Making Path Consistency Stronger for SAT *

Pavel Surynek

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
surynek@ktiml.mff.cuni.cz

Abstract. We are dealing with the problem of enforcing certain level of consistency in Boolean satisfaction problems (SAT problems). As a starting point we took path-consistency which we made stronger by requiring a more restrictive condition on paths connecting consistent pairs of values. Unfortunately, we found that enforcing such a modified path-consistency is an NP-complete problem. However, the definition of the modified path-consistency provides many degrees of freedom for relaxing it. This allowed us to propose an approximation algorithm for enforcing the modified path-consistency. The algorithm enforces a relaxation of the originally proposed modified path-consistency. This relaxed consistency represents a trade-off between the inference strength and the complexity of propagation algorithm. Our experimental evaluation on several SAT benchmark problems showed that the consistency enforced by our approximation algorithm has still significantly better inference strength than the standard path-consistency.

Keywords: local consistency, global consistency, path-consistency, SAT

1 Introduction and Motivation

We are introducing a new consistency technique in this paper. Our new consistency is inspired by *path-consistency* [19] which we made stronger by strengthening the condition required by the consistency over the problem. The standard path-consistency was designed to enforce certain level of local consistency in *constraint satisfaction problems* (*CSPs*) [9]. If we simplify the situation the task of finding a solution of the CSP is to assign values to the variables so that the constraints over these variables are all satisfied. A *consistency technique* in this context is a tool for making the given CSP easier for solving by forbidding values or combinations of values from participating in solution (values or combinations of values are filtered out by the consistency).

The standard path-consistency regards the given CSP as an undirected graph where the possible values for the variables represent vertices and pairs of values allowed by

* This work is supported by the Czech Science Foundation (Grantová agentura České republiky - GAČR) under the contract number 201/05/H014 and by Grant Agency of Charles University (Grantová agentura Univerzity Karlovy - GAUK) under the contract number 356/2006/A-INF/MFF.

the constraints represent edges of the graph. A pair of values is path-consistent in this interpretation with respect to a selected sequence of variables if these values are connected by an edge and they can be connected by a path going through the possible values for the selected sequence of variables (the path must connect vertices representing the values of the variables neighboring in the selected sequence of variables).

Our modification of path-consistency makes the condition on path existence more restrictive. We partition the set of values into disjoint sets. For each such set we calculate the maximum number of visits by a path according to the restrictions determined by the constraints. A pair of values is consistent in our modification with respect to a sequence of variables if these values are connected by an edge and there exists a path going through the possible values for the selected sequence of variables (this is the same as standard path-consistency) and this path respects the maximum numbers of visits in sets of values of the partitioning (the path does not visit any set of the partitioning more times than it is allowed).

As the primary area of application of our new consistency technique we intend *Boolean satisfaction problems* (*SAT*) [7, 10]. It is possible to model a given SAT problem as a CSP and hence we can reason about the SAT problem is terms of *constraint programming* [9] techniques. We use the standard transformation of SAT problems to CSPs that is called a *literal encoding* [24].

We are especially interested in SAT problems which are difficult for today's SAT solvers [1, 2]. Our aim is to use the new consistency technique to simplify such problems. Another field of our interest are problems where the standard local filtration techniques such as *unit propagation* [10], *arc-consistency* [18], *path-consistency*, *(i,j)-consistency* [13] (where *i,j* are incomparably smaller than the size of the problem) are too weak to provide any useful filtration. This implies that possible usage of the proposed new consistency technique is preprocessing of SAT problems. The given SAT problem may be first simplified by the consistency and then submitted to the general solver. In this process we generally hope that the simplified problem is solved faster than the original one. Alternatively the technique may be directly integrated into the solver. However, this is currently out of scope of this work.

The usage of path-consistency and its derivations is limited by the fact that they are too costly to be enforced. The standard path-consistency can be enforced along all possible paths by enforcing it for paths of the length at most three [13, 19]. However, this is still too costly. We are trying to overcome this limitation by enforcing the consistency for a set of selected paths only. The paths for that we enforce the consistency are heuristically selected in the way to provide the maximum filtering effect.

