dramatically reduce the effort needed to solve it. Technically the pre-processing task is done by transforming the input instance into another one (hopefully simpler) which is subsequently submitted to a general purpose *SAT solver* [7, 12]. It is crucial that the pre-processing step is fast enough relatively to the runtime of the SAT solver on the pre-processed instance.

It this work we further develop ideas from [21] where the input Boolean formula is interpreted as a graph in which graph structures – namely complete sub-graphs – are

identified and after some calculation involving the number and the size of complete sub-graphs, an inference is made. The drawback of the original idea from [21] is that it requires the input instance to be relatively well structured to be able identify an acceptable complete sub-graph decomposition. In this work we overcome this major drawback by two new techniques. First, a **new type of consistency** derived from $(2,k)$-*consistency* [10, 24] called *bounded* $(2,k)$-*consistency with complete graphs* (*B2C-consistency*) is proposed. It uses graph interpretation of a sub-problem on which reasoning over its decomposition into complete sub-graphs is made and can be therefore regarded as a partially global reasoning mechanism. Second, a **new mechanism for selecting** a sub-problem suitable for applying the consistency is proposed. To maximize the benefit of inferences made by the consistency we proposed to apply it on regions of the input instance with locally difficult constraint setup. It means that we are trying to choose such sub-problem for applying the consistency that itself is difficult in certain sense (concentrating on difficulty proved to beneficial in [21] but the old technique required the whole instance to exhibit a difficult constraint setup). We were primarily inspired by the difficulty of well known problems such as *pigeon/hole principle* (*P/H principle*) or *FPGA routing* [1, 2] and we are trying to select regions of the instance which are similar in terms of certain properties to these difficult instances. To do this, a characteristic called *expected number of satisfied tuples of values* is used so that regions that have this characteristic similar to difficult instances are used as sub-problems on which B2C-consistency is applied. In this way we are able to discover sub-problems with hidden difficulty and simplify them with the proposed consistency reasoning which results in faster solving of the output instance.

As a validation of proposed concepts a prototype SAT pre-processing tool `preprocessSIGMA` [25] based on B2C-consistency and the new sub-problem selection

technique has been implemented. The performed experimental evaluation showed that our prototype pre-processing tool is competitive with respect to existent prominent tools such as `LiVer` [20], `NiVer` [20], `HyPre` [5], and `Shatter` [2].

This work has been iteratively developed and preceding works related to the presented one appeared in [22, 23, 24]. The organization of the manuscript is as follows: basic concepts from *constraint programming* [10] and SAT are introduced in **Section 2**. Then the concept of B2C-consistency is developed (**Section 3**). The section that follows (**Section 4**) is devoted to a question of how to build a pre-processing tool exploiting B2C-consistency. Finally (**Section 5**), an extensive experimental evaluation focused on the competitiveness and the investigation of internal properties of the implemented pre-processor is presented.

2 Background from Constraint Programming and Boolean Satisfiability

Let us start with the basic notation and definitions used in the rest of the paper. This section represents the basic background from *constraint programming* [10] and *Boolean satisfiability* [7] which the new concepts rely on.

Definition 1 ***(Constraint Satisfaction Problem)*** [10]**.** A *constraint satisfaction problem* (*CSP*) over a given finite universe $\mathbb{D}$ is a triple (X, D, C) where X is a finite set of *variables*, C is a finite set of *constraints*, and $D: X \longrightarrow 2^{\mathbb{D}}$ is a function assigning each variable a finite *domain*. A constraint $c \in C$ is a construct of the form $\langle [x_1^c, x_2^c, \dots, x_{a^c}^c], R^c \rangle$ where $a^c \in \mathbb{N}$ is an *arity* of constraint c, $[x_1^c, x_2^c, \dots, x_{a^c}^c]$ with $x_i^c \in X$ for $i = 1,2, \dots, a^c$ is called a *scope* of c, and $R^c \subseteq D(x_1^c) \times D(x_2^c) \times \dots \times D(x_{a^c}^c)$ is a relation that enumerates a set of tuples of values for which constraint c is satisfied. □

For simplicity, it is sometimes assumed that $D(x) = \mathbb{D}$ for every $x \in X$. We will use this assumption as well in certain cases. Furthermore it is assumed that we can reorder variables in the scope of a constraint arbitrarily using the above notation. That is for example, if there is a constraint $c = \langle [x, y], R^c \rangle$ in C we can suppose that there is also an equivalent formulation of c as a constraint $e = \langle [y, x], R^e \rangle$ in C where relation R^e can be obtained from R^c by swapping its components.

Definition 2 ***(Solution of CSP)*** [10]**.** An assignment $v: X \longrightarrow \mathbb{D}$ such that $v(x) \in D(x)$ for every $x \in X$ is called a solution of a given CSP (X, D, C) if it is defined for every variable in X and all the constraints in C are satisfied by v. That is, it holds that $[v(x_1^c), v(x_2^c), \dots, v(x_{a^c}^c)] \in R^c$ for every constraint $c = \langle [x_1^c, x_2^c, \dots, x_{a^c}^c], R^c \rangle \in C$. □

Closely related to CSP is *Boolean satisfiability problem* (*SAT*) [7, 9]. It is introduced in the following two definitions. Notice, that in CSP we are trying to find a valuation of variables such that **all** the constraints are satisfied (the conjunction of all the constraints is satisfied). In SAT the task is similar, we are trying to find a Boolean valuation that satisfies **all** the clauses of the input formula (the formula has typically the form of conjunction of clauses).

Definition 3 ***(Boolean Formula)*** [9, 16]**.** A *Boolean formula* in the *conjunctive normal form* (*CNF*) over a given set of Boolean variables Ω is a conjunction: $\bigwedge_{i=1}^{n} \Gamma_i$ where $n \in \mathbb{N}_0$ and each Γ_i with $i \in \{1,2, \dots, n\}$ is a *clause* that puts into disjunction *literals* over variables from Ω. That is, $\Gamma_i = \bigvee_{k=1}^{\alpha^i} \psi_k^i$ for $i = 1,2, \dots, n$ where $\alpha^i \in \mathbb{N}$ is size of the clause and either $\psi_k^i = \beta$ or $\psi_k^i = \neg\beta$ for some variable $\beta \in \Omega$ for every $k = 1,2, \dots, \alpha^i$. □

Definition 4 ***(Boolean Satisfiability Problem)*** [9]**.** A *valuation* of Boolean variables is an assignment $\omega: \Omega \longrightarrow \{FALSE, TRUE\}$. The given valuation of variables ω can be naturally extended to a valuation of formulae over Ω denoted as ω^*. A *Boolean satisfiability problem* (*SAT*) with a formula Φ over Ω is the task of determining whether there exists a valuation ω of Ω such that $\omega^*(\Phi) = TRUE$. □

We are about to work with the concept of *consistencies* [10] in SAT which is however the concept from constraint programming used over CSPs. Hence it is convenient to define translation of SAT to CSP so that we are able to work with consistencies in SAT through this translation. For this purpose we chose a so called *literal encoding* [23] which provides such a translation in the natural way.

Definition 5 ***(Literal Encoding of SAT)*** [23]**.** Let $\Phi = \bigwedge_{i=1}^{n} \Gamma_i$ with $\bigvee_{l=1}^{\alpha^i} \psi_l^i$ for $i = 1,2,\dots,n$ be a Boolean formula in CNF over Ω. A *literal encoding* of Φ is a CSP $E^0(\Phi) = (X_\Phi^0, D_\Phi^0, C_\Phi^0)$ where $X_\Phi^0 = \{\bar{\Gamma}_1, \bar{\Gamma}_2, \dots, \bar{\Gamma}_n\}$, $D_\Phi^0(\bar{\Gamma}_i) = \{\bar{\psi}_l^i | l = 1,2,\dots,\alpha^i\}$ for every $i = 1,2,\dots,n$; and there are constraints between all the pairs of variables as follows: $[\bar{\psi}_l^i, \bar{\psi}_t^j]$ where $\bar{\psi}_l^i \in D_\Phi^0(\bar{\Gamma}_i)$ and $\bar{\psi}_t^j \in D_\Phi^0(\bar{\Gamma}_j)$ is forbidden by a relation R^c defining a constraint $c = \langle[\bar{\Gamma}_i, \bar{\Gamma}_j], R^c\rangle$ with $i, j \in \{1,2,\dots,n\}$, $l \in \{1,2,\dots,\alpha^i\}$, and $t \in \{1,2,\dots,\alpha^j\}$ if there is $\beta \in \Omega$ such that either $\beta = \psi_l^i$ and $\neg\beta = \psi_t^j$ or $\neg\beta = \psi_l^i$ and $\beta = \psi_t^j$. □

The stripe above generic symbols is used to distinguish constant symbols (with the stripe) which do not evaluate from variables (without the stripe) which do evaluate (to other constants). Notice, that literal encoding is a *binary CSP*; that is, all the constraints have the arity of at most 2.

For our purposes, literal encoding is further processed to capture constraints imposed by the original formula more explicitly (notice that there is an incompatibility between complementary literals only at this stage). A new incompatibility is introduced as a constraint between every two literals ψ_l^i and ψ_t^j with $i, j \in \{1,2,\dots,n\}$, such that $i \neq j$, $l \in \{1,2,\dots,\alpha^i\}$, and $t \in \{1,2,\dots,\alpha^j\}$ if the *singleton unit propagation* [11, 21, 22, 23, 24] with setting $\psi_l^i = TRUE$ infers that $\psi_t^j = FALSE$ with respect to Φ (that is, it is set $\psi_l^i = TRUE$; all the other variables are left unassigned and then unit propagation is performed). Let this modification of literal encoding be called an *explicit literal encoding* and it will be denoted as $E^1(\Phi) = (X_\Phi^1, D_\Phi^1, C_\Phi^1)$ (the upper index means that the first stage of inference has been made).

We are now ready to define so called $(2,k)$-consistency [10, 24]. It is a generalization of k-consistency [19] which checks whether a value is supported by a k-tuple of values from the domains of other variables. Within $(2,k)$-consistency, it is checked whether a pair of consistent values has a supporting k-tuple of values. If there is no such supporting k-tuple of values the value or the pair of values respectively can be ruled out from further consideration by the additional constraint.

An auxiliary operation of *projection* denoted as $T|_{A\to B}$ will be used to transform a tuple T into another tuple with respect to patterns A and B. Tuple T and pattern A are of the same size and B is subsumed by A. The result of projection is obtained by matching pattern A on T followed by selecting components of T associated with their counterparts in A that correspond to B (for instance, $[1,2,3]|_{a,b,c\to c,b} = [3,2]$).

Definition 6 ***($(2,k)$-Consistency)*** [10, 24]**.** Let $k \in \mathbb{N}$ be a natural number, (X, D, C) be a CSP, and $x_0, x_1, \dots, x_k, x_{k+1} \in X$ be a $(k+2)$-tuple of distinct variables. A pair of values $d_0 \in D(x_0)$ and $d_{k+1} \in D(x_{k+1})$ with $[d_0, d_{k+1}] \in R^c$ for every binary con-

straint $c = \langle [x_0, x_{k+1}], R^c \rangle$ in C is called to be $(2,k)$-*consistent* with respect to k-tuple of variables $x_1, x_2, \dots, x_k$ if there exists a k-tuple of values $d_1 \in D(x_1)$, $d_2 \in D(x_2)$,..., $d_k \in D(x_k)$ such that for every constraint $e = \langle [z_1^e, z_2^e, \dots, z_{a^e}^e], R^e \rangle$ in C with $\{z_1^e, z_2^e, \dots, z_{a^e}^e\} \subseteq \{x_0, x_1, \dots, x_{k+1}\}$ it holds that $[d_0, d_1, \dots, d_{k+1}]|_{x_0, x_1, \dots, x_{k+1} \rightarrow z_1^e, z_2^e, \dots, z_{a^e}^e} \in R^e$. The pair of values $d_0 \in D(x_0)$ and $d_{k+1} \in D(x_{k+1})$ is called to be $(2,k)$-*consistent* if it is $(2,k)$-consistent with respect to all the k-tuples of variables $x_1, \dots, x_k \in X$. Finally, the CSP (X, D, C) is called to be $(2,k)$-*consistent* if all the pairs of values from domains of every two distinct variables are $(2,k)$-consistent. □

It is not difficult to see that checking whether there exists a supporting k-tuple of values with respect to a fixed k-tuple of variables of unbounded size k is an *NP-complete problem* [17] in both k-consistency and $(2,k)$-consistency (for example the *graph coloring problem* can be reduced to the task of searching for a supporting k-tuple). Hence, unless $P = NP$ the support cannot be found in polynomial time. Another simple observation is that the support with respect to a fixed tuple of variables can be found in $\mathcal{O}(|\mathbb{D}|^k)$ by traversing all the involved k-tuples of values. This is also the currently best known upper bound of the time complexity of the search for a support within $(2,k)$-consistency enforcing algorithms [10].

