Between Path-Consistency and Higher Order Consistencies in Boolean Satisfiability

Pavel Surynek[*]

Charles University in Prague
Faculty of Mathematics and Physics
Department of Theoretical Computer Science and Mathematical Logic
Malostranské náměstí 25, Praha, 118 00, Czech Republic
pavel.surynek@mff.cuni.cz

Abstract. The task of enforcing certain level of consistency in Boolean satisfiability problem (SAT problem) is addressed in this paper. A recently developed concept of modified variant of path-consistency (PC) is recalled and experimentally evaluated. The modified PC combines the standard PC on the literal encoding of the given SAT instance with global properties calculated from constraints imposed by the instance – namely with the maximum number of visits of a certain set by the path being checked to exist. An experimental evaluation discovered that modified PC performs well on difficult SAT instances with structured constraint graph. However, the performance was not so good on other classes of SAT instances. Therefore the possible limits of improvements of modified PC were investigated by evaluating (2,*k*)-consistency which represent the strongest possible variant of modified PC in the given context. This evaluation showed that not only there are SAT instances for which it worth to study improvements of modified PC (namely they are represented by the difficult integer factorization problems) but also that (2,*k*)-consistency consistency itself can be used as an efficient preprocessing tool.

Keywords: local consistency, global consistency, path-consistency, (2,*k*)-consistency, CSP, SAT

1 Introduction and Motivation

A method for increasing the inference strength of *path-consistency* (PC) [14, 15] with regard on applications in Boolean satisfiability (SAT) [6] called *modified PC* [20] is revisited in this paper. The work on modified PC is still in progress and this paper should be regarded as a next step in this progress.

Applying consistencies in SAT has been already studied in [3]. The concept of modified PC combines the standard PC on the literal encoding model [21] of the SAT instance [6] with global properties calculated from constraints forming the instance.

[*] This work is 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.

The existence of a path in a graph interpretation of the instance which is normally being checked within PC is further restricted by additional requirements. Regarding the inference strength, modified PC lies between $(2,k)$-consistency [8] and standard PC (see Fig. 1 for illustration). The goal is to tune modified PC so that it will eventually represents a good trade-off between PC and $(2,k)$-consistency in terms of strength and computational cost. The ultimate goal of whole design of modified PC is a tool for preprocessing SAT instances.

In this paper, we investigate positive and negative properties of the current version of modified PC and assess its possible improvements by evaluating $(2,k)$-consistency. Surprisingly, this evaluation showed that $(2,k)$-consistency itself can be successfully used as a SAT preprocessing tool. The paper has two parts; in the first part modified PC is recalled; in the second part experimental evaluation is presented and its implications are discussed.

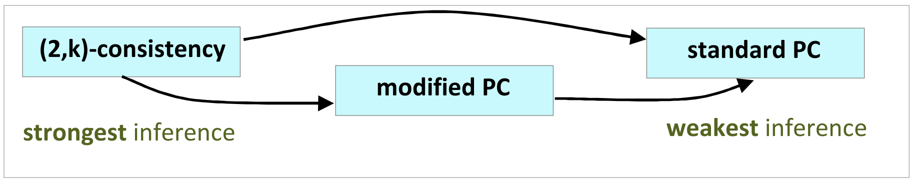


Fig. 1. *The concept modified PC in the context of other consistencies.* The inference strength is depicted using arrows. The goal is to shift modified PC towards (2,*k*)-consistency.

2 Notations and Definitions

Concepts of *constraint satisfaction problem* (CSP) [8] and *Boolean satisfiability* (SAT) [6] need to be established first to make reasoning about PC in the context of SAT easier to understand.

Definition 1 (*Constraint satisfaction problem - CSP*). Let $\mathbb{D}$ be a finite set representing *domain universe*. A *constraint satisfaction problem* [8] is a triple (X, C, D) where X is a finite set of *variables*, C is a finite set of *constraints*, and $D: X \longrightarrow \mathcal{P}(\mathbb{D})$ is a function that defines *domains* of individual variables from X (that is, $D(x) \subseteq \mathbb{D}$ is a set of values that can be assigned to the variable $x \in X$). Each constraint from $c \in C$ is of the form $\langle (x_1^c, x_2^c, \ldots, x_{K^c}^c), R^c \rangle$ where $K^c \in \mathbb{N}$ is called an arity of the constraint c, the tuple $(x_1^c, x_2^c, \ldots, x_{K^c}^c)$ with $x_i^c \in X$ for $i = 1,2, \ldots, K^c$ is called a *scope* of the constraint, and the relation $R^c \subseteq D(x_1^c) \times D(x_2^c) \times \ldots \times D(x_{K^c}^c)$ defines the set of tuples of values for that the constraint c is satisfied. The task is to find a valuation of variables $v: X \longrightarrow \mathbb{D}$ such that $v(x) \in D(x)$ $\forall x \in X$ and $(v(x_1^c), v(x_2^c), \ldots, v(x_{K^c}^c)) \in R^c$ $\forall c \in C$. □

A constraint $c \in C$ with the scope $(x_1^c, x_2^c, \ldots, x_{K^c}^c)$ will be denoted as $c(\{x_1^c, x_2^c, \ldots, x_{K^c}^c\})$; this notation is useful when the ordering of variables in the scope is not known from the context; when ordering of variables in the scope matters, then a notation $c(x_1^c, x_2^c, \ldots, x_{K^c}^c)$ will be used instead.

A CSP is called *binary* if all the constraints has the arity of two. The expressive power of a binary CSP is not reduced in comparison with a general one since every

CSP can be transformed into an equivalent binary CSP [17]. The key concept of *path-consistency* (PC) [15] that is addressed in this paper is defined for binary CSPs only. It is also convenient to suppose, that each pair of variables is constrained by at most one constraint.