The main contribution of this paper is the proposal of the new consistency technique. Next we develop a simple approximation algorithm for enforcing the relaxed version of the proposed consistency. The justification for designing an approximation algorithm is the proof that enforcing the proposed consistency exactly as it is defined is an *NP*-complete problem. Finally, we give an experimental comparison of the new technique with path-consistency on several SAT benchmarks by which demonstrate the strength of the new consistency. We present two types of comparisons in this paper. First, we compare the number of inferred inconsistent pairs of values by the standard path-consistency and by the relaxation of the modified path-consistency (this is comparison of pure inference strength). Second, we compare the effect of preprocessing of the problems by the by same consistency techniques as in the first compari-

son. Both tests showed that our modified path-consistency provides significantly stronger filtering effect and it is more efficient for problem preprocessing. The problems preprocessed by the modified path-consistency are solved faster than the problems preprocessed by the standard path-consistency.

The paper is organized as follows. First we introduce the new consistency technique. Then we show that enforcing the proposed consistency in an *NP*-complete problem. Next we describe an approximation algorithm for enforcing the consistency. Finally, we evaluate our proposal experimentally on a set of benchmark SAT problems.

2 Modification of Path-Consistency

Although the standard version path-consistency [19] seems to take into account a large part of the problem (a path that is a sequence of variables) at once it is actually a local consistency. This observation is due to the well known theorem that states that the problem is path-consistent for all possible paths if it is path-consistent for paths consisting of at most 3 variables. This is the first aspect that we concentrated on when we were proposing a modification of the consistency. That is, we are trying to make path-consistency more global than the original version.

The second main property of path-consistency is that it ignores constraints whose scopes does not contain any pair of neighboring variables on the selected path. This property may relax the problem too much so that the filtration technique based on path-consistency may not detect any conflict. This aspect was the second that we are trying to overcome.

Before we can describe formally our modification of path-consistency we need to introduce definitions of some basic concepts we are using in the following text.

Definition 1 ***(CNF, SAT).*** A Boolean formula in *conjunctive normal form* (*CNF*) form is a conjunction of disjunctions where the individual conjuncts are disjunctions of literals. A *literal* is a Boolean variable or the negation of a Boolean variable. A *Boolean variable* is a variable that can be assigned either the value *true* or the value *false*. A *Boolean satisfaction problem* (*SAT*) [7, 10] consists in answering the question whether it is possible to assign values to the variables so that the given formula is satisfied. □

Definition 2 ***(CSP).*** A *constraint satisfaction problem* (*CSP*) [9] is a pair (X,C) where X is a finite set of variables and C is a finite set of constraints. Each variable from the set X has assigned a finite domain of values; a domain of a variable $x \in X$ is denoted as $D(x)$. Constraints from the set C are arbitrary relations over their variables. The set of variables constrained by a constraint is called a *scope* of the constraint. If scopes of all the constraints contain at most two variables we are speaking about *binary* CSP. A *solution* of a given constraint satisfaction problem is an assignment of values to variables respecting the domains such that all the constraints are satisfied. □

Definition 3 ***(Path-consistency).*** Let us have a binary CSP $=(X,C)$ and a sequence $P=\{P_1,P_2,\ldots,P_m\}\subseteq X$ consisting of different variables. A pair of values $p_1\in D(P_1)$ and $p_m\in D(P_m)$ (the first value is from the first variable's domain and the second value is from the last variable's domain in the sequence) - is *path-consistent* (*PC*) if the assignment $P_1=p_1$ and $P_m=p_m$ satisfies all the constraints that contains these two variables in their scopes and there exists an assignment of values to the variables $P_2,\ldots,P_{m-1}$ such that all the constraints containing any two variables neighboring in the sequence in their scopes are satisfied. □

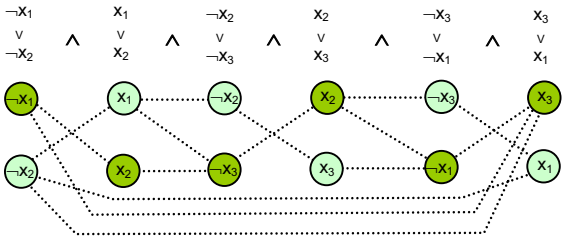

Fig. 1. *Illustration of path-consistency in the CSP model of a SAT problem.* The SAT problem shown here is a representation of the requirement of selecting an odd number of variables from every of the following sets to be *true*: $\{x_1, x_2\}$, $\{x_1, x_3\}$, $\{x_2, x_3\}$. The problem has no solution. However, the support $\neg x_1$ from the left most variable and the support x_3 from the right most variable are path-consistent with respect to the sequencing of variables from the left to the right since they are non-conflicting and there exists a path from the left to the right consisting of the edges between neighboring variables connecting the supports (supports on the path are darker).