Both discussed higher level consistencies represent powerful techniques when k is bounded by the number of variables only. After enforcing k-consistency/$(2,k)$-consistency with k high enough it is possible to obtain a solution of a problem in back-track-free manner [10]. Without providing more details, the high enough k means that is at least the *width of the constraint graph* of the given CSP which is at most the number of variables [13].

3 Bounded $(2, k)$-Consistency with Complete Graphs – B2C Consistency

Our new concept of a so called *bounded* $(2, k)$*-consistency with complete graphs* (*B2C-consistency*) combines the inference strength of $(2, k)$-consistency with partial global reasoning. The global oriented reasoning in SAT which is of our interest has been first introduced in [21] and newer versions appeared in [22, 23, 24]. Particularly, the idea of exploiting global information reflected in complete sub-graphs in a certain graph interpretation of the problem was taken from previous works and was further elaborated. However, the global reasoning itself turned out to be unilateral and hence not ideally suitable for using in SAT preprocessing. Therefore it is suggested in this work to enhance global reasoning with $(2, k)$-consistency which is universal enough to cover cases where global reasoning is unsuitable. If both approaches – global and $(2, k)$-consistency - are applied together a synergic effect arises in certain situations.

Local consistencies such as k-consistency and related ones in SAT have been studied in several works [6, 18, 27]. The common approach in these works is to encode a given task so that a local consistency of interest is simulated by *unit propagation* [11]. Our approach takes an instance of SAT problem as a list of clauses (constraints) and applies the consistency directly without caring of the way how the original task was encoded into the instance. The result is a set of forbidden value assignments in the case of *B2C-consistency* which is subsequently submitted to a SAT solver together with the original instance as a list of additional clauses.

The major obstacle with $(2, k)$-consistency is that it is difficult to be enforced as it is necessary to search for a consistent k-tuple of values which means to traverse the search space of the size of $|\mathbb{D}|^k$ in the worst case (supposed that all the variables have identical domain of $\mathbb{D}$). Hence, to preserve low computation costs of the consistency enforcing algorithm we suggest to bound the consistency somehow. It has been chosen

to bound the number of steps of the search for a consistent k-tuple by a constant Λ.

B2C-consistency is again defined with respect to a $(k+2)$-tuple of distinct variables. Again, it checks whether a given pair of values from domains of two distinct variables has a supporting k-tuple in domains of remaining k variables. The upcoming sections describes how the new consistency is enforced supposed that $(k+2)$-tuple of variables has been already determined. The process how a promising $(k+2)$-tuple can be selected is discussed later.

A Graph Derived from SAT – Graph Interpretation

Let $E^1(\Phi) = (X^1_\Phi, D^1_\Phi, C^1_\Phi)$ be an explicit literal encoding of a given Boolean formula Φ. Next, let us have $k \in \mathbb{N}$ and an ordered $(k+2)$-tuple of selected variables $K^2_+ = [\bar{\Gamma}_{i_0}, \bar{\Gamma}_{i_1}, \bar{\Gamma}_{i_2}, \dots, \bar{\Gamma}_{i_k}, \bar{\Gamma}_{i_{k+1}}] \subseteq X^1_\Phi$ with $i_0, i_1, \dots, i_{k+1} \in \{1,2,\dots,n\}$ where $i_\zeta \neq i_\xi$ for $\zeta, \xi \in \{0,1,\dots,k+1\}$ with $\zeta \neq \xi$.

It is more convenient to define the consistency with respect to an undirected graph derived from the constraint network. The target undirected graph will be represented by a so called *graph interpretation* in the given context. It is defined with respect to K^2_+ as an undirected graph $I(K^2_+) = (I_V, I_E)$ where a set of vertices I_V consists of $\bigcup_{\zeta=0}^{k+1}\{\bar{\psi}_l^{i_\zeta} | l = 1,2,\dots,\alpha^{i_\zeta}\}$ and a set of edges I_E contains an edge $\{\bar{\psi}_l^{i_\zeta}, \bar{\psi}_t^{i_\xi}\}$ with $\zeta, \xi \in \{0,1,\dots,k+1\}$ such that $\zeta \neq \xi$, $l \in \{1,2,\dots,\alpha^{i_\zeta}\}$, and $t \in \{1,2,\dots,\alpha^{i_\xi}\}$ if it holds that $[\bar{\psi}_l^{i_\zeta}, \bar{\psi}_t^{i_\xi}] \notin R^c$ for some constraint $c = \langle[\bar{\Gamma}_{i_\zeta}, \bar{\Gamma}_{i_\xi}], R^c\rangle$ in C_Φ (edges stand for forbidden pairs of values; that is, an edge represents a *conflict*).

Boolean Formula Φ

$\Omega = \{\beta_1, \beta_2, \beta_3\}$

$\Phi =$
Γ_1: ($\beta_1 \vee \beta_2$) $\wedge$
Γ_2: ($\neg\beta_1 \vee \neg\beta_2$) $\wedge$
Γ_3: ($\beta_1 \vee \neg\beta_3$) $\wedge$
Γ_4: ($\neg\beta_1 \vee \beta_2 \vee \beta_3$)

Explicit Literal Encoding $E^1(\Phi)$

$E^1(\Phi) = (X^1_\Phi, D^1_\Phi, C^1_\Phi)$ $\quad X^1_\Phi = \{\bar{\Gamma}_1, \bar{\Gamma}_2, \bar{\Gamma}_3, \bar{\Gamma}_4\}$

$D^1_\Phi(\bar{\Gamma}_1) = \{\bar{\beta}_1, \bar{\beta}_2\}$ $\quad D^1_\Phi(\bar{\Gamma}_2) = \{\neg\bar{\beta}_1, \neg\bar{\beta}_2\}$
$D^1_\Phi(\bar{\Gamma}_3) = \{\bar{\beta}_1, \neg\bar{\beta}_3\}$ $\quad D^1_\Phi(\bar{\Gamma}_4) = \{\neg\bar{\beta}_1, \bar{\beta}_2, \bar{\beta}_3\}$

$C^1_\Phi = \{c_{\{1,2\}}, c_{\{1,3\}}, c_{\{1,4\}}, c_{\{2,3\}}, c_{\{2,4\}}, c_{\{3,4\}}\}$

$$c_{\{1,2\}} = \langle[\bar{\Gamma}_1, \bar{\Gamma}_2], D^1_\Phi(\bar{\Gamma}_1) \times D^1_\Phi(\bar{\Gamma}_2) \setminus \{[\bar{\beta}_1, \neg\bar{\beta}_1], [\bar{\beta}_2, \neg\bar{\beta}_2]\}\rangle$$
$$c_{\{1,3\}} = \langle[\bar{\Gamma}_1, \bar{\Gamma}_3], D^1_\Phi(\bar{\Gamma}_1) \times D^1_\Phi(\bar{\Gamma}_3) \setminus \{[\bar{\beta}_1, \bar{\beta}_1]\}\rangle$$
$$c_{\{1,4\}} = \langle[\bar{\Gamma}_1, \bar{\Gamma}_4], D^1_\Phi(\bar{\Gamma}_1) \times D^1_\Phi(\bar{\Gamma}_4) \setminus \{[\bar{\beta}_1, \neg\bar{\beta}_1], [\bar{\beta}_1, \bar{\beta}_2], [\bar{\beta}_1, \bar{\beta}_3], [\bar{\beta}_2, \bar{\beta}_3]\}\rangle$$
$$c_{\{2,3\}} = \langle[\bar{\Gamma}_2, \bar{\Gamma}_3], D^1_\Phi(\bar{\Gamma}_2) \times D^1_\Phi(\bar{\Gamma}_3) \setminus \{[\neg\bar{\beta}_2, \bar{\beta}_2]\}\rangle$$
$$c_{\{2,4\}} = \langle[\bar{\Gamma}_2, \bar{\Gamma}_4], D^1_\Phi(\bar{\Gamma}_2) \times D^1_\Phi(\bar{\Gamma}_4) \setminus \{[\neg\bar{\beta}_2, \neg\bar{\beta}_1], [\neg\bar{\beta}_2, \bar{\beta}_2], [\neg\bar{\beta}_2, \bar{\beta}_3]\}\rangle$$
$$c_{\{3,4\}} = \langle[\bar{\Gamma}_3, \bar{\Gamma}_4], D^1_\Phi(\bar{\Gamma}_3) \times D^1_\Phi(\bar{\Gamma}_4) \setminus \{[\bar{\beta}_2, \bar{\beta}_3], [\neg\bar{\beta}_3, \bar{\beta}_3]\}\rangle$$

Graph Interpretation $I(K^2_+)$

$K^2_+ = [\bar{\Gamma}_1, \bar{\Gamma}_2, \bar{\Gamma}_4]$
$I(K^2_+) = (I_V, I_E)$

$I_V = \{\bar{\psi}^1_1, \bar{\psi}^1_2, \bar{\psi}^2_1, \bar{\psi}^2_2, \bar{\psi}^3_1, \bar{\psi}^3_2, \bar{\psi}^3_3\}$
$I_E =$
$\{\{\bar{\psi}^1_1, \bar{\psi}^2_1\}, \{\bar{\psi}^1_1, \bar{\psi}^4_1\}, \{\bar{\psi}^1_1, \bar{\psi}^4_2\},$
$\{\bar{\psi}^1_2, \bar{\psi}^2_2\}, \{\bar{\psi}^1_2, \bar{\psi}^4_3\},$
$\{\bar{\psi}^2_2, \bar{\psi}^4_1\}, \{\bar{\psi}^2_2, \bar{\psi}^4_2\}, \{\bar{\psi}^2_2, \bar{\psi}^4_3\}\}$

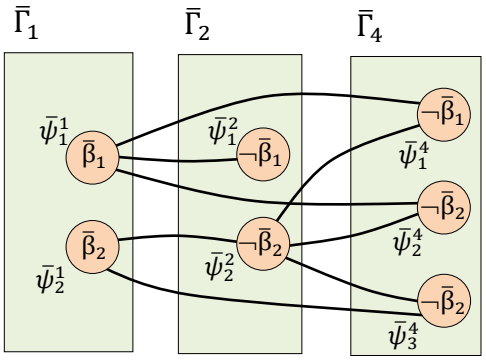


Figure 1. *Graph interpretation.* An original input Boolean formula Φ with four clauses is shown (upper left). Then a corresponding explicit literal encoding (upper right – that is, a literal encoding after singleton unit propagation) – the CSP model consisting of four variables is shown. The lower part depicts a graph interpretation over three variables selected in the CSP model.

Initial Setup of B2GS-Consistency

We are about to utilize structural information contained in the graph interpretation. It has been shown in previous works [21, 22, 23] that useful structural information is constituted by the knowledge of complete constraint sub-graphs. Regarding the given context, we can observe that **at most one literal** can be satisfied in a complete sub-graph in the graph interpretation of a literal encoding of a SAT instance. If a large enough com-

plete sub-graph is detected in the graph interpretation its knowledge can be used for an efficient search space pruning or a strong global inference. It will be elaborated in the following text how this is exactly done.