Definition 2 (*Boolean satisfiability problem - SAT*). Let B be a finite set of *Boolean variables*; that is, a set of variables that can be assigned either $FALSE$ or $TRUE$. A Boolean formula F over the set of variables B in a so called *conjunctive normal form* (*CNF*) [13] is the construct of the form $\bigwedge_{i=1}^{N}(\bigvee_{j=1}^{K_i} l_j^i)$ where l_j^i with either $l_j^i = y$ or $l_j^i = \neg y$ for some $y \in B$ for $i = 1,2,\dots,N$; $j = 1,2,\dots,K_i$ is called a *literal* and $(\bigvee_{j=1}^{K_i} l_j^i)$ for $i = 1,2,\dots,N$ is called a *clause*. The task is to find a valuation of Boolean variables $b: B \longrightarrow \{FALSE, TRUE\}$ such that F evaluates to $TRUE$ under b while $\neg$ (*negation*), $\vee$ (*disjunction*), and $\wedge$ (*conjunction*) are interpreted commonly in the Boolean algebra. A formula for that such a satisfying valuation exists is called *satisfiable*. □

2 Path-consistency in CSP

The standard definition of *path-consistency* in CSP will be recalled before the augmented versions and their relaxations are introduced. The following definition refers to general paths of variables which is not necessary in fact. However, this style of definition will be more suitable for making intended augmentations.

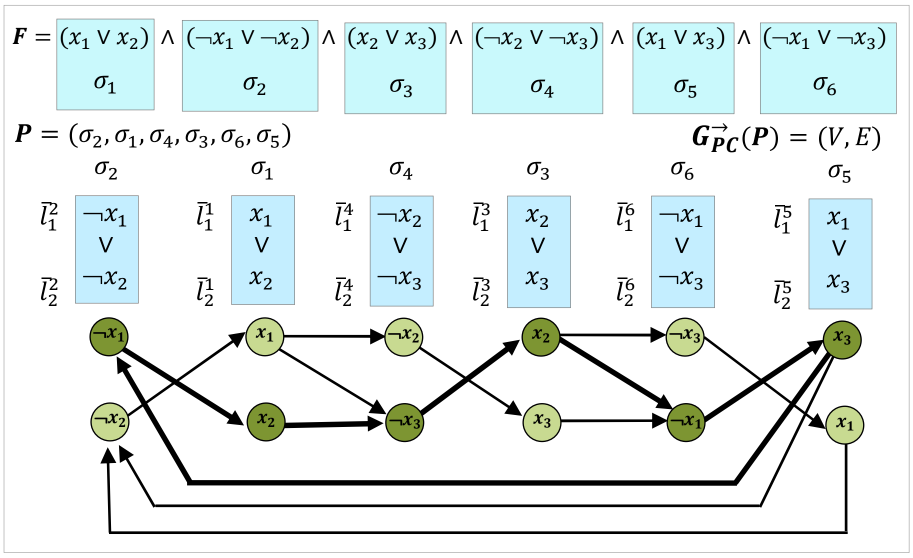
Definition 3 (*Path-consistency - PC*). Let (X, C, D) be a binary CSP and let $P = (x_0, x_1, \dots, x_K)$ with $x_i \in X$ for $i = 0,1,\dots,K$ be a sequence of variables called a *path*. A pair of values $d_0 \in D(x_0)$ and $d_K \in D(x_K)$ is *path-consistent* with respect to P if there exists a valuation $v: \{x_0, x_1, \dots, x_K\} \longrightarrow \mathbb{D}$ with $v(x_0) = d_0 \wedge v(x_K) = d_K$ such that constraints $c(\{x_i, x_{(i+1) \bmod K}\})$ are satisfied by v for every $i = 0,1,\dots,K$. The path P is said to be *path-consistent* if all the pairs of values from $D(x_0)$ and $D(x_K)$ respectively are path-consistent with respect to P. Finally, the CSP (X, C, D) is said to be *path-consistent* if it is path-consistent for every path. □

Notice that variables forming the path in the definition do not need to be necessarily distinct. Although the notion of PC seems to be computationally infeasible since there are typically too many paths, it is sufficient to check PC for all the paths consisting of triples of variables only to ensure that the given CSP is path-consistent [14, 15]. In other words, although it seems that PC captures the problem globally (a path can go through large portion of variables of the instance), it merely defines a local property.

There exist many algorithms for enforcing PC in a CSP such as PC-4 [11] and PC-6 [1, 4]. They differ in the representation of auxiliary data structures and the efficiency. The common feature of PC algorithms is however the process how the consistency is enforced. It is done by eliminating pairs of inconsistent values until a path-consistent state is reached (the smallest set of pairs of values such that their elimination makes the problem path-consistent is being pursued). The process of elimination of pairs of values is typically done by pruning extensional representation of constraints (lists of allowed tuples) to forbid more pairs of values.

3 Standard Path-Consistency in SAT

The aim of this work is to modify PC to make it applicable on SAT and to increase its inference strength by incorporating certain global reasoning into it. The easier task is to make PC applicable on SAT - it is sufficient to model SAT as CSP. A so called *literal encoding* [21], which of the result is a binary CSP, is particularly used. This kind of encoding is especially suitable since it allows natural expressing of PC in terms of graph constructs.



$F = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$
σ_1 σ_2 σ_3 σ_4 σ_5 σ_6
$P = (\sigma_2, \sigma_1, \sigma_4, \sigma_3, \sigma_6, \sigma_5)$
$G_{\vec{PC}}(P) = (V, E)$
$\bar{l}_1^2$ $\bar{l}_2^2$ $\bar{l}_1^1$ $\bar{l}_2^1$ $\bar{l}_1^4$ $\bar{l}_2^4$ $\bar{l}_1^3$ $\bar{l}_2^3$ $\bar{l}_1^6$ $\bar{l}_2^6$ $\bar{l}_1^5$ $\bar{l}_2^5$

Fig. 2. *An illustration of path-consistency in the CSP model of a SAT problem.* The SAT problem represented by a formula F shown here is a representation of the requirement of selecting an odd number of variables from every of the following sets to be true: $\{x_1, x_2\}$, $\{x_1, x_3\}$, $\{x_2, x_3\}$. Observe, that there is no satisfying valuation of F. However, a pair of literals $\neg x_1$ and x_3 from the left most variable and from the right most variable respectively are path-consistent with respect to a depicted path P since they are non-conflicting and there exists a path from the left to the right consisting of edges between neighboring variables connecting allowed pairs of values (the path is marked by bold edges and by darker vertices).