As we stated above we design the consistency for application on SAT problems. In order to use a concept designed for CSP in Boolean satisfiability we model a given SAT problem as a CSP. We suppose the SAT problem is given as a formula in CNF. We model it as a binary CSP where the variables ($\{C_1,C_2,\ldots,C_n\}$) are clauses with the finite domains consisting of the literals supporting the clause (values in the domains of the variables are called supports). Binary constraints are introduced over every pair of variables. The constraints forbid assignments of incompatible literals to the variables. More precisely, a constraint with the scope $\{C_i,C_j\}$ forbids assignments $C_i=x,C_j=\neg x$ and $C_j=\neg x,C_i=x$ for any Boolean variable x. This model is called a *literal encoding* in [24].

The CSP model proposed here can be regarded as an undirected graph where vertices are represented by values in variable domains and edges are represented by pairs of allowed values (that is, compatible literals from the domains of different variables are connected by an edge). Path-consistency in this interpretation looks like as follows. A pair of values (supports) from two different variables is path-consistent along a given sequence of variables if there is an edge between the supports and there exists a path connecting the supports consisting of edges between the neighboring variables in the sequence. This interpretation of path-consistency allows us to explain its modification more easily.

Let us have a sequence of different variables of the CSP model $C_{s(1)},C_{s(2)},\ldots,C_{s(m)}$ for $m\le n$. We partition supports from the variables of the sequence into disjoint sets

$L_1, L_2, \ldots, L_k$ called layers (the same literals supporting different variables are regarded as different supports). A single layer contains at most one support from the domain of each variable (formally it is $D(C_{s(j)}) \cap L_i \leq 1$ for $j = 1,2,\ldots,m$ and $i = 1,2,\ldots,k$). For each support we compute or estimate the maximum number of supports that can be selected in its layer preceding the support (including the support) with respect to the sequencing of variables. More formally, let $p \in D(C_{s(a)})$ be a support and let L_b be a layer such that $p \in L_b$ then $t(p)$ denotes estimation of the maximum number of supports that can be selected (visited by a path) in the set $\{q \mid q \in L_b \,\&\, q \in \bigcup_{j=1}^{a} D(C_{s(j)})\}$ not violating the constraints over the set. Having this definition we can propose a more restrictive condition for path-consistency with respect to a given sequence of variables. A pair of supports $p_1 \in D(C_{s(1)})$ and $p_m \in D(C_{s(m)})$ is path-consistent with respect to a modified definition of path-consistency if there is an edge connecting p_1 and p_m and there is a path $p_1, p_2, \ldots, p_{m-1}, p_m$, where $p_j \in D(C_{s(j)})$ for $j = 1,2,\ldots,m$ and $\sum_{j=1}^{i} \left| \{p_j\} \cap L_c \right| \leq t(p_i)$ for $i = 1,2,\ldots,m$ where c is such that $p_i \in L_c$.

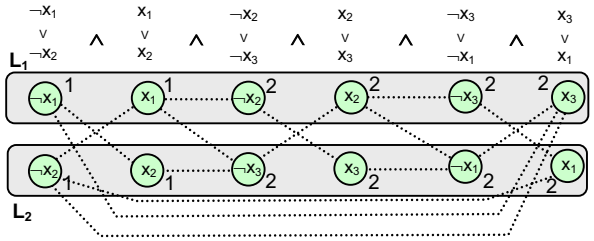

Fig. 2. *Illustration of the modified version of path-consistency in the CSP model of a SAT problem.* The problem is the same as in figure 1. Individual supports are partitioned into disjoint sets called *layers*. For each element of a layer the maximum number of supports preceding the element (including the element) in the layer such that they can be selected together is computed (or estimated). The support $\neg x_1$ from the left most variable and the support x_3 from the right most variable are not path-consistent with respect to the sequencing of variables from the left to the right since they cannot be connected by a path that respects maximum numbers of supports that can be selected (visited by the path) in the layers.