A *decomposition into complete sub-graphs* of a given graph interpretation $I(K_+^2) = (I_V, I_E)$ is constructed first. It is a task of finding a number $\delta \in \mathbb{N}$ and sets I_V^1, I_V^2,..., $I_V^\delta \subseteq I_V$ called *decomposition sets* that satisfy the following conditions:

(i) $\bigcup_{i=1}^{\delta} I_V^i = I_V$; that is, all the vertices are covered by the decomposition;

(ii) $I_V^i \not\subseteq I_V^j$ for any two $i, j \in \{1,2,\dots,\delta\}$ such that $i \neq j$; that is, the decomposition is not allowed to contain redundancies;

(iii) I_V^i induces a complete sub-graph over edges from I_E from for every $i \in \{1,2,\dots,\delta\}$;

(iv) $\forall u, v \in I_V$ with $\{u, v\} \in I_E$ there exists $i \in \{1,2,\dots,\delta\}$ such that $\{u, v\} \subseteq I_V^i$; that is, all the edges are covered by complete sub-graphs.

Observe that if no further objective is imposed on the decomposition into complete sub-graphs it can be easily constructed by setting $\delta = |I_E|$ and putting each edge into its own decomposition vertex set. On the other hand the construction of decomposition with respect to any reasonable objective (such as maximizing the size of complete sub-graphs or minimizing the number δ) is a difficult task [14, 17].

In our approach we are trying to obtain large complete sub-graphs. However, this requirement is not that strict so we settle for a greedy approach for the construction of a decomposition. The greedy algorithm used in our work is shown using pseudo-code as Algorithm 1.

The algorithm always prefers a vertex with the highest degree with respect to the remaining set of edges. Such a vertex is included into the constructed complete graph and the task is reduced on its neighborhood. This is repeated until the neighborhood of

the currently constructed complete sub-graph is non-empty (neighborhood of a complete sub-graph is a set of vertices that are connected to all of the vertices of the sub-graph). Once the complete sub-graph is finished its edges are removed from the original graph and the process continues until there are any edges.

Algorithm 1. *Greedy algorithm for decomposing a graph interpretation into complete sub-graphs.* The output decomposition is returned as a sequence of decomposition sets of vertices where each of them induces a complete sub-graph.

function *Decompose-Graph-Interpretation*($I(K_+^2) = (I_V, I_E)$): **sequence**
    /* Parameters: $I(K_+^2)$ - a graph interpretation for decomposing */
1: $\delta \leftarrow 1$
2: **while** $I_E \neq \emptyset$ **do**
3: $\quad I_V^\delta \leftarrow \emptyset$
4: $\quad (T_V, T_E) \leftarrow (I_V, I_E)$ /* an auxiliary graph for gradual dismantling */
5: $\quad$ **while** $T_V \neq I_V^\delta$ **do**
6: $\quad\quad$ **let** $v_{max} \in T_V \setminus I_V^\delta$ be a vertex such that $\deg_{(T_V,T_E)}(v_{max}) =$
7: $\quad\quad\quad \max\{\deg_{(T_V,T_E)}(v) \,|v \in T_V \setminus I_V^\delta\}$
8: $\quad\quad I_V^\delta \leftarrow I_V^\delta \cup \{v_{max}\}$
9: $\quad\quad T_V \leftarrow T_V \setminus \{u | \{v_{max}, u\} \notin T_E\})$
10: $\quad\quad T_E \leftarrow T_E \cap \binom{T_V}{2}$
11: $\quad I_E \leftarrow I_E \setminus \binom{I_V^\delta}{2}$
12: $\quad I_V \leftarrow I_V \setminus \{v \in I_V | \deg_{(I_V,I_E)}(v) = 0\}$
12: $\quad \delta \leftarrow \delta + 1$
13: **return** $[I_V^1, I_V^2, \ldots, I_V^\delta]$

The construction of decomposition as shown in Algorithm 1 heuristically prefers construction of large complete sub-graph at the beginning. This strategy proved to produce decompositions of acceptable quality for sub-sequent usage within the B2C-consistency enforcing algorithm.

Proposition 1 ***(Greedy Time/Space Complexity).*** *The greedy algorithm for decomposition of a graph interpretation $I(K_+^2) = (I_V, I_E)$ into complete sub-graphs can be implemented to have the worst case time complexity of $\mathcal{O}(|I_E||I_V|^2)$). The corresponding*

worst case space is of $\mathcal{O}(|I_V| + |I_E|)$. ■

Commentary: Observe that there may be up to $|I_E|$ complete sub-graphs in the decomposition (each edge constitutes a decomposition set). All the edges of the input graph interpretation may be investigated within the construction of an individual complete sub-graph which adds $|I_E|$ steps (which is $\mathcal{O}(|I_V|^2)$). Adding a vertex with the maximum degree into a complete sub-graph consumes $|I_V|$ steps while it may be repeated up to $|I_V|$ times. Altogether we have $|I_V|^2$ steps for one complete sub-graph.

Regarding the space complexity it can be argued that several copies of the input graph need to be stored which makes $\mathcal{O}(|I_V| + |I_E|)$ if neighborhood of a vertex is represented using linked lists. ■

There are some more properties of the decomposition into complete sub-graphs. Notice that decomposition sets intersect vertices corresponding to a domain of a single variable at most once. This is due to the fact that there are no edges between vertices corresponding to the single domain and due to condition (iii). On the other hand a single vertex may be included into several decomposition sets.

B2C-Consistency Enforcing Algorithm

B2C-consistency will be defined algorithmically as it is the most natural way to do that. Suppose that a decomposition into complete sub-graphs of a given graph interpretation has been already constructed. The basic idea is to enforce bounded $(2, k)$-consistency using only Λ steps in the search for a supporting k-tuple. This search will be accompanied by a special pruning which will use the decomposition in complete sub-graphs to obtain more global reasoning. It is supposed that the search is done in some systematic way by extending partial selection of a supporting tuple of values. No matter how exact-

ly the search for the support proceeds we can assume that some values/vertices are selected into the partial supporting tuple at every time of the process. The selection automatically rules out several other values/vertices – more precisely, the values/vertices that are together with selected ones in some complete sub-graph are ruled out (this is due to the condition that at most one literal can be selected in a complete sub-graph).

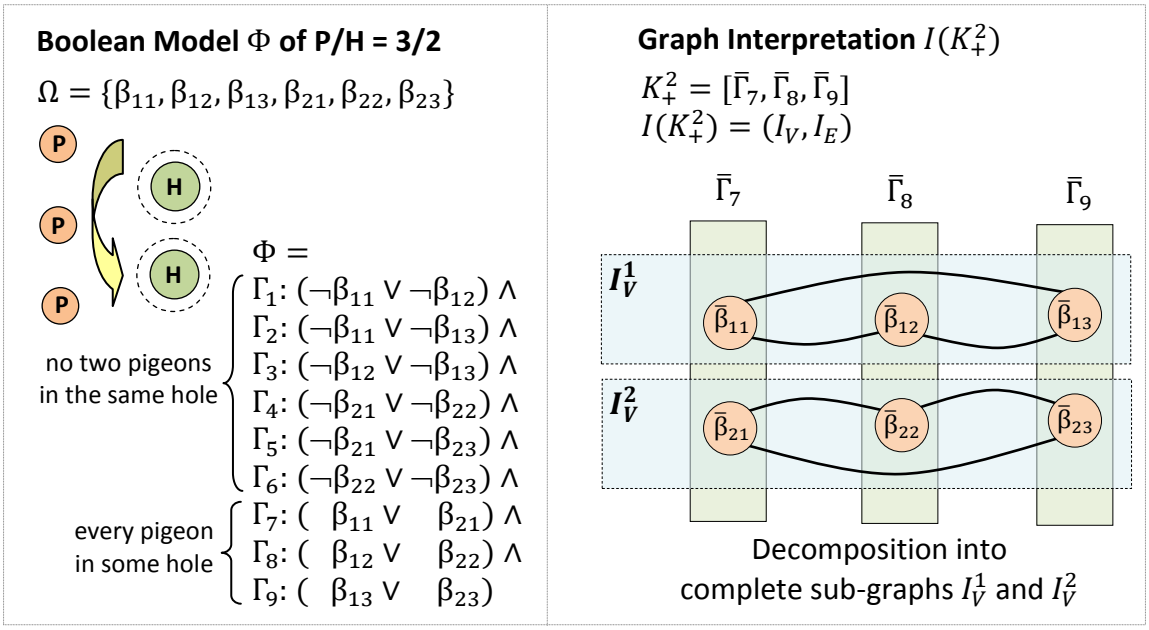


Figure 2. *Pigeon hole (P/H) principle – graph interpretation with complete sub-graphs.* The standard Boolean model of the P/H principle Φ for $p = 3$ and $h = 2$ is shown in the left part. A graph interpretation over the explicit literal encoding of Φ with selected variables $\bar{\Gamma}_7$, $\bar{\Gamma}_8$, and $\bar{\Gamma}_9$ is shown in the right part together with its decomposition into complete sub-graphs (notice that the decomposition shown here can be found by the presented greedy algorithm - Algorithm 1).

Nevertheless, the main innovative reasoning mechanism uses the decomposition in a different way. At every time of the process there are still some candidate values/vertices for selection into the final supporting k-tuple. Each one is included in some decomposition sets from which no value/vertex has been selected yet. Let $\mathcal{L}$ be a set of such not yet used decomposition sets and let S be a set of already selected vertices. As only one value/vertex can be selected from each complete sub-graph we can make the

following pruning: if it happens that $|\mathcal{L}| + |S| < k$ the search in the current branch of the search tree can be terminated as it is not possible to extend the partial selection so that it will finally consist of k elements. This kind of reasoning is especially useful for problems with **non-local** properties such as P/H principle or FPGA Switch-Box routing [1]. For illustration see Figure 2 (if $\bar{\beta}_{11}$ and $\bar{\beta}_{23}$ have been already selected, then $\mathcal{L} = \emptyset$, $k = 3$, and $S = \{\bar{\beta}_{11}, \bar{\beta}_{23}\}$ and hence we can conclude that $\bar{\beta}_{11}$ and $\bar{\beta}_{23}$ inconsistent).

The process of consistency enforcing with B2C-consistency for a pair of values and a fixed tuple of variables $K_+^2 = [\bar{\Gamma}_{i_0}, \bar{\Gamma}_{i_1}, \bar{\Gamma}_{i_2}, \ldots, \bar{\Gamma}_{i_k}, \bar{\Gamma}_{i_{k+1}}]$ is shown as Algorithm 2. The algorithm searches for a supporting k-tuple of values for a given pair of values $\bar{\psi}_{t_0}^{i_0} \in D(\bar{\Gamma}_{i_0})$ and $\bar{\psi}_{t_{k+1}}^{i_{k+1}} \in D(\bar{\Gamma}_{i_{k+1}})$ in domains of $\bar{\Gamma}_{i_1}, \bar{\Gamma}_{i_2}, \ldots, \bar{\Gamma}_{i_k}$. The search is done by systematic extension of the current partial selection of supporting values/vertices. This functionality is implemented using recursive calls which simulates chronological backtracking search.

The algorithm for enforcing B2C-consistency for a pair of values should be regarded as an incomplete prover of non-existence of a support. That is, if the algorithm finds the given pair of values to be inconsistent then there is actually no support for them (that is, it managed to prove that there is no support using Λ search steps and other techniques; $FALSE$ is returned by *Check-B2C-Consistency* in this case). However, if it does not find the given pair of values to be inconsistent then one of the following cases might happen: a supporting k-tuple of values was found or the algorithm ran out of the allowed number of search steps Λ ($TRUE$ is returned in this case).

Proposition 2 ***(B2C Time/Space Complexity).*** *If $\Lambda = \infty$ then the algorithm for enforcing B2C-consistency with a decomposition into complete sub-graphs $\mathcal{J}$ of a graph inter-*

pretation $I(K_+^2) = (I_V, I_E)$ *of a* $(k+2)$*-tuple of variables* K_+^2 *can be implemented to have the worst case time complexity of* $\mathcal{O}(k|\mathcal{J}||\mathbb{D}|^k)$*; otherwise the worst case time complexity is* $\mathcal{O}(|\mathcal{J}|\Lambda)$*. The corresponding worst case space complexity is* $\mathcal{O}(|I_V| + |I_E|)$. ■

Algorithm 2. *Search for a supporting k-tuple of values within B2C-consistency.* It is supposed that a decomposition into complete sub-graphs $\mathcal{J}$ of a given graph interpretation $I(K_+^2)$ with respect to a $(k+2)$-tuple of variables K_+^2 has been already calculated.