Let $F = \bigwedge_{i=1}^{N}(\bigvee_{j=1}^{K_i} l_j^i)$ be a Boolean formula in CNF over a set of Boolean variables B. Let $\mathbb{D} = \bigcup_{i=1}^{n}(\bigcup_{j=1}^{k_i}\{\bar{l}_j^i\})$ be a domain universe; that is, a constant symbol with the stripe is introduced into $\mathbb{D}$ for each literal occurrence in F (notice that, each occurrence of a literal corresponds to a different constant symbol). The corresponding CSP (X, C, D) using literal encoding is built as follows: $X = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$; that is, a variable is introduced for each clause of F; it holds for $D: X \longrightarrow \mathcal{P}(\mathbb{D})$ that $D(\sigma_i) = \bigcup_{j=1}^{K_i}\{\bar{l}_j^i\}$; that is, the domain of an i-th clause contains constant symbols corresponding to all its literals. A constraint $c(\{\sigma_{i_1}, \sigma_{i_2}\}) = \langle(\sigma_{i_1}, \sigma_{i_2}), R^c\rangle$ is introduced over every pair of variables with $i_1, i_2 \in \{1,2,\ldots,N\} \wedge i_1 \neq i_2$ where a variable $x \in B$ such that either $x \in D(\sigma_{i_1}) \wedge \neg x \in D(\sigma_{i_2})$ or $\neg x \in D(\sigma_{i_1}) \wedge x \in D(\sigma_{i_2})$ exists. Such a constraint $c(\{\sigma_{i_1}, \sigma_{i_2}\})$ then forbids every tuple of values $(\bar{l}_{j_1}^{i_1}, \bar{l}_{j_2}^{i_2})$ such that there exists $x \in B$ for that either $\vec{l}^{i_1}_{j_1} = x \wedge \vec{l}^{i_2}_{j_2} = \neg x$ or $\vec{l}^{i_1}_{j_1} = \neg x \wedge \vec{l}^{i_2}_{j_2} = x$ (that is, the tuple $(\vec{l}^{i_1}_{j_1}, \vec{l}^{i_2}_{j_2})$ is removed from R^c which has been initially set to $D(\sigma_{i_1}) \times D(\sigma_{i_2})$). A solution of the resulting CSP (X, C, D) corresponds to the valuation of Boolean variables of B that satisfies F and vice versa [21].

Having the CSP model of SAT it is possible to check PC for the corresponding CSP model and proclaim the original SAT path-consistent or path-inconsistent accordingly. If elements of variable domains are interpreted as vertices and allowed tuples of values as directed edges connecting them, then PC with respect to a given path can be interpreted as existence of paths in the resulting directed graph. More precisely, let $P = (\sigma_{i_0}, \sigma_{i_1}, \dots, \sigma_{i_K})$ with $i_j \in \{1,2,\dots,N\}$ for $j = 0,1,\dots,K$ be a sequence of variables in the literal encoding CSP model (X, C, D). A directed graph $G_{\vec{PC}}(P) = (V, E)$, in which PC can be interpreted as the existence of paths, is defined as follows: $V = \bigcup_{j=0}^{K} D(\sigma_{i_j})$ and if $(\vec{l}^{i_j}_{j_1}, \vec{l}^{i_{(j+1) \bmod K}}_{j_2}) \in R^{c(\sigma_{i_j}, \sigma_{i_{(j+1) \bmod K}})}$ then a directed edge $(\vec{l}^{i_j}_{j_1}, \vec{l}^{i_{(j+1) \bmod K}}_{j_2})$ is included into E. A pair of values $\vec{l}^{i_0}_{j_1} \in D(\sigma_{i_0})$ and $\vec{l}^{i_K}_{j_2} \in D(\sigma_{i_K})$ is path-consistent with respect to the path P if there is an edge $(\vec{l}^{i_K}_{j_1}, \vec{l}^{i_0}_{j_1})$ in $G_{\vec{PC}}(P)$ and there exists a path from the vertex $\vec{l}^{i_0}_{j_1}$ to the vertex $\vec{l}^{i_K}_{j_2}$ in $G_{\vec{PC}}(P)$. The graph $G_{\vec{PC}}(P)$ will be called a *graph interpretation* of PC – see Fig. 2 for illustration.

Notice that PC is *incomplete* in the sense that a pair of values may be path-consistent even if there is no solution of the problem that contains this pair of values (see Fig. 2 again). Analogically, the problem may be path-consistent (that is, path-consistent with respect to all the paths) even if it has no solution actually. The partial reason for this weakness of PC is that many constraints are ignored when a pair of values is checked. This is especially apparent if a longer path of variables is considered. Only constraints over pairs of variables neighboring in the path are considered while many constraints such as that for example over the first and the third variable in the path are ignored. This property is disadvantageous especially in SAT where stronger reasoning is typically more beneficial.

For further augmentation of PC, it is also convenient to prepare a so called *auxiliary constraint graph* for the model with respect to the path P that reflects all the constraints over the variables of the path P. It is an undirected graph $G_{CSP}(P) = (V, E)$ and it is defined as follows: $V = \bigcup_{j=0}^{K} D(\sigma_{i_j})$; an edge $\{\vec{l}^{i_j}_{j_1}, \vec{l}^{i_k}_{j_2}\}$ is added to E if $(\vec{l}^{i_j}_{j_1}, \vec{l}^{i_k}_{j_2}) \notin R^{c(\sigma_{i_j}, \sigma_{i_k})}$; and all the edges $\{\vec{l}^{i_j}_{j_1}, \vec{l}^{i_j}_{j_2}\}$ for all $j = 1,2,\dots,N$ and $j_1, j_2 = 1,2,\dots,K_j \wedge j_1 \neq j_2$. Observe that the auxiliary constraint graph subsumes complement of the graph interpretation with respect to the same path.

4 Making Path-Consistency Stronger

A modification of PC has been proposed to overcome mentioned limitations of the standard version. To increase inference strength of PC additional requirements on the path in the graph model are imposed. These additional requirements reflect constraints over non-neighboring variables in the path of variables. As the auxiliary constraint graph represents an explicit representation of constraints, it is exploited for determining additional requirements.

An approach adopted in this work restricts the size of the intersection of the constructed path with certain subsets of vertices in the graph interpretation of PC. More precisely, let $G_{\vec{PC}}(P) = (V, E)$ be a graph interpretation of PC in a CSP model of SAT (X, C, D). The set of vertices V is partitioned into disjoint sequences $L_1, L_2, \ldots, L_M$ called *layers* (that is, $\bigcup_{i=1}^{M} \hat{L}_i = V$ and $\hat{L}_i \cap \hat{L}_j$ $\forall i, j \in \{1,2,\ldots,M\} \wedge i \neq j$; where denotes the union of the sequence $\hat{A}$, that is $\hat{A} = \bigcup_{i=1}^{n}\{a_i\}$ for $A = [a_1, a_2, \ldots, a_n]$). The maximum size of the intersection of the path being checked to exist with individual layers is determined using the set of constraints C (notice that all the constraints over P are considered – not only constraints over neighboring variables in P). This proposal will be called an *initial augmentation* of PC in the rest of the text.