3 A Motivating Example of Application on Difficult SAT Problem

We plan to apply the proposed modified version of path-consistency for solving difficult SAT problems [2, 12]. By the term difficult problem we mean a problem that is difficult for today's SAT solvers. An illustration of the modified version of path-consistency on the *pigeon holes principle* problem that is a well known difficult problem is shown in figure 3. This problem resists standard local consistencies (that is the consistency cannot perform any inference) [2, 12]. It is also difficult for today's SAT solvers. Despite this fact the proposed path-consistency successfully infers that the problem has no solution.

In fact this problem served us as the first motivating example for that we were trying to generalize the standard path-consistency. It is also possible to observe that the modified path-consistency can detect global properties of the problem.

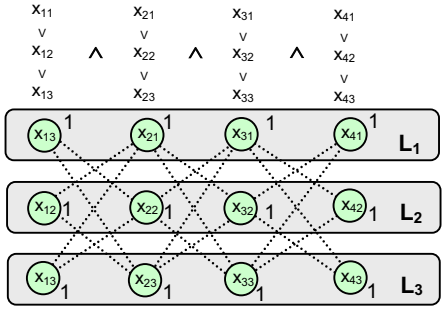

Fig. 3. *The modified version of path consistency on pigeon holes principle problem.* The problem consists of 4 pigeons and 3 holes. The task is to place all the pigeons into holes such that no two pigeons are placed in the same hole. The first clause represent requirement that first pigeon is placed in hole 1 or in hole 2 or in hole 3 and so on. Binary clauses are not shown.

4 Approximation Algorithm for Modified Path-Consistency

To be able to further study properties of the proposed consistency technique we need to formally define its enforcing algorithm. We will try to develop a consistency enforcing algorithm for a given sequence of variables $C_{s(1)},C_{s(2)},\ldots,C_{s(m)}$ and for a given partitioning of values into layers $L_1,L_2,\ldots,L_k$ with pre-calculated estimations of maximum numbers visits. In the ideal case we want the algorithm to calculate all pairs of inconsistent values from the Cartesian product $D(C_{s(1)})\times D(C_{s(m)})$ with respect to the defined consistency. But first, we should deal with feasibility of developing of such an algorithm. Unfortunately the following theorem gives us a negative result.

Theorem 1 ***(NP-completeness of modified PC).*** The problem of finding a Hamiltonian path [5, 6] is polynomial time reducible to the problem of enforcing modified path-consistency for a given sequence of variables and for a given partitioning of values into layers. Hence the problem of enforcing modified path-consistency is *NP*-complete. ■

Proof. Having an undirected graph $G=(V,E)$ it is not difficult to construct an instance of CSP where paths proclaiming modified path-consistency directly correspond to Hamiltonian paths.

Let us construct a square array of $|V|$ rows and $|V|$ columns. Each cell of the array is labeled by the first component of the corresponding element of the Cartesian product $V\times V$ (the first component stands for a row, the second component stands for a column). Columns of the array represent variables of the constructed CSP. The do-

main of a variable consists of elements of the corresponding column. A constraint over a pair of variables u and v neighboring in the array allows all pairs $(x, y) \in D(u) \times D(v)$ such that $\{x, y\} \in E$. Let us partition the values of the CSP according to the rows of the array (each row is a layer). Next let us set all the estimations of maximum numbers of values that can be selected in layers to 1 (by this we express a requirement that a Hamiltonian path visits each vertex at most once). The sequencing of variables in the array determines a sequence of variables for checking the consistency. If we detect that a pair of values - the first is from the domain of the variable corresponding to the first column of the array and the second is from the domain of the variable corresponding to the last column - is path-consistent according to our modification there exists a path witnessing this fact (the path goes from one side of the array to the other side of the array; by this we express a requirement that the length of a Hamiltonian path is equal to the number of vertices). This path directly corresponds to a Hamiltonian path. It is trivial to observe that the construction of the above CSP and final extraction of the Hamiltonian path can be done in polynomial time with respect to the size of the input graph G. ■

In the light of the above theorem there is little hope to develop an efficient algorithm identifying all the inconsistent pairs of values with respect to the modified version of path-consistency. Hence we have to settle for some kind of an approximation algorithm. Our approximation algorithm should try to follow modified path-consistency definition but we relax from the requirement of identifying all the inconsistent pairs of values. In other words, when the algorithm identifies a pair of inconsistent values then these values are inconsistent but we do not require the other implication (if the pair is inconsistent the algorithm does not need to find it).