---

function *Check-B2C-Consistency*$(\bar{\psi}_{t_0}^{i_0}, \bar{\psi}_{t_{k+1}}^{i_{k+1}}, I(K_+^2) = (I_V, I_E), \mathcal{J}, \Lambda)$: **boolean**

/* Parameters: $\bar{\psi}_{t_0}^{i_0}, \bar{\psi}_{t_{k+1}}^{i_{k+1}}$ - a pair of values for consistency checking
$I(K_+^2)$ - a graph interpretation for decomposing,
$\mathcal{J}$ - a decomposition of $I(K_+^2)$ into complete sub-graphs,
Λ - the number of allowed search steps. */

1: $(\omega, \Lambda) \leftarrow$ *Search-B2C-Support*$(\bar{\psi}_{t_0}^{i_0}, \bar{\psi}_{t_{k+1}}^{i_{k+1}}, \emptyset, I(K_+^2) = (I_V, I_E), \mathcal{J}, \Lambda)$
2: **return** ω

function *Search-B2C-Support*$(\bar{\psi}_{t_0}^{i_0}, \bar{\psi}_{t_{k+1}}^{i_{k+1}}, S, I(K_+^2) = (I_V, I_E), \mathcal{J}, \Lambda)$: **pair**

/* Parameters: S - a set of already selected supports. */

1: **if** $|S| = k$ **then return** $(TRUE, \Lambda)$
2: **let** $[\bar{\psi}_{t_1}^{i_1}, \bar{\psi}_{t_2}^{i_2}, \ldots, \bar{\psi}_{t_l}^{i_l}] = S$
3: **for each** $\bar{\psi}_{t_{l+1}}^{i_{l+1}} \in D(\bar{\Gamma}_{i_{l+1}})$ do
4: **if** $\Lambda \leq 0$ **then return** $(TRUE, \Lambda)$ /* all the steps were consumed */
5: $\alpha \leftarrow TRUE$
6: **for each** $I_V \in \mathcal{J}$ **do** /* check of constraints */
7: **if** $\left|I_V \cap (\cup[\bar{\psi}_{t_0}^{i_0}].S.[\bar{\psi}_{t_{l+1}}^{i_{l+1}}, \bar{\psi}_{t_{k+1}}^{i_{k+1}}])\right| > 1$ **then** $\alpha \leftarrow FALSE$
8: **let** $\mathcal{L} = \{I_V \in \mathcal{J} | I_V \cap (\cup S.[\bar{\psi}_{t_{l+1}}^{i_{l+1}}]) = \emptyset\}$
9: **if** $|\mathcal{L}| + |S| + 1 < k$ **then** $\alpha \leftarrow FALSE$ /* global check */
10: **if** α **then**
11: **if** $l + 1 < k$ **then** /* some supports still remain to be found */
12: $(\omega, \Lambda) \leftarrow$ *Search-B2C-Support*$(\bar{\psi}_{t_0}^{i_0}, \bar{\psi}_{t_{k+1}}^{i_{k+1}}, S.[\bar{\psi}_{t_{l+1}}^{i_{l+1}}],$
13: $I(K_+^2), \mathcal{J}, \Lambda)$
14: **if** ω **then return** $(TRUE, \Lambda)$
15: **else** /* all the supports have been found */
16: **return** $(TRUE, \Lambda)$
17: $\Lambda \leftarrow \Lambda - 1$
18: **return** $(FALSE, \Lambda)$

Commentary: It is not difficult to observe that the algorithm needs to go through all the $|\mathbb{D}|^k$ k-tuples in the worst case if the number of allowed search steps Λ is unbounded. Checking a k-tuple may consume up to $k|\mathcal{J}|$ constraint checks (namely checks against complete sub-graphs). If Λ is bounded then obviously at most Λ steps are done while each step consumes up to $|\mathcal{J}|$ constraint checks.

As all the data elements are accessed sequentially no extra data structures are needed. Hence we need to store graph interpretation and its decomposition into complete sub-graphs which we already know to be of $\mathcal{O}(|I_V| + |I_E|)$. The space needed to store resulting k-tuple is again of $\mathcal{O}(|I_V| + |I_E|)$. ■

Here it depends what is our perception of k. It is natural perceive it as part of the input and hence the complexity of search for a support is exponential with unbounded Λ. Therefore the time consumption represents a main bottleneck of the method. However, having the global reasoning based on complete sub-graphs still much can be done in Λ steps while Λ is bounded.

4 Building a SAT Pre-processing Tool

We intended to use B2C-consistency as a basis for a SAT pre-processing tool. As we have seen it may not be simply used for that task in its raw form due to its time complexity. A good compromise between computational effort and strength of the inference has to be found. This section describes how a tuple of variables should be chosen and how to set particular parameters of B2C-consistency to be suitable for the intended pre-processing tool.

Selection of k-tuples of CSP Variables

As it is computationally infeasible to achieve B2C-consistency with respect to all the

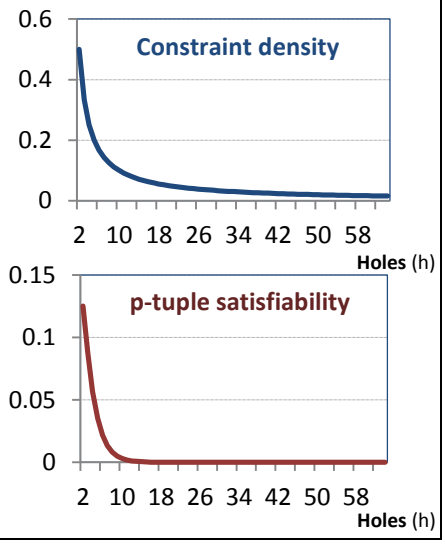
k-tuples of variables and pairs of values in their domains in a non-trivially large SAT instance some selection of promising subsets of variables on that the consistency will be applied has to be done. The selection is considered to be promising if there is a chance that the consistency rules out some pair of values (that is, the ruled out pair of values cannot be part of any solution). At the same time, the information captured in the fact that a given pair of values is incompatible should be valuable for a SAT solver in a certain sense. This requirement is imposed by the intention to use B2C-consistency as a preprocessing tool. Hence, the information should not be easily derivable by the SAT solver itself since informing the SAT solver about inconsistency between a pair of trivially incompatible values is not helpful.

Our approach is to select k-tuples of variables that induces a region of the instance with difficult constraint setup that however can be tackled by the consistency. Such a setup provides a chance to extract valuable information by B2C-consistency. The well known SAT model of *pigeon hole principle* (P/H principle) – more precisely the explicit literal encoding of it – is a representative of such a setup which is well known for resisting from being handled by SAT solvers [1]. All the instances of P/H principle are unsatisfiable. Having a suitable graph interpretation for P/H principle as it is shown in Figure 2 (that is, clauses modeling that each pigeon is placed in some hole are selected as a $(k+2)$-tuple of CSP variables for the graph interpretation) we are able to calculate various useful probabilistic characteristics.

Let p be the number of pigeons and let h be the number of holes where it holds that $h = p - 1$. A *constraint density* ρ in a binary CSP will be defined as the ratio of the number of allowed pairs of values to the number of all the possible pairs of values. Particularly in the case of P/H principle it holds that $\rho = \frac{\binom{p}{2}h}{\binom{p}{2}h^2} = \frac{1}{h}$ in the graph interpretation as described above.

Table 1. Probabilistic characteristics of graph interpretation of pigeon hole principle.

Configuration: **pigeons** (p) × **holes** ($h = p - 1$)	Constraint **density** ρ $\frac{\binom{p}{2}h}{\binom{p}{2}h^2} = \frac{1}{h}$	Probability of **satisfiability** of a random p-tuple σ $(1-\frac{1}{h})^{\binom{h+1}{2}}$	**Expected number** of satisfied p-tuples ε $h^{h+1}(1-\frac{1}{h})^{\binom{h+1}{2}}$
3 × 2	0.5	0.125	1
4 × 3	0.333333	0.087791	7.111111
5 × 4	0.25	0.056314	57.66504
6 × 5	0.2	0.035184	549.7558
7 × 6	0.166667	0.021737	6084.888
8 × 7	0.142857	0.01335	76961.62



Another interesting characteristic is the probability of a randomly selected assignment of values to p variables σ calculated from the constraint density. It is a reasonable assumption that satisfaction of individual pairs of values within the assignment is independent on each other. Then it holds for the *probability of satisfaction* of a random p-tuple of values that $\sigma = (1-\rho)^{\binom{p}{2}}$ which is $(1-\frac{1}{h})^{\frac{(h+1)h}{2}}$ in the case of P/H principle. Finally, we will investigate the expected number of satisfied p-tuples of values ε which will be defined as the total number of possible p-tuples multiplied by π. It holds that $\varepsilon = h^p\sigma = h^{h+1}(1-\frac{1}{h})^{\frac{(h+1)h}{2}}$ in the case of P/H principle. Several examples of probabilistic characteristics are shown in Table 1. The limit behavior of above characteristics with $h \to \infty$ is summarized in the following easy to prove proposition.

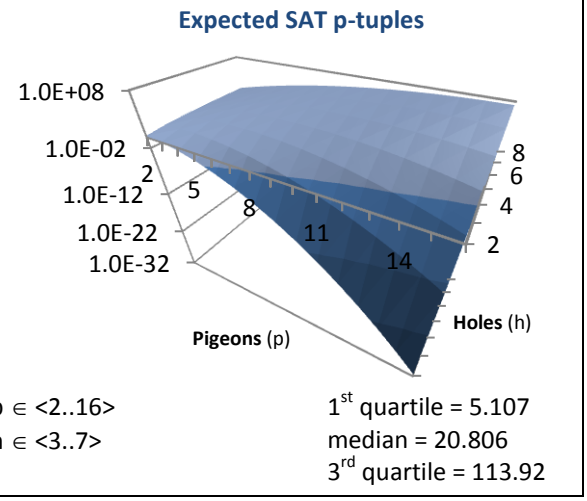
Proposition 3 ***(Limit P/H Characteristics).*** *The probability of satisfiability of a random p-tuple of values π in graph interpretation of P/H principle converges to* 0 *for $h \to \infty$; that is,* $\lim_{h\to\infty}(1-\frac{1}{h})^{\binom{h+1}{2}} = 0$. *The expected number of satisfied p-tuples of values ε in P/H principle is $\mathcal{O}(e^{(h+1)(-\frac{1}{2}+\ln h)})$ which is $\mathcal{O}(\left(\frac{h}{\sqrt{e}}\right)^{(h+1)})$ and blows up to $+\infty$ for*

$h \to \infty$; *that is,* $\lim_{h\to\infty} h^{h+1}(1-\frac{1}{h})^{\binom{h+1}{2}} = +\infty$. ■

We will generalize the P/H principle so that there will be strictly less holes than pigeons but not necessarily one fewer. Generalized P/H principle is unsatisfiable as well. A sample of probabilistic characteristics of the model of generalized P/H principle is shown in Table 2.

Table 2. Expected number of satisfied tuples of values in generalized P/H principle.

	Expected number of satisfied p-tuples $\varepsilon = h^p(1-\frac{1}{h})^{\binom{p}{2}}$		
	Number of **holes** h		
Number of **pigeons** p	***3***	***4***	***5***
2	6.0	12.0	20.0
3	8.0	27.0	64.0
4	7.111	45.563	163.84
5	4.213	57.665	335.544
6	1.664	54.737	549.756
7	0.438	38.968	720.576



p ∈ <2..16>
h ∈ <3..7>

1st quartile = 5.107
median = 20.806
3rd quartile = 113.92

Our aim is to select $(k+2)$-tuples of CSP variables for B2C-consistency in the explicit literal encoding $E^1(\Phi) = (X^1_\Phi, D^1_\Phi, C^1_\Phi)$ which has similar probabilistic characteristics that are exhibited by the model of (generalized) P/H principle. This selection is supposed to ensure required properties – that is, the similar level of difficulty as P/H principle and the similar constraint setup. The following incremental mechanism for selecting the next variable based on estimating probabilistic characteristics from the currently selected variables will be used.