The concept of the initial augmentation of PC comes from [19]. The process of decomposition of the set of vertices into layers is done over the corresponding auxiliary constraint graph $G_{CSP}(P)$. Vertices of $G_{CSP}(P)$ are decomposed into vertex disjoint complete subgraphs. The knowledge of such decomposition can be then used to partition vertices into layers that directly correspond to found complete subgraphs. However, determining a complete subgraph is a difficult task itself. Hence a greedy approach has been used to obtain an acceptable solution.

Since it is possible to assign to a variable at most one value from values corresponding to vertices of the complete subgraph in $G_{CSP}(P)$, the maximum size of the intersection of the path with a layer is thus at most 1. Notice, that at most one value from vertices corresponding to the domain of a variable can be selected (this is due to the presence of the complete subgraph over the set of vertices corresponding to the domain of a variable in $G_{CSP}(P)$). Notice further, that if a value corresponding to a vertex in a complete subgraph is selected than all the values corresponding to other vertices of the complete subgraph are ruled out since they are in conflict with the selected value with respect to constraints.

A quite negative result has been obtained in [19]. It has been shown that finding a path, which conforms to the calculated maximum size of the intersection with individual layers, corresponds to finding a Hamiltonian path [5]. This is known to be an *NP*-hard problem. Hence, it is not tractable to find a path that satisfies defined requirements exactly. Moreover, initial experiments showed that it is almost impossible to make any reasonable relaxation of proposed requirements. Every relaxation of requirements on the path being constructed proposed by the author leads to weakening the modified PC down to the level of the standard version of PC (specifically, several adaptations of the algorithm for finding single source shortest paths [7] have been evaluated by the author).

These initial findings founded an effort to further augment requirements on the constructed path in order to allow developing stronger and more efficient relaxations. The result of this effort is a concept of a so called *modified version of PC*.

4.1 A Modified Version of Path-Consistency

Again, partitioning of vertices of $G_{\vec{PC}}(P)$ into layers is supposed. In addition, the sequencing of variables in the path P is exploited for defining the maximum size of the intersection of the constructed path with layers. Particularly, the path being constructed is required to conform to the calculated maximum size of the intersection with vertices of the layer preceding a given vertex of the path with respect to the sequencing of variables in P. The maximum size of the intersection is again imposed by the set of constraints C. More precisely, let $L_1, L_2, \dots, L_M$ be layers of $G_{\overrightarrow{PC}}(P)$; let a function $\chi: V \longrightarrow \mathbb{N}$ defines requirements on the maximum size of intersections imposed by constraints as follows: $\chi(v_j^l)$ is the maximum size of the intersection of the constructed path with a set of vertices $\{v_0^l, v_1^l, \dots v_i^l\}$ where $L_l = [v_0^l, v_1^l, \dots, v_{K^l}^l]$ with $l \in \{1,2,\dots,M\}$ and $j \in \{0,1,\dots,K^l\}$. Let a consistency defined by this new requirement on the constructed path be called a *modified PC*. Observe that this new concept is a generalization of the initial augmentation described above (see Fig 3 for illustration). It is intractable to construct a path conforming to the maximum sizes of intersections determined by χ as in the case of the initial augmentation. Nevertheless, it is possible to make a tractable relaxation of these requirements which does not collapse down to the level of the standard PC.

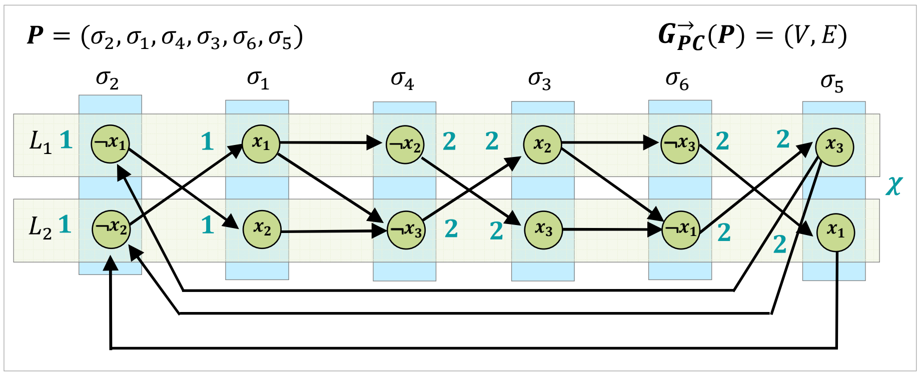


Fig 3. *An illustration of modified path-consistency in the CSP model of a SAT problem.* The maximum size of the intersection of the constructed path with vertices preceding the given vertex (including) in its layer is calculated using constraints for each vertex - these maximum sizes are denoted as the function χ. For example, having $\chi(\bar{l}_1^3) = 2$ then the constructed path can intersect the subset of vertices $\{\bar{l}_1^2, \bar{l}_1^1, \bar{l}_1^4, \bar{l}_1^3\}$ (first occurrences of literals in first four variables of the path P) of the layer L_1 in at most two vertices. Observe, that these requirements on the path being constructed rules out its existence for connecting a pair of vertices $\bar{l}_1^2$ from the left most variable (occurrence of literal $\neg x_1$) and $\bar{l}_1^5$ from the right most variable (occurrence of literal x_3). Compare it with the standard PC in Fig. 2 where the corresponding path connecting the same pair of vertices exists.