The requirement which is specified as the part of input together with k is the interval for expected number of satisfied $(k+2)$-tuples of values. Let ε_L and ε_U be the lower and upper bound for this interval respectively. The first CSP variable into the

$(k+2)$-tuple is supposed to be selected by some specific process (randomly or systematically; actually a systematic process is used within the experimental implementation). Other CSP variables are selected incrementally; suppose that $K_+^2 = [\bar{\Gamma}_{i_0}, \bar{\Gamma}_{i_1}, \bar{\Gamma}_{i_2}, \dots, \bar{\Gamma}_{i_\kappa}]$ is a tuple of the already selected CSP variables (if $\kappa = k$ then the process is finished). Let $\bar{\Gamma}_{i_{\kappa+1}}$ be a candidate CSP variable. The expected number of satisfied $(k+2)$-tuples with $\bar{\Gamma}_{i_{\kappa+1}}$ denoted as $\varepsilon(\bar{\Gamma}_{i_{\kappa+1}})$ is estimated as follows: let $\rho(\bar{\Gamma}_{i_{\kappa+1}})$ be the constraint density among variables from the set $\bigcup K_+^2 \cup \{\bar{\Gamma}_{i_{\kappa+1}}\}$ (already selected variables together with the new candidate) then $\varepsilon(\bar{\Gamma}_{i_{\kappa+1}}) = \left(\sqrt[\kappa+1]{\prod_{l=0}^{\kappa+1} \left| D(\bar{\Gamma}_{i_l}) \right|} \right)^{k+2} \left(1 - \rho(\bar{\Gamma}_{i_{\kappa+1}})\right)^{\binom{k+2}{2}}$. That is, the product of sizes of domains of the final $(k+2)$-tuple is estimated as $(k+2)$th power of geometric mean of sizes of domain of already selected variables. The constraint density is supposed to be preserved for final $(k+2)$-tuple. If $\varepsilon_L \leq \varepsilon(\bar{\Gamma}_{i_{\kappa+1}}) \leq \varepsilon_U$ then $\bar{\Gamma}_{i_{\kappa+1}}$ may be used as the next CSP variable for the $(k+2)$-tuple. If there are multiple variables satisfying this condition any of them may be selected. The whole process of selection CSP variables for B2C-consistency is formalized as Algorithm 3.

Proposition 4 ***(Selection Time/Space Complexity).*** *The algorithm for selecting CSP variables can be implemented to have the worst case time complexity of* $\mathcal{O}(k^2 |X_\Phi^1| |\mathbb{D}|^2)$. *The space of* $\mathcal{O}(k + |X_\Phi^1|)$ *is needed in addition to the space necessary for storing CSP* $E^1(\Phi)$. ■

Commentary: Each new CSP variable is selected for the resulting tuple out of at most $|X_\Phi^1|$ CSP variables for which estimation of the expected number of satisfied $(k+2)$-tuples must be calculated. Calculating this estimation with respect to a single variable consumes $\mathcal{O}(k|\mathbb{D}|^2)$ steps as it is necessary calculate constraint density relatively to

all the already selected variables. A new variable is included exactly k times.

Additional space is needed for storing probabilistic characteristics for CSP variables which consumes the space of $\mathcal{O}(|X_\Phi^1|)$. The space of $\mathcal{O}(k)$ is needed to store the resulting tuple of CSP variables. ■

Algorithm 3. *Process of selection a suitable* $(k+2)$*-tuple of CSP variables.* Variables are heuristically selected to prefer resulting expected number of satisfied $(k+2)$-tuples of values in the interval of $\langle \varepsilon_L, \varepsilon_U \rangle$ or near to this interval from below or above.

---

function *Select-CSP-Variables*$(k, \bar{\Gamma}_{i_0}, E^1(\Phi) = (X_\Phi^1, D_\Phi^1, C_\Phi^1), \varepsilon_L, \varepsilon_U)$: **tuple**

/* Parameters: k - size of the tuple of CSP variables,
$\bar{\Gamma}_{i_0}$ - the first CSP variable,
$E^1(\Phi)$ - explicit literal encoding,
$\varepsilon_L, \varepsilon_U$ - lower and upper bounds for expected number of satisfied $(k+2)$-tuples of values. */

1: **for** $\kappa = 0,1,\dots,k$ **do**
2: **for each** $\bar{\Gamma}_{i_{\kappa+1}} \in X_\Phi^1$ **do**
3: **let** $\rho(\bar{\Gamma}_{i_{\kappa+1}})$ is the constraint density in $\bigcup[\bar{\Gamma}_{i_0}, \bar{\Gamma}_{i_1}, \bar{\Gamma}_{i_2}, \dots, \bar{\Gamma}_{i_{\kappa+1}}]$
4: $\varepsilon(\bar{\Gamma}_{i_{\kappa+1}}) \leftarrow \left(\sqrt[\kappa+1]{\prod_{l=0}^{\kappa+1} \left| D(\bar{\Gamma}_{i_l}) \right|} \right)^{k+2} \left(1 - \rho(\bar{\Gamma}_{i_{\kappa+1}})\right)^{\binom{k+2}{2}}$
/* the following let form assigns $\perp$ if undefined */
5: **let** $\bar{\Gamma}_\alpha \in X_\Phi^1$ such that $\varepsilon(\bar{\Gamma}_\alpha) \leq \varepsilon_L \wedge (\forall \bar{\Gamma}_l \in X_\Phi^1)\varepsilon(\bar{\Gamma}_\alpha) \geq \varepsilon(\bar{\Gamma}_l)$
6: **let** $\bar{\Gamma}_\beta \in X_\Phi^1$ such that $\varepsilon(\bar{\Gamma}_\beta) \geq \varepsilon_U \wedge (\forall \bar{\Gamma}_l \in X_\Phi^1)\varepsilon(\bar{\Gamma}_\beta) \leq \varepsilon(\bar{\Gamma}_l)$
7: **let** $\bar{\Gamma}_\mu \in X_\Phi^1$ such that $\varepsilon_L \leq \varepsilon(\bar{\Gamma}_\mu) \leq \varepsilon_U$
8: **if** $\bar{\Gamma}_\mu \neq \perp$ **then**
9: $i_\kappa \leftarrow \mu$
10: **else**
11: **if** $\bar{\Gamma}_\alpha = \perp$ **then**
12: $i_\kappa \leftarrow \beta$
13: **else**
14: **if** $|\varepsilon(\bar{\Gamma}_\alpha) - \varepsilon_L| < \left|\varepsilon(\bar{\Gamma}_\beta) - \varepsilon_U\right|$ **then**
15: $i_\kappa \leftarrow \alpha$
16: **else**
17: $i_\kappa \leftarrow \beta$
18: **return** $[\bar{\Gamma}_{i_0}, \bar{\Gamma}_{i_1}, \bar{\Gamma}_{i_2}, \dots, \bar{\Gamma}_{i_\kappa}]$

It is infeasible in larger SAT instances to compute and to store constraint density between all the pairs of variables on the current commodity hardware as there are too many such pairs (notice that there may be more than $1.0\mathrm{E}+6$ clauses in large SAT in-

stances which makes more than $\binom{1.0\mathrm{E}+6}{2} \approx 1.0\mathrm{E}+12$ pairs of variables; that would require approximately several terabytes of memory). Hence it is necessary to compute constraint density on demand.

SAT Preprocessing with B2C-Consistency

An experimental SAT preprocessing tool based on B2C-consistency called `prepro-cessSIGMA` [25] was implemented C++ in order to conduct an experimental evaluation and to provide proof of the concept. To achieve the best inference strength of preprocessing, $(k+2)$-tuples are selected according to the theory in the previous section so that the expected number of satisfied tuples of values belongs into the interval typical for the model of generalized P/H principle. We select k uniformly from the interval $\langle 2..10 \rangle$ as it experimentally proved to be computationally manageable in reasonable time.

In typical SAT instances arity of clauses ranges from 2 to 10 [15] while the most common are small clauses with arities 3, 4, and 5 - domain sizes in the corresponding literal encodings are exactly the same. The expected number of satisfied tuples of values for a setup of P/H principle with corresponding $p \in \langle 2..10 \rangle$ and $h \in \langle 3..5 \rangle$ belongs into the interval $\langle 0.001, 755.579 \rangle$ while the 1st quartile, median, and 3rd quartile equal to 5.107, 20.806, 113.92 respectively. Taking into account that we are preferring non-existence of satisfied tuple of values, it is advisable to select the preferred interval for expected number of satisfied tuples of values $\langle \varepsilon_L, \varepsilon_U \rangle$ with ε_L low below the median and slightly below the 1st quartile and ε_U slightly above the median. A preliminary experimental evaluation with SAT instances containing mainly small clauses showed that the best setting is $\langle \varepsilon_L, \varepsilon_U \rangle = \langle 3.0, 32.0 \rangle$ which perfectly correlates with the above probabilistic estimations.

As the computation of B2C-consistency is a time consuming operation it is done only for certain number of tuples of variables. More precisely small formulae with less than or equal to 2048 variables has allowed 16 times the number variables B2C-consistency checks. Large formulae (that is, those with more than 2048 variables) are allowed 4 times square root of the number of variables B2C-consistency checks (currently, there is no smooth transition between these two rates as it was not necessary to be implemented for experimental evaluation). In both cases the number of steps of the search for a consistent k-tuple was bounded by the constant $\Lambda = 4096$. This setup was manually tailored during the development.

5 Experimental Evaluation

The experimental evaluation of our prototype SAT preprocessing `preprocessSIGMA` was concentrated on discovering the benefit of B2C-consistency in the context of other existent preprocessing techniques and on evaluation of internal properties of the experimental implementation. It also should provide a justification for the theory we have discussed earlier.

Basic Competitive Experimental Evaluation

The experimental implementation of B2C-consistency within our prototype preprocessing tool `preprocessSIGMA` has been competitively evaluated with respect to the most prominent existing tools for SAT preprocessing. Particularly the following preprocessing tools have been chosen: `LiVer` [20], `NiVer` [20], `HyPre` [5], and `Shatter` [2]. As the reference SAT solver `MiniSat` version 2.2 [12] with build in `SatElite` pre-processing step has been used.

`LiVer` and `NiVer` use resolution based variable elimination for preprocessing; `LiVer` allows a bounded increase in the total number of literals in the resulting formula while `NiVer` does not allow any increase of this number. The `HyPre` preprocessing tool is based on binary hyper-resolution and equivalence reasoning. `Shatter` represents a tool most akin to our `preprocessSIGMA` as it exploits certain kind of global reasoning as well. Symmetries in the input formula are detected and symmetry breaking clauses are added by `Shatter` into the output formula. To detect symmetries, *graph isomorphism* problem [26] needs to be solved during the preprocessing process.

The experimental evaluation has been done with a set of 185 relatively difficult SAT instances (mixture of satisfiable and unsatisfiable) selected from *Satisfiability Library* (SATLib) [15] and from *SAT Competition 2002/2003*. Relatively difficult means here that it takes a SAT solver a considerable effort to solve the instance relatively to its size. The complete set of instances selected for tests can be found at the website: http://ktiml.mff.cuni.cz/~surynek/research/j-preprocess-2011. This website contains also experimental data in the raw form and all the source code necessary to reproduce all the presented experiments.

Several characteristics were measured during the evaluation process. The most informative characteristic is the number of *conflicts* that occurred during the process of solving. The conflict can be regarded as a dead-end in the backtracking based search process. The number of conflicts has been measured for the original instances and for instances processed by individual SAT pre-processors from our test suite. The number of conflict corresponds well with the overall runtime. The real runtime[1] has been measured as well to obtain the complete picture of performance of all the SAT pre-processors.

[1] All the test were run on a machine with Intel Xeon 2.0GHz CPU, 12 GiB of RAM, under Ubuntu Linux version 8.04, Kernel 2.6.24-19 SMP.

Table 3. Results for the *fraction* of the set of testing instances used in our experimental evaluation. The number of conflicts `MiniSat 2.2` has encountered on the original instances and on those pre-processed with `HyPre`, `LiVer`, `Niver`, `Shatter`, and our `preprocessSIGMA` are shown. The best performing pre-processors on each instance are depicted in bold (`MiniSat` was set to the timeout of 512 seconds).

Conflicts	*Variables*	*Clauses*	*C/V Ratio*	*HyPre*	*Original*	*LiVer*	*NiVer*	*Shatter*	*sigma*	*SAT/UNSAT*
bart12.shuffled	*180*	*820*	4.555	212	**105**	118	118	606	**105**	SAT
bart14.shuffled	*195*	*905*	4.641	402	104	**100**	**100**	112	104	SAT
bart16.shuffled	*210*	*990*	4.714	106	**103**	**103**	**103**	105	**103**	SAT
bart20.shuffled	*270*	*1476*	5.466	127	121	**103**	**103**	607	121	SAT
ca004.shuffled	*80*	*168*	2.1	**29**	43	32	**29**	42	43	UNSAT
ca008.shuffled	*130*	*370*	2.846	117	145	**102**	150	151	145	UNSAT
ca016.shuffled	*272*	*780*	2.867	**293**	449	416	326	357	433	UNSAT
ca032.shuffled	*558*	*1606*	2.878	752	943	739	**657**	901	790	UNSAT
difp_19_99_arr_rcr	*1201*	*6563*	5.464	141649	209417	58305	304092	209417	**92754**	SAT
difp_19_99_wal_rcr	*1775*	*10446*	5.885	**31031**	134284	108681	158235	91397	15245	SAT
difp_21_1_arr_rcr	*1453*	*7967*	5.483	63546	191884	538426	427292	191884	**45453**	SAT
difp_21_99_arr_rcr	*1453*	*7967*	5.483	97408	190663	249983	350142	190663	**35704**	SAT
dp04u03.shuffled	*1017*	*2411*	2.370	**26**	70	72	63	89	61	UNSAT
dp05s05.shuffled	*1885*	*4818*	2.555	138	90	116	100	347	**46**	SAT
ezfact32_6.shuffled	*769*	*4777*	6.211	33088	422	32957	32957	422	**209**	SAT
ezfact32_7.shuffled	*769*	*4777*	6.211	29574	5744	46659	46659	5744	**836**	SAT
ezfact32_9.shuffled	*769*	*4777*	6.211	47191	1181	64056	64056	1181	**160**	SAT
ezfact32_10.shuffled	*769*	*4777*	6.211	1988	1990	22500	22500	1990	**448**	SAT
fpga10_11_uns_rcr	*220*	*1122*	5.1	8315862	4935017	4866421	4866421	11509	**2**	UNSAT
fpga10_12_uns_rcr	*240*	*1344*	5.6	7219129	7209341	7183640	7218248	8889	**1**	UNSAT
fpga10_13_uns_rcr	*260*	*1586*	6.1	6511919	6466487	6497147	6497268	12521	**1**	UNSAT
fpga10_15_uns_rcr	*300*	*2130*	7.1	5401469	5390760	5387934	5405715	8517	**1**	UNSAT
fpga10_8_sat	*120*	*448*	3.733	163	201	201	201	**100**	201	SAT
fpga10_9_sat	*135*	*549*	4.066	168	202	202	202	**61**	202	SAT
fpga12_11_sat	*198*	*968*	4.888	405	200	200	200	**69**	200	SAT
fpga12_12_sat	*216*	*1128*	5.222	102	208	208	208	**77**	208	SAT
homer06.shuffled	*180*	*830*	4.611	209811	272019	258487	258487	1085	**1**	UNSAT
homer10.shuffled	*360*	*3460*	9.611	502279	641132	464639	464639	1070	**2**	UNSAT
homer16.shuffled	*264*	*1476*	5.590	6527180	6525641	6766937	6682636	28720	**3**	UNSAT
homer20.shuffled	*440*	*4220*	9.590	3249156	3230156	3265756	3207038	12290	**2**	UNSAT
lisa19_0_a.shuffled	*1201*	*6563*	5.464	117828	235824	381242	108878	235824	**15534**	SAT
lisa19_1_a.shuffled	*1201*	*6563*	5.464	439709	445563	**208567**	528589	445563	320076	SAT
lisa21_1_a.shuffled	*1453*	*7967*	5.483	121498	328846	**4841**	309122	328846	93629	SAT
med11.shuffled	*341*	*5556*	16.293	197	**41**	101	101	**41**	**41**	SAT
med17.shuffled	*782*	*18616*	23.805	151	**106**	808	808	**106**	**106**	SAT
qg1-7.shuffled	*686*	*6816*	9.935	115	**49**	67	67	242	**49**	SAT
term1_gr_2pin_w3.shuffled	*746*	*3517*	4.714	69	52	21	124	56	**9**	UNSAT
term1_gr_rcs_w3.shuffled	*606*	*2518*	4.155	7	7	7	7	10	**1**	UNSAT

A small fraction of the set of instances (38 out of 185) used for experimental evaluation together with their characteristics and results regarding number of conflicts after preprocessing is shown in Table 3. In all the tests presented in this manuscript, `MiniSat` was set to the timeout of 512 seconds.

The full competitive comparison of the number of conflicts that `MiniSAT` encountered during solving the original instances and pre-processed instances is shown in Figure 3.

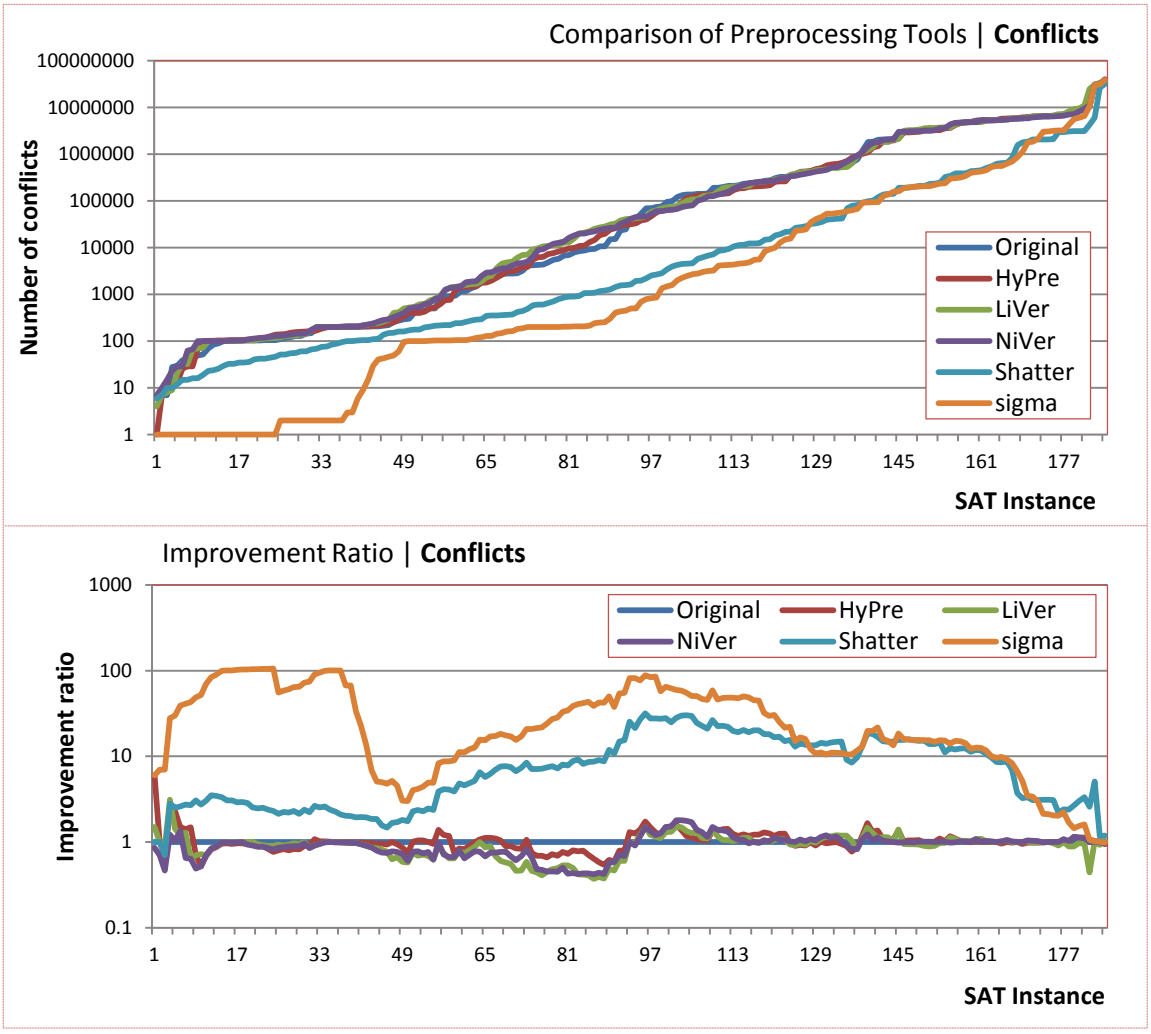


Figure 3. *Competitive comparison of preprocessing tools (conflicts).* The number of conflicts that occurred during solving the original and pre-processed SAT instances with `MiniSat 2.2` and the improvement ratio are shown (instances are sorted for each preprocessor to get non-descending sequences in the upper part). Our `preprocessSIGMA` is compared with `HyPre`, `LiVer`, `NiVer`, and `Shatter` on a set of selected difficult SAT instances. It can be observed that `HyPre`, `LiVer`, and `NiVer` have almost no positive effect contrary to `preprocessSIGMA` and `Shatter` which both deliver significant improvement.

It can be observed that preprocessors solely relying on simplification through resolution and hyper-resolution – that is `HyPre`, `LiVer`, and `Niver` – deliver almost no improvement. On instances of medium difficulty even worsening appears. The significant benefit of pre-processing is delivered by `Shatter` as well as by `preprocessSIGMA` that both exploit global reasoning. On instances of easy to medium difficulty

preprocessSIGMA delivers better positive effect in pre-processing than Shatter – up to 100 times less conflicts are encountered in instances pre-processed with preprocessSIGMA than in the original ones. The difference between preprocessSIGMA and Shatter diminishes on instances of top difficulty (Shatter becomes marginally better on several instances).

The results however should not be interpreted as that pre-processing by resolution/hyper-resolution is useless. On simpler instances it is typically more beneficial [4] if we take into account tradeoff between the benefit and computational costs. Moreover, we need to consider that the version of MiniSat we used has its own built-in preprocessor SatElite. The results may hence show that simple resolution-based preprocessing is not enough to outperform the benefit of using SatElite (however this claim may require more investigation).

An experimental evaluation regarding the runtime is shown in Figure 4. It can be observed that if solving runtime only is measured, then the picture is almost the same is in the case of conflicts – preprocessSIGMA and Shatter clearly outperforms the others (HyPre, LiVer, and Niver). The situation changes if the time for pre-processing is accounted (that is, total runtime = pre-processing runtime + solving runtime is took into account). Here preprocessSIGMA becomes lagging behind all others on easier instances due to its long runtime.

The similar phenomenon but not that profound can be observed for Shatter which loses against HyPre, LiVer, and NiVer on easier instances. The situation changes on harder instances where Shatter and preprocessSIGMA perform better than the others. Even preprocessSIGMA matches Shatter on yet harder instances.

If the total runtime for the whole testing suite is considered than we get an interesting comparison: both Shatter and preprocessSIGMA save up to 36% of the

total runtime compared to the situation without pre-processing while the other tools (`HyPre`, `LiVer`, and `Niver`) provide no or marginal improvement only.