Let us now briefly describe such a tractable relaxation. Suppose that χ is already known (the process of calculation of χ will be described in the following section). Let $d_0 \in D(\sigma_{i_0})$ and $d_K \in D(\sigma_{i_K})$ be a pair of values for that a consistency is being checked. Two assignments will be maintained: $\Sigma: V \longrightarrow \mathbb{N}_0$ and $\psi: V \longrightarrow \mathbb{N}_0{}^{M\times(K+1)}$ where $\mathbb{N}_0{}^{M\times(K+1)}$ denotes matrices of the size $M \times (K+1)$ over $\mathbb{N}_0$. The assignment Σ will express the total number of distinct paths in $G_{\overrightarrow{PC}}(P) = (V, E)$ starting in d_0 and ending in a given vertex. Observe, that it is easy to calculate $\Sigma(v)$. It is determined recursively by the expression: $\Sigma(v) = \sum_{u \in V, (u,v) \in E} \Sigma(u)$, while $\Sigma(d_0) = 1$. The assignment ψ expresses statistical information about paths in $G_{\overrightarrow{PC}}(P)$ starting in d_0 and ending in a given vertex regarding the size of the intersection with layers. More precisely, an element of $\psi(v)$ at i-th row and j-th column (that is, $\psi(v)_{i,j}$) with $v \in V$, $i \in \{1,2,\dots,M\}$, and $j \in \{0,1,\dots,K\}$ represents the number of distinct paths starting in

d_0 and ending in v intersecting with the layer L_i in exactly j vertices that conform to relaxed requirements (that is, the size of the intersection of these paths with L_i is j). If the mentioned conformation to relaxed requirements is omitted, the information maintained in ψ is not difficult to be calculated recursively for every vertex of $G_{\vec{PC}}(P)$. However, as it is algorithmically more complex calculation, it is deferred to the section devoted to algorithms.

Requirements on the size of the intersection of the constructed path with layers represented by χ are relaxed in the following way. If it is detected that all the paths staring in d_0 and ending in v intersects the layer containing v in more vertices than it is allowed by χ, then it is possible to conclude that there is no path connecting d_0 and v that conforms to calculated maximum sizes of intersections with layers. Hence, v is unreachable from d_0 under given circumstances. The described relaxation can be expressed using defined assignments Σ and ψ. Let L_{l^v} be a layer containing v (that is, $v \in \hat{L}_{l^v}$). If there is some $j > \chi(v)$ such that $\psi(v)_{l^v,j} = \Sigma(v)$, then there is no path connecting d_0 and v conforming to the maximum sizes of intersection with layers. Observe, that although there is no $j > \chi(v)$ such that $\psi(v)_{l^v,j} = \Sigma(v)$, the required path still need not to exist. This is the principle which is called the relaxation in the context of this paper.

If it is detected that there is no path connecting d_0 and d_K that conforms to relaxed requirements on the maximum sizes of intersections with layers, the pair of values d_0 and d_K is said to be *inconsistent* with respect to the *modified PC*.

The detailed pseudo-code of greedy algorithms for determining layer decomposition and calculating maximum sizes of intersection is given in [20]. The pseudo-code of consistency enforcing algorithm also given there.

5 Augmenting Modified Path-Consistency to (2,k)-Consistency

Modified PC is trying to exploit more information from the constraints with scope containing a variable from the selected sequence with respect to which the consistency is checked. A question arises how strong the consistency would be if all the information from involved constraints is used; that is, if all the tuples of values forbidden by constraints are taken into account when a path connecting a pair of values is constructed. Such a consistency check corresponds to the concept of $(2,k)$-consistency [8] which is formalized in the following definition.

Definition 4 (($2,k$)-*consistency*). Let (X, C, D) be a binary CSP and let $K = \{x_0, x_1, \ldots, x_k, x_{k+1}\}$ with $x_i \in X$ for $i = 0,1,\ldots,k+1$ be a set of distinct variables. A pair of values $d_0 \in D(x_0)$ and $d_{k+1} \in D(x_{k+1})$ is said to be consistent with respect to $(2,k)$-consistency and the set of variables K if there exists a valuation $v\colon \{x_0, x_1, \ldots, x_k, x_{k+1}\} \longrightarrow \mathbb{D}$ with $v(x_0) = d_0 \wedge v(x_{k+1}) = d_k$ such that all the constraints $c(\{y, z\})$ such that $y, z \in K$ are satisfied by v. The CSP (X, C, D) is said to be $(2,k)$-consistent if it is $(2,k)$-consistent with respect to all the sets of size $k+2$ and all the pairs of values. □

Intuitively, $(2,k)$-consistency checks for a pair of values from the domains of distinct variables whether there exists a k-tuple of values from other variables such that all these values form a consistent assignment; that is, a supporting k-tuple of consistent values is searched for a given pair of values (see Fig 4). The relation to (modified) PC is that enforcing $(2,k)$-consistency subsumes enforcing (modified) PC along paths of length $k+2$. Notice that $(2,k)$-consistency represents the limit of the possible strength of any variant of modified PC. This fact has a consequence that if $(2,k)$-consistency eventually turn out to be unsuccessful when applied on SAT it is then useless to deal with the modified PC and make any further improvements of it. Fortunately, a part of the experimental evaluation showed that this is not the case.

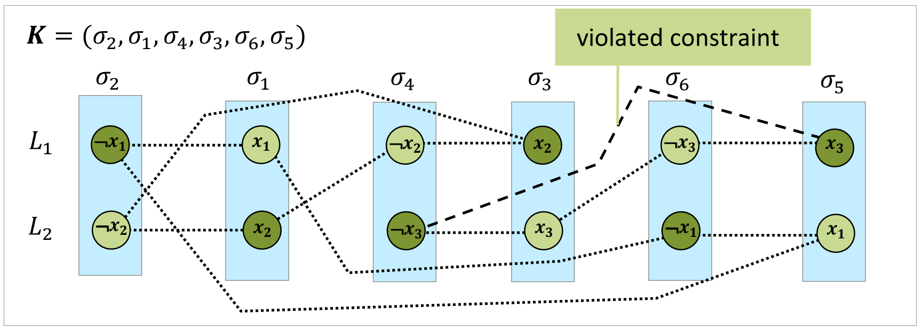


Fig 4. *An illustration of* $(2,k)$*-consistency for* $k=4$ *in the CSP model of a SAT problem.* Forbidden pairs of values are depicted using dotted edges. The $(2,k)$-consistency can detect that a pair of literals $\neg x_1$ and x_3 is inconsistent since there is no set supporting consistent 4-tuple of values in σ_1, σ_2, σ_3, and σ_6. Notice that the path justifying standard PC of $\neg x_1$ and x_3 in Fig. 2 is forbidden here.