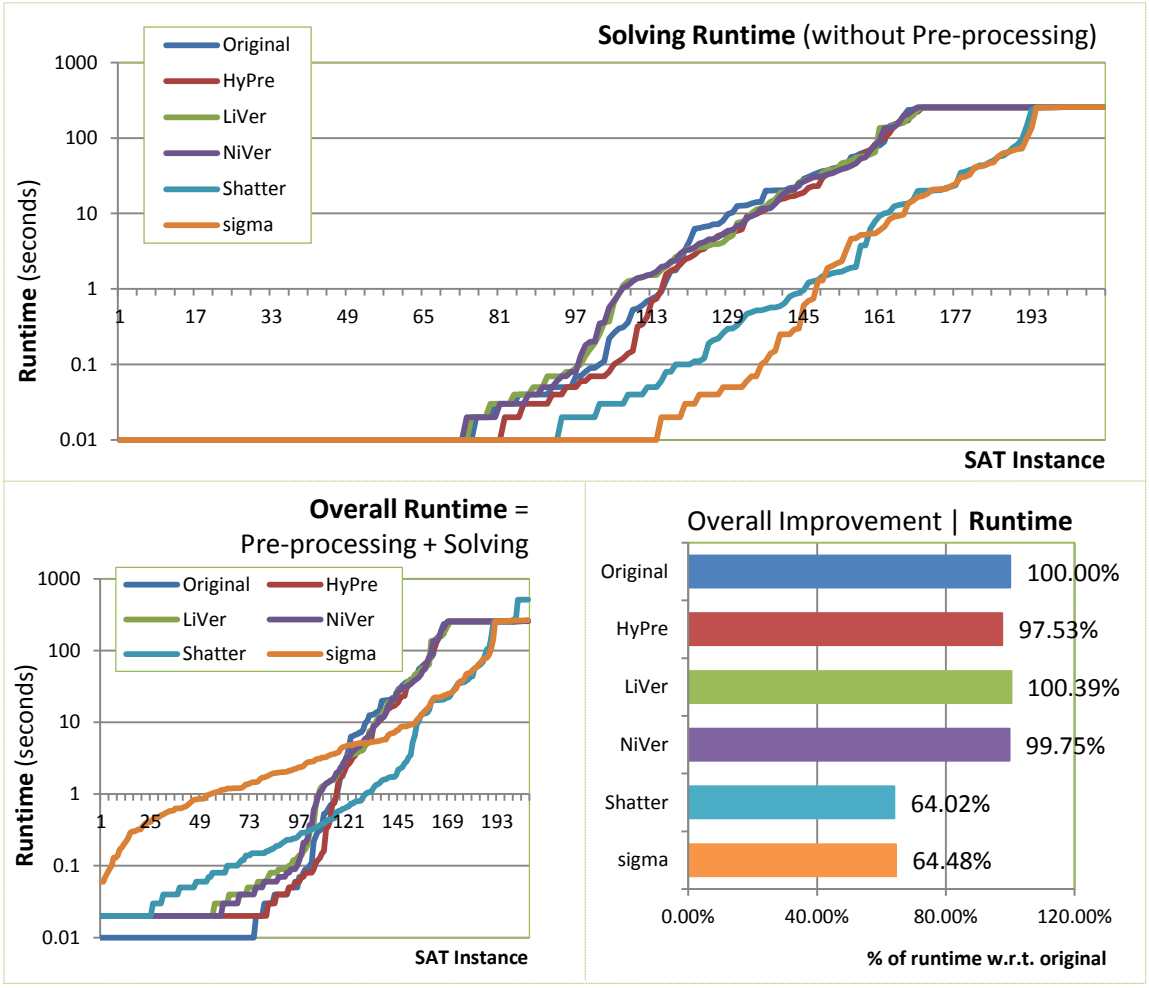


Figure 4. *Competitive comparison of preprocessing tools (runtime).* SAT instances are sorted in the same order as in Figure 3. Both `preprocessSIGMA` and `Shatter` are far ahead the other tools in terms of solving runtime improvement (upper part) while `preprocessSIGMA` tends to provide better outputs for easier instances. If preprocessing time is accounted then `preprocessSIGMA` loses on easy instances[2] and matches `Shatter` on harder ones while other tools getting to be outperformed as well. If the total runtime for all the tested instances is accounted (pre-processing runtime + solving runtime) then both `Shatter` and `preprocessSIGMA` save up to 36% compared to the situation without pre-processing while the other tools provide no or marginal improvement only.

[2] This is mainly due to not well optimized implementation of `preprocessSIGMA`.

Notice, that the match in overall runtime with `Shatter` on harder instances has been achieved despite the not well optimized implementation of `preprocessSIGMA`. Regarding the pre-processing time with `preprocessSIGMA` there is a great room for further improvement.

B2C-Consistency on Integer Factorization

Especially good performance was exhibited by our preprocessing tool based on B2C-consistency on instances encoding *integer factorization problem* [3] (satisfiable instances). The first observation made on these instances is that B2C-consistency is able to make many inferences of inconsistent pairs of values that can be ruled out in the pre-processed instance afterwards. The additional experimental evaluation showed that the more inconsistent pairs of values are inferred the greater the reduction of the number of conflicts (as well as runtime) can be achieved on the resulting instance. This property however goes against the requirement of bounding the number of B2C-consistency checks which is needed to be low to preserve reasonable time consumption (if we want to infer as many as possible inconsistent pairs of values we should perform as many as possible consistency checks). Hence, there is still room for improvement on integer factorization problems using fine tuning the parameters of B2C-consistency such as the allowed number of constraint checks.

The competitive results regarding integer factorization problem are shown in Figure 5. Clearly `preprocessSIGMA` is the best for almost all the instances in terms of the number of conflicts that it can save. Regarding the relative improvement it happen much less frequently with `preprocessSIGMA` than with other tools that the pre-processed instance is worse than the original one. If we look at the total number of con-

flicts over all the instances in the sub-set, `preprocessSIGMA` can deliver a saving of 36.09% which is far the best among all the tested pre-processing tools.

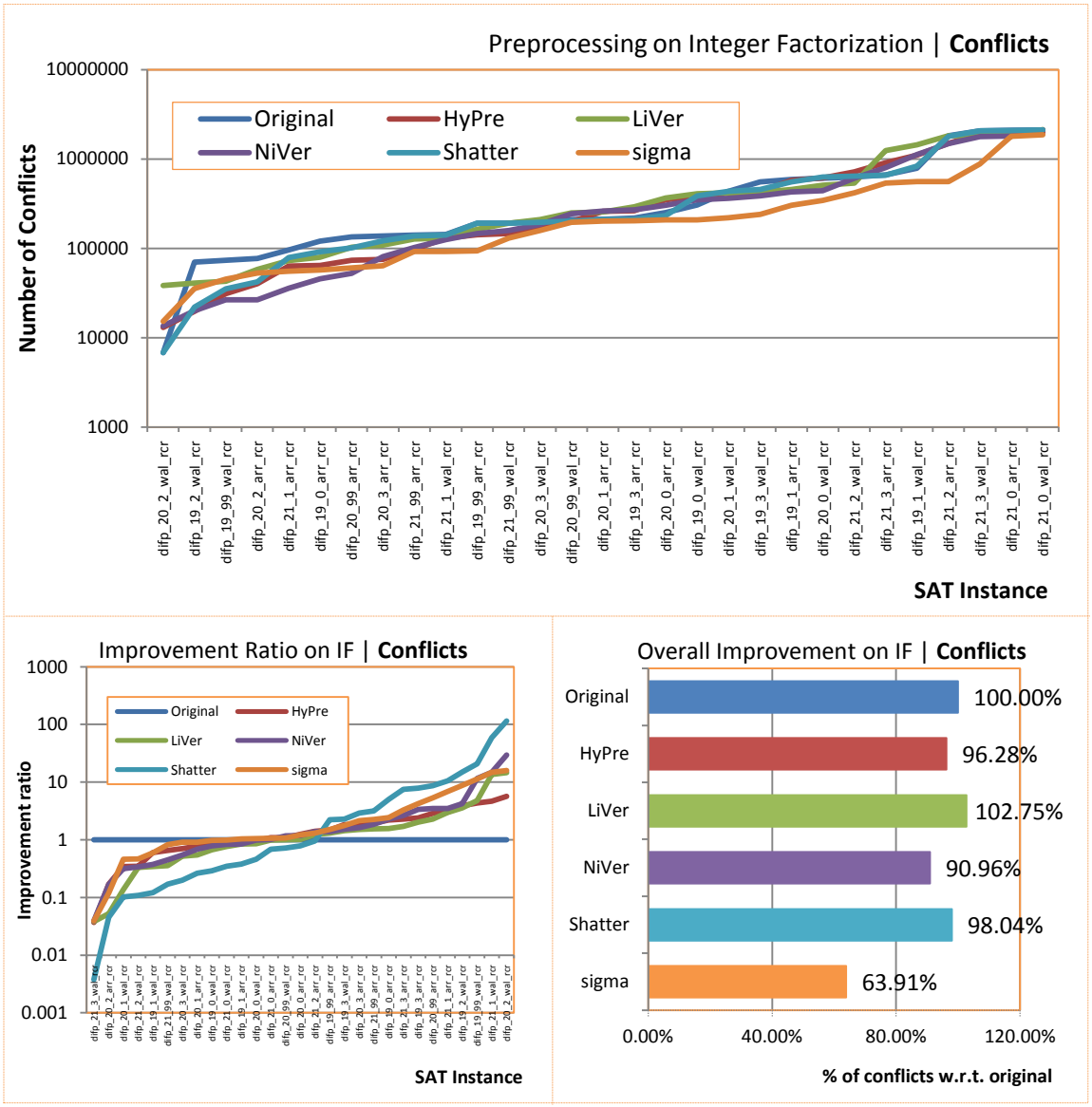


Figure 5. *Competitive evaluation on instances encoding integer factorization problem.* The first part shows absolute number of conflicts (logarithmic scale) `MiniSat 2.2` has encountered on instances encoding integer factorization [3] after pre-processing by tested SAT pre-processors. The lower two charts show relative improvement that can be achieved by using tested SAT pre-processors in terms of the number of conflicts. Notice that our `preprocessSIGMA` can save up to $100.00 - 63.91 = 36.09\%$ of the overall number of conflicts that occur during solving all the original instances which is the best of all the tested pre-processing tools.

The surprising result was obtained for `Shatter` which was unexpectedly outperformed by both `HyPre` and `Niver`. Results regarding the runtime are almost the same as for the number of conflicts so they are not presented.

Experimental Evaluation of the Process of Selection of Variables

The last part of experiments was devoted to an evaluation of the selection of variables for consistency checks. This evaluation is important for verification whether all the internal processes of B2C-consistency work as it was expected. This aspect concerns mainly the selection of a tuple of variables for the consistency check.

The expected number of satisfied tuples of values over the variables selected by Algorithm 3 with the setup of $\langle \varepsilon_L, \varepsilon_U \rangle = \langle 3.0{,}32.0 \rangle$ over all the consistency checks and all the testing instances has the following probabilistic characteristics – minimum, first quartile, median, third quartile, maximum equal to 2.131, 14.899, 27.562, 130.149, and 1141710.567 respectively. The more detailed insight into the distribution of the expected number of satisfied tuples of values over selected variables is provided by a partial histogram shown in Figure 6.

According to these results we can conclude that Algorithm 3 selects variables according to the specified bounds ε_L and ε_U sufficiently well. The most of the selections of tuples of variables have the expected number of satisfied tuples of values within the interval $\langle \varepsilon_L, \varepsilon_U \rangle = \langle 3.0{,}32.0 \rangle$ as it was originally required.

As the variable selection algorithm is trying to push the expected number of satisfied tuples towards lower bound ε_L and the upper bound ε_U from below and from above respectively the distribution tends to concentrate around these bounds (the histogram has peaks in the bounds). It may be an interesting research question for future

work how the performance of B2C-consistency can be influenced if the distribution has a different shape.