The application of higher order consistencies in SAT has been already studied. It has been shown in [16] that k-consistency can be simulated by clause learning mechanism within conflict-directed SAT solvers on a so called *direct encoding* of the given CSP instance. It is not known whether this is also true for $(2,k)$-consistency. The positive answer would also limit the effect of enforcing modified PC on situations where simulating the consistency by clause learning mechanism is inefficient. This is partially the reason why the modified PC is targeted on instances where the standard conflict-directed SAT solvers are inefficient.

6 Preliminary Experimental Evaluation

We have performed a preliminary experimental evaluation of the above suggestions. The evaluation was targeted on several aspects of the proposed concepts.

First, we investigated the quality of layer decomposition on standard SAT benchmarks. This is important for assessing the potential of modified PC. Next, we tried to identify a class of SAT instances where the described variant of the modified PC is beneficial when used as a preprocessing tool. The result of this evaluation is quite negative; it has shown that the current variant of modified PC can positively affect the performance of a SAT solver only on classical hard instances from [1]. These instances can be however solved by other techniques as well [1, 18].

The last aspect we have addressed in the evaluation is a question whether there exists a class of SAT instances where the modified PC after some improvements may be beneficial. The identification of such a class was done by applying $(2, k)$-consistency as a preprocessing tool. If $(2, k)$-consistency is successful on some SAT instance then there is a chance that an improvement of the modified PC may be also successful there. This experiment has shown that such a promising class of instances is represented by encodings of so called *difficult integer factorization* [2].

6.1 Experimental Implementation and Setup

There are several important issues regarding the implementation of the described method and the experimental setup. Consistencies which were tested as a preprocessing tool were first applied on a given instance for which they can infer new binary clauses. An augmented instance with additional binary clauses was then submitted to a SAT solver which eventually gave an answer to the instance. Several characteristics such as the number of inferred clauses, the number of decisions, and runtime were measured along the process. For our experiments, the Minisat2 SAT solver [10] was used.

To improve inference strength of preprocessing the *singleton unit propagation* has been performed first [9] (that is, each literal is assigned the value $TRUE$ and unit propagation is performed; assignments to literals enforced by the propagation together with assignment to the original literal form additional constraints). It can discover some binary clauses that can be used in the literal encoding of the SAT instance as implied constraints. These implied constraints subsequently reduce the size of the intersection with layers in modified PC.

As it is computationally too expensive to enforce modified PC as well as $(2, k)$-consistency with respect to all paths and subsets of variables respectively only some promising paths and subsets of variables were used. The length of the sequence of variables varied from 4 to 8. For each length of the sequence of variables and for each clause of the instance, the most promising sequence of variables starting in the given clause was determined and consistency was calculated with respect to it. That is, the existence of a path or a consistent k-tuple was checked for all the pairs of values from the first and the last variable of the selected sequence.

When constructing the most promising sequence of variables/clauses, a clauses that is most constrained with respect to the currently last clause was added with the probability of 0.7 otherwise a randomly selected clause has been added.

Although the given consistency was computed in a partial manner using the above approach, it represents a suitable trade-off between the number of inferred clauses and the effort to compute them.

All the tested algorithms were implemented in C++. The tests were run on an dual AMD Opteron 1600 MHz, with 1GB RAM under Mandriva Linux 10.1, 32-bit edition; gcc 3.4.3 with optimization level –o3 was used for compilation.

6.2 Tests of Modified Path-Consistency

The first part of the evaluation of modified PC is devoted to getting a visual insight of the consistency on standard satisfiability benchmarks from SATLib [12].

Table 1. *Maximum intersection sizes with the first layer of the layer decomposition*. The intersection sizes are calculated in the graph interpretation of several satisfiability instances from SATLib using the greedy algorithm described in [20].

SAT instance	Maximum **intersection** with $L_1 = [v_0^1, v_1^1, \dots, v_7^1]$							
	$\chi(v_0^1)$	$\chi(v_1^1)$	$\chi(v_2^1)$	$\chi(v_3^1)$	$\chi(v_4^1)$	$\chi(v_5^1)$	$\chi(v_6^1)$	$\chi(v_7^1)$
ais12.cnf	1	1	1	1	1	1	1	1
hanoi4.cnf	1	2	2	3	3	3	4	4
huge.cnf	1	1	2	2	2	2	3	3
jnh1.cnf	1	2	2	3	4	4	4	5
par16-1.cnf	1	1	1	2	2	2	2	2
par16-1-c.cnf	1	2	2	3	3	4	4	5
pret150_75.cnf	1	1	2	2	3	3	4	4
s3-3-3-8.cnf	1	1	2	3	3	4	4	5
ssa7552-160.cnf	1	1	2	3	4	4	5	6
sw100-5.cnf	1	1	2	2	2	2	3	3
Urq8_5.cnf	1	1	2	2	3	3	4	4
uuf250-0100.cnf	1	1	2	2	3	3	4	4

Results regarding maximum sizes of intersections with layers calculated by greedy algorithms from [20] are shown in Table 1. Parts of the auxiliary constraint graph restricted on the first layer of several instances are shown in Fig 5.

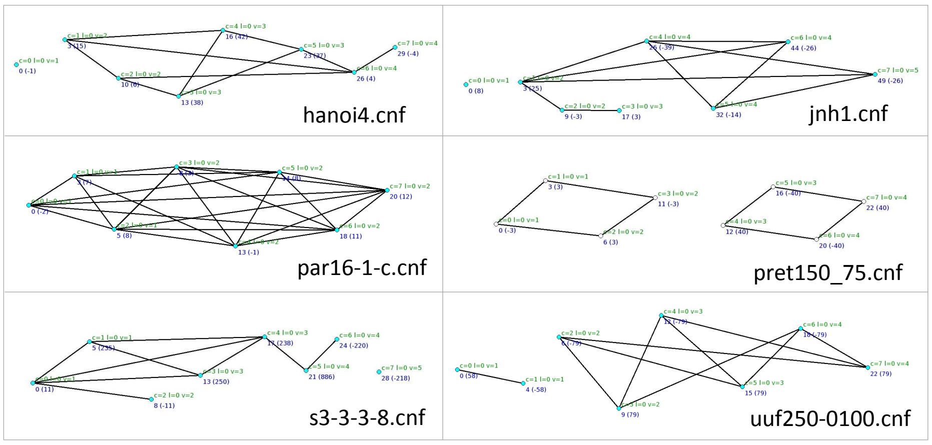


Fig 5. *An illustration of first layers of the auxiliary constraint graph of several satisfiability instances.* Several sparse and dense graphs are shown together with calculated maximum sizes of intersection of the path being constructed with the layer.