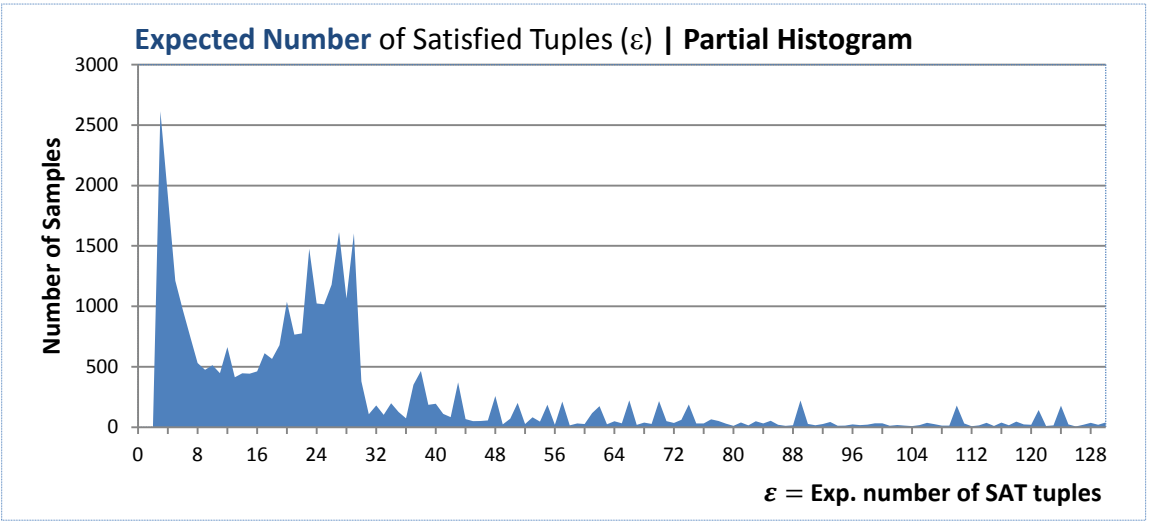


Figure 6. *Partial histogram of expected number of satisfied tuples (ε).* The histogram characterizes the selection of variables made by Algorithm 3 over all the testing SAT instances and all the B2C-Consistency checks. Only the part up to the 3rd quartile is shown. It can be observed that the most of the selections of tuples of variables have the expected number of satisfied tuples of values within the interval $\langle \varepsilon_L, \varepsilon_U \rangle = \langle 3.0, 32.0 \rangle$ as it was required.

Summary of Experimental Evaluation

If we summarize results of the experimental evaluation we can state that B2C-consistency with the proposed process for selection of variables represent a powerful technique that can be used as a basis of a SAT preprocessing tool. Our experimental evaluation proved that prototype pre-processing tool `preprocessSIGMA` based on B2C-consistency is fully competitive with respect to the existent prominent SAT preprocessing tools in terms of the saving of the number of conflicts as well as in terms of the overall runtime. The competitiveness in terms of runtime was achieved despite the not well optimized implementation of the prototype.

Especially good performance was exhibited by `preprocessSIGMA` on instances encoding integer factorization problems where there is still room for fine tuning the parameters of B2C-consistency to achieve yet better performance.

The evaluation of the internal characteristics of our prototype pre-processing tool – namely evaluation of the process of selection of the tuple of variables for consistency check – indicates a good match with theoretical expectations.

6 Conclusion and Future Work

We presented in this manuscript a new type of consistency called B2C-consistency (bounded $(2,k)$-consistency) for use in Boolean satisfiability (SAT). The new consistency is inspired by both global constraints and local consistency. Basically it is $(2,k)$-consistency with the bounded number of search steps for proving inconsistency enriched by reasoning over complete sub-graphs of pair-wise conflicting literals. Complete sub-graphs reasoning brings a global aspect into proving inconsistency and can improve the consistency enforcing process significantly especially on SAT instances encoding well known P/H principle (pigeon/hole principle) and similar which are known to be difficult for the standard solving process based on search.

The whole design of new consistency is explained in the context of modeling SAT as a constraint satisfaction problem (CSP) using so called explicit literal encoding (that is, literal encoding with explicit clauses obtained by singleton unit propagation).

Next we investigated probabilistic properties of a so called generalized P/H principle – particularly the expected number of satisfied (consistent) tuples of values with respect to a tuple of selected variables for consistency check. The investigation showed that certain distribution of the expected number of satisfied tuples is characteristic for P/H principle where many inconsistent tuples of values can be found. Therefore

we proposed a process for selection of variables which is trying to select variables so that the corresponding expected number of satisfied tuples of variables has the similar probabilistic distribution as in the case of P/H principle. Using this process we are trying to identify hard sub-problems (such a P/H principle) that can be yet resolved by B2C-consistency.

To evaluate our proposal we implemented B2C-consistency and the process of selection of variables within a prototype SAT pre-processing tool `preprocessSIGMA`. The experiments confirmed that B2C-consistency and variable selection process are beneficial and that we are able to select variables for consistency checks with similar probabilistic characteristics as in the case of generalized P/H principle. The competitive evaluation on a set of 185 SAT instances (mixture of satisfiable and unsatisfiable) showed that `preprocessSIGMA` delivers better results than existent pre-processing tools `HyPre`, `LiVer`, and `Niver` which are based on local reasoning and comparable results to `Shatter` which is based on symmetry breaking. On instances encoding integer factorization problem `preprocessSIGMA` performed as far the best of all the tested pre-processing tools. Moreover, `preprocessSIGMA` has some advantages with respect to the comparable `Shatter`. It is easier to implement – in `Shatter`, graph isomorphism which is a difficult problem itself needs to be solved – and it has many parameters that can be further fine tuned. Notice, that we achieved competitive performance despite the not well optimized implementation of `preprocessSIGMA`.

There are several interesting question for future work. At the present time we used characterization of the distribution of the expected number of satisfied tuples of values with two parameters – the lower and upper bound. It would be interesting to use more parameters to control the shape of the resulting distribution over all the consistency checks more precisely.

Another interesting investigation may be done with repeated used B2C-consistency. Consider a pre-processed instance which is pre-process once more. This approach is unfortunately impractical at the current state of the implementation as the setup of pre-processing is relatively time consuming and to preserve relatively acceptable competitiveness we cannot afford to run it more than once. However, more efficient implementation may change the situation.

Acknowledgement

This work is partially supported by The Czech Science Foundation (Grantová agentura České republiky - GAČR) under the contract number 201/09/P318 and by The Ministry of Education, Youth and Sports, Czech Republic (Ministerstvo školství, mládeže a tělovýchovy ČR – MŠMT ČR) under the contract number MSM 0021620838 and partially also by Japan Society for the Promotion of Science (JSPS) within the post-doctoral fellowship of the author (reference number P11743).

References

1. Aloul, F. A., Ramani, A., Markov, I. L., Sakallah, K. A.: *Solving Difficult SAT Instances in the Presence of Symmetry*. Proceedings of the 39th Design Automation Conference (DAC 2002), pp. 731-736, USA, ACM Press, 2002, http://www.aloul.net/benchmarks.html, [March 2011].
2. Aloul, F. Markov, I. L., Sakallah, K.: Shatter: *Efficient Symmetry-Breaking for Boolean Satisfiability*. Proceedings of the Design Automation Conference (DAC 2003), pp. 836-839, ACM Press, 2003, http://www.aloul.net/Tools/shatter/, [July 2011].
3. Aloul, F. A.: *SAT Benchmarks, Difficult Integer Factorization Problems.* Research web page, http://www.aloul.net/benchmarks.html, [March 2011].

4. Anbulagan, Slaney, J.: *Multiple Preprocessing for Systematic SAT Solvers*. Proceedings of The 6th International Workshop on the Implementation of Logics, CEUR-WS.org, 2006.
5. Bacchus, F., Winter, J.: *Effective Preprocessing with Hyper-Resolution and Equality Reduction.* Proceedings of the Theory and Applications of Satisfiability Testing, 6th International Conference, (SAT 2003), pp. 341-355, LNCS 2919, Springer 2004, http://www.cs.toronto.edu/~fbacchus/sat.html, [July 2011].
6. Bessière, C., Hebrard, E., Walsh, T: *Local Consistencies in SAT.* Proceedings of the Theory and Applications of Satisfiability Testing, 6th International Conference (SAT 2003), pp. 299-314, LNCS 2919, Springer, 2004.
7. Biere, A., Heule, M., van Maaren, H., Walsh, T.: *Handbook of Satisfiability*. IOS Press, 2009.
8. Bomze, I. M., Budinich, M., Pardalos, P. M., Pelillo, M.: *The maximum clique problem, Handbook of Combinatorial Optimization.* Kluwer Academic Publishers, 1999.
9. Cook, S. A.: *The Complexity of Theorem Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971), pp. 151-158, ACM Press, 1971.
10. Dechter, R.: *Constraint Processing.* Morgan Kaufmann Publishers, 2003.
11. Dowling, W., Gallier, J.: *Linear-time algorithms for testing the satisfiability of propositional Horn formulae.* Journal of Logic Programming, Volume 1 (3), 267-284, Elsevier Science Publishers, 1984.
12. Eén, N., Sörensson, N.: *An Extensible SAT-solver.* Proceedings of Theory and Applications of Satisfiability Testing, 6th International Conference (SAT 2003), pp. 502-518, LNCS 2919, Springer 2004, http://minisat.se/MiniSat.html, [July 2011].
13. Freuder, E. C.: *A Sufficient Condition for Backtrack-Free Search.* Journal of the ACM, Volume 29, pp. 24-32, ACM Press, 1982.
14. Golumbic, M. C.: *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.
15. Holger, H. H., Stützle, T.: *SATLIB: An Online Resource for Research on SAT.* Proceedings of Theory and Applications of Satisfiability Testing, 4th International Conference (SAT 2000), pp.283-292, IOS Press, 2000, http://www.satlib.org, [March 2011].

16. Jackson, P., Sheridan, D.: *Clause Form Conversions for Boolean Circuits.* Theory and Applications of Satisfiability Testing, 7th International Conference (SAT 2004), Revised Selected Papers, pp. 183–198, Lecture Notes in Computer Science 3542, Springer 2005.
17. Papadimitriou, C.: *Computational Complexity.* Addison Wesley, 1994.
18. Petke, J., Jeavons, P.: *Local Consistency and SAT-Solvers.* Proceedings of the Principles and Practice of Constraint Programming - 16th International Conference (CP 2010), pp. 398-413, Lecture Notes in Computer Science 6308, Springer 2010.
19. Seidel, R.: *On the Complexity of Achieving K-Consistency*. Technical Report, University of British Columbia Vancouver, BC, Canada,1983.
20. Subbarayan, S., Pradhan, D., K.: *NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT Instances*. Proceedings of The 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), pp. 276-291, LNCS 3542, Springer 2005, http://www.itu.dk/people/sathi/niver.html, [July 2011].
21. Surynek, P.: *Solving Difficult SAT Instances Using Greedy Clique Decomposition.* Proceedings of the 7th Symposium on Abstraction, Reformulation, and Approximation (SARA 2007), LNAI 4612, pp. 359-374, Springer, 2007.
22. Surynek, P.: *Making Path Consistency Stronger for SAT*. Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2008), ISTC-CNR, 2008.
23. Surynek, P.: *An Adaptation of Path Consistency for Boolean Satisfiability: a Theoretical View of the Concept.* Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming, 2010 (CSCLP 2010), pp. 16-30, Fraunhofer FIRST, 2010.
24. Surynek, P.: *Between Path-Consistency and Higher Order Consistencies in Boolean Satisfiability*. Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming, 2011 (CSCLP 2011), York, United Kingdom, pp. 120-134, University of York, 2011.
25. Surynek, P.: *sigmaSAT / preprocessSIGMA - a collection of experimental SAT processing tools.* Research web page, Charles University in Prague, 2011, http://ktiml.mff.cuni.cz/~surynek/index.html.php?select=software&product=sigmasat, [July 2011].

26. Torán, J.: *The Hardness of Graph Isomorphism.* SIAM Journal on Computing, , pp. 1093-1108, volume 33, number 5, SIAM, 2004.
27. Walsh, T.: *SAT vs. CSP*. Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, 441-456, LNCS 1894, Springer Verlag, 2000.