These results indicate that information captured by constraints over non-neighboring variables in the sequence of variables is relatively strongly reflected in the maximum sizes of intersections with layers. It is evident that the calculation of maximum sizes of intersections is more successful (that is, the resulting allowed size of the intersection is smaller) on instances with dense auxiliary constraint graphs. The extreme case is when a layer is represented by a complete graph with maximum size of the intersection equal to 1. Consequently, modified PC is most successful in such cases.

Table 2. *Comparison of the standard path-consistency (PC) and modified path-consistency (mPC) in terms of* ***inference strength***. The number of newly inferred clauses by both tested consistencies on difficult SAT instances [1] is reported. The comparison of preprocessing abilities is shown in terms of the number of decisions made by the Minisat2 SAT solver of instances augmented by inferred clauses (N/A indicated that the result was not obtained due to the timeout of 10.0 seconds).

SAT instance	**Instance characteristics**		**Inferred binary clauses**		**Minisat2 decisions**		
	Variables	Clauses	PC	mPC	Original	PC	mPC
hole6	42	133	0	42	1777	1777	1
hole7	56	204	0	56	10123	10123	1
hole8	72	297	0	72	40554	40554	1
hole9	90	415	0	90	202160	202160	1
chnl10_11	220	1122	0	220	N/A	N/A	1
chnl10_12	240	1344	0	240	N/A	N/A	1
chnl10_13	260	1586	0	260	N/A	N/A	1
chnl11_12	264	1476	0	264	N/A	N/A	1
chnl11_13	286	1742	0	286	N/A	N/A	1
chnl11_20	440	4220	0	440	N/A	N/A	1
fpga10_12_uns_rcr	240	1344	0	240	N/A	N/A	1
fpga10_13_uns_rcr	260	1586	0	260	N/A	N/A	1
fpga10_15_uns_rcr	300	2130	0	300	N/A	N/A	1
fpga10_20_uns_rcr	400	3840	0	400	N/A	N/A	1
fpga11_11_uns_rcr	264	1476	0	264	N/A	N/A	1
fpga11_12_uns_rcr	286	1742	0	286	N/A	N/A	1

Table 3. *Comparison of the standard path-consistency (PC) and modified path-consistency (mPC) in terms of* ***runtime***. Runtimes necessary for preprocessing, solving original and preprocessed instance, and total runtime (preprocessing + solving) are reported.

SAT instance	**Preprocessing runtime** (sec.)		**Minisat2 solving runtime** (sec.)			**Total solving runtime** (sec.)	
	Runtime PC (sec.)	Runtime mPC (sec.)	Original	PC	mPC	PC	mPC
hole6	0.01	0.04	0.00	0.00	0.00	0.01	0.04
hole7	0.02	0.14	0.10	0.10	0.00	0.12	0.14
hole8	0.04	0.32	0.48	0.48	0.00	0.52	**0.32**
hole9	0.07	0.64	3.61	3.61	0.00	3.68	**0.64**
chnl10_11	0.23	2.38	> 10.0	> 10.0	0.00	> 10.0	**2.38**
chnl10_12	0.25	2.6	> 10.0	> 10.0	0.00	> 10.0	**2.6**
chnl10_13	0.27	2.82	> 10.0	> 10.0	0.00	> 10.0	**2.82**
chnl11_12	0.36	4.18	> 10.0	> 10.0	0.00	> 10.0	**4.18**
chnl11_13	0.39	4.54	> 10.0	> 10.0	0.00	> 10.0	**4.54**
chnl11_20	0.63	7.05	> 10.0	> 10.0	0.00	> 10.0	**7.05**
fpga10_12_uns_rcr	0.25	2.61	> 10.0	> 10.0	0.00	> 10.0	**2.61**
fpga10_13_uns_rcr	0.28	2.82	> 10.0	> 10.0	0.00	> 10.0	**2.82**
fpga10_15_uns_rcr	0.32	3.27	> 10.0	> 10.0	0.00	> 10.0	**3.27**
fpga10_20_uns_rcr	0.45	4.37	> 10.0	> 10.0	0.00	> 10.0	**4.37**
fpga11_11_uns_rcr	0.36	4.18	> 10.0	> 10.0	0.00	> 10.0	**4.18**
fpga11_12_uns_rcr	0.39	4.54	> 10.0	> 10.0	0.00	> 10.0	**4.54**

Instances where this phenomenon can be observed are shown in Table 2 and Table 3. These results show how the modified PC can be successful as preprocessing tool when applied on well known difficult SAT instances which auxiliary constraint graphs can be decomposed into layers forming complete graphs. A significant improvement was achieved using modified PC in terms of runtime as well as in terms of the number of decisions made the SAT solver. The comparison with standard PC is shown for reference – expectably it is unable to infer any new clause.

Although additional experimental evaluation showed that modified PC can infer many new clauses and reduce the number of decisions of the SAT solver on other types of instances (for example that from Table 1) [20], current implementation has too long runtime to be successfully applied as a preprocessing tool these instances in a greater scale.

6.2 Tests of (2,*k*)-Consistency

Limits how modified PC can be further improved were explored using $(2,k)$-consistency. Regarding the inference strength $(2,k)$-consistency any improvement of modified PC cannot infer more than $(2,k)$-consistency. Hence, if there is no relevant set of instances where $(2,k)$-consistency significantly more clauses than PC, then modified PC has even less chance. Results shown in Table 4 and Table 5 indicate not only that this is not the case but even that $(2,k)$-consistency itself can successfully used as a preprocessing tool in spite of the fact that it is computationally expensive. The class of instances where the preprocessing was successful encodes difficult integer factorization problems [2].

Table 4. *Comparison of the standard path-consistency (PC) and (2,k)-consistency ((2,k)-c) in terms of* ***inference strength***. The number of newly inferred clauses on difficult integer factorization instances [2] is reported. The performance in terms of the number of decisions made by the Minisat2 solver on original and preprocessed instances is shown.

SAT instance	**Instance characteristics**		**Inferred binary clauses**		**Minisat2 decisions**		
	Variables	Clauses	PC	(2,k)-c	Original	PC	(2,k)-c
difp_19_0_arr_rcr	1201	6563	0	675	142710	142710	61317
difp_19_0_wal_rcr	1755	10446	103	281	73018	339662	25340
difp_19_1_arr_rcr	1201	6563	6	307	250692	87894	81144
difp_19_1_wal_rcr	1755	10446	363	561	129235	133055	77039
difp_19_2_wal_rcr	1755	10446	38	212	288500	207775	98374
difp_19_3_arr_rcr	1201	6563	128	342	114648	122379	100824
difp_19_3_wal_rcr	1755	10446	36	202	609247	968223	109741
difp_20_0_arr_rcr	1201	6563	91	754	8174	39097	12598
difp_20_0_wal_rcr	1755	10446	378	553	65601	752497	123562
difp_20_1_wal_rcr	1755	10446	10	131	362145	540378	147005
difp_20_2_arr_rcr	1201	6563	57	611	62119	438572	49700
difp_20_2_wal_rcr	1755	10446	866	2375	184778	177142	15415
difp_20_3_arr_rcr	1201	6563	0	73	142823	142823	89801
difp_20_3_wal_rcr	1755	10446	357	5798	26905	159962	45492

It is often the case that a preprocessing influences the SAT solver's performance in an unpredictable way. Hence it is another important point here that the improvement of the number of decisions and runtime seems to be relatively stable on preprocessed instances from the given class.

Table 5. *Runtime Comparison of the standard path-consistency (PC) and (2,k)-consistency ((2,k)-c) in terms of* ***runtime***. Runtimes for preprocessing, solving original and preprocessed instance, and total runtime (preprocessing + solving) are reported.

SAT instance	**Preprocessing time** (sec.)		**Minisat2 solving time** (sec.)			**Total solving time** (sec.)	
	Runtime PC	Runtime (2,k)-c	Original	PC	(2,k)-c	PC	(2,k)-c
difp_19_0_arr_rcr	3.15	3.19	27.78	27.78	9.63	30.93	**12.82**
difp_19_0_wal_rcr	3.74	3.58	10.97	68.94	2.68	72.68	**6.26**
difp_19_1_arr_rcr	3.00	3.06	54.17	15.99	14.21	**18.99**	**17.27**
difp_19_1_wal_rcr	3.56	3.48	24.19	24.86	14.21	28.42	**17.69**
difp_19_2_wal_rcr	3.58	3.43	65.13	41.53	18.85	**45.11**	**22.28**
difp_19_3_arr_rcr	3.00	3.57	20.59	22.80	16.86	25.80	**20.43**
difp_19_3_wal_rcr	3.79	3.48	164.05	286.71	19.59	290.50	**23.07**
difp_20_0_arr_rcr	3.04	3.43	0.73	5.69	1.11	8.73	12.16
difp_20_0_wal_rcr	3.64	3.62	10.48	208.25	23.45	211.89	27.07
difp_20_1_wal_rcr	3.99	3.69	83.49	134.27	28.22	137.96	**31.91**
difp_20_2_arr_rcr	3.01	3.36	9.57	108.28	7.40	111.29	10.76
difp_20_2_wal_rcr	23.3	25.6	38.49	37.47	1.62	60.77	**27.22**
difp_20_3_arr_rcr	17.33	19.6	27.73	27.73	16.78	45.06	36.38
difp_20_3_wal_rcr	21.94	23.8	3.54	31.27	7.33	53.21	31.13

7 Conclusion and Future Work

The recently proposed concept of consistency for Boolean satisfiability called modified PC has been revisited in this paper. The new type of consistency augments the standard PC by exploiting global properties of the input instance. Particularly, stronger requirements are imposed on the path being checked to exist.

The experimental evaluation where modified PC is used as a SAT preprocessing tool discovered that it is especially successful on hard satisfiability instances with structured constraint network [1]. Unfortunately, modified PC is not so successful on other instances; the preprocessing runtime typically nullify all the eventual benefit gained by the shorter runtime of the SAT solver.

Another experimental evaluation was targeted on assessing possible limits of improvement of the modified PC by running $(2, k)$-consistency on the selected set of SAT instances. This experiment showed that even relatively computationally costly $(2, k)$-consistency can serve as an efficient preprocessing tool.

There is still lot of work for the future. It is necessary to improve implementation of the modified PC and to evaluate it on larger set of benchmark instances. Nevertheless, the most desirable goal is to make improvement of modified PC so that it can be better alternative than both standard PC and $(2, k)$-consistency.

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), 731-736, USA, ACM Press, 2002, http://www.aloul.net/benchmarks.html, [March 2011].

2. Aloul, F. A.: SAT Benchmarks, Difficult Integer Factorization Problems. Research web page, http://www.aloul.net/benchmarks.html, [March 2011].
3. 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.
4. Chmeiss, A., Jégou, P.: Efficient Constraint Propagation with Good Space Complexity. Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (CP 1996), pp. 533-534, LNCS 1118, Springer, 1996.
5. Chvátal, V.: Tough Graphs and Hamiltonian Circuits. Discrete Mathematics 306, Volume 10-11, pp. 910-917, Elsevier, 2006.
6. 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.
7. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C.: Introduction to Algorithms (Second edition), MIT Press and McGraw-Hill, 2001.
8. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers, 2003.
9. 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.
10. Eén, N., Sörensson, N.: MiniSat — A SAT Solver with Conflict-Clause Minimization. Poster, 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2005), 2005.
11. Han, C. C., Lee, C. H.: Comments on Mohr and Henderson's Path Consistency Algorithm. Artificial Intelligence, Volume 36(1), pp. 125-130, Elsevier, 1988.
12. 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].
13. 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.
14. Mohr, R., Henderson, T. C.: Arc and Path Consistency Revisited. Artificial Intelligence, Volume 28 (2), 225-233, Elsevier Science Publishers, 1986.
15. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences, Volume 7, pp. 95-132, Elsevier, 1974.
16. Petke, J., Jeavons, P.: Local Consistency and SAT-Solvers. Proceedings of Principles and Practice of Constraint Programming, 16th International Conference (CP 2010), pp. 398-413, LNCS 6308, Springer, 2010.
17. Rossi, F., Dhar, V., Petrie, C.: On the Equivalence of Constraint Satisfactions Problems. Proceedings of the 9th European Conference on Artificial Intelligence (ECAI 1990), pp. 550-556, 1990.
18. 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.
19. 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.
20. 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.
21. 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.