Our approximation algorithm is inspired by the Dijkstra's algorithm for single source shortest paths [8]. Briefly said, our augmentation of this algorithm is that we are trying to calculate minimum number of visits to each layer of the decomposition by all the restricted paths ending in a single vertex. The formalization of this idea is given by the code in the algorithm 1.

The algorithm is structured into two functions. The main control function *enforceeModifiedPC* calculates pairs of inconsistent values. The auxiliary function *calculateConflictsMPC* calculates values that are in conflict with a given value. The core of the algorithm (lines 14-29) proceeds from the beginning of the sequence of variables to the end. In each step the minimum number of visits in each layer by paths ending in a given value is calculated. The minimum number of visits for a variable is calculated according to the minimum number of visits for the previous variable in the sequence while we are respecting additional conditions. The additional conditions are as follows. A path can proceed from one value to other value if constraints allow this (lines 21-22). The second major condition is that a path can proceed to a value if it does not exceed the maximum number of visits to a corresponding layer (line 24).

Let us briefly summarize main properties of the proposed consistency enforcing algorithm. Observe that the algorithm runs in polynomial time of $O(m|D|^2 k)$, where m is the length of the sequence of variables, $|D|$ is the size of variable domains, and k is the number of layers in the decomposition. It is also easy to see that although the algorithm relaxes from the original definition of modified path-consistency it still

enforces a stronger level of consistency than the original PC. Finally let us note that the algorithm can be directly used as a building block of a propagation algorithm.

We use the term *modified path-consistency* for the consistency enforced by the algorithm 1 in the following text.

Algorithm 1: Modified path-consistency enforcing approximation algorithm.

function *enforceModifiedPC* ($[C_{s(1)}, C_{s(2)}, \ldots, C_{s(m)}], [L_1, L_2, \ldots, L_k]$) : **set**
1: $inconsistent \leftarrow \emptyset$
2: **for each** $p \in D(C_{s(1)})$ **do**
3: $conflicts \leftarrow calculateConflictsMPC(p_1, [C_{s(1)}, C_{s(2)}, \ldots, C_{s(m)}], [L_1, L_2, \ldots, L_k])$
4: $inconsistent \leftarrow inconsistent \cup conflicts$
5: **return** *inconsistent*

function *calculateConflictsMPC* ($p_1, [C_{s(1)}, C_{s(2)}, \ldots, C_{s(m)}], [L_1, L_2, \ldots, L_k]$) : **set**
6: **for every** $j = 1, 2, \ldots, m$ **do**
7: **for every** $i = 1, 2, \ldots, k$ **do**
8: **for each** $q \in D(C_{s(j)})$ **do**
9: $visits[i][j][q] \leftarrow +\infty$
10: **for every** $i = 1, 2, \ldots, k$ **do**
11: $visits[i][1][p_1] \leftarrow 0$ *{staying in* p_1 *no other layer except* L_l *was visited}*
12: **let** l be an index of a layer such that $p_1 \in L_l$
13: $visits[l][1][p_1] \leftarrow 1$ *{staying in* p_1 *only layer* L_l *was visited}*
14: **for every** $j = 2, \ldots, m$ **do**
15: $max_visits \leftarrow 0$
16: **for each** $r \in D(C_{s(j)})$ **do**
17: $max_visits \leftarrow max_visits + t(r)$
18: **if** $max_visits \geq j$ **then**
19: **for each** $q \in D(C_{s(j-1)})$ **do**
20: **for each** $r \in D(C_{s(j)})$ **do**
21: **if** assignment $C_{s(j-1)} = q$, $C_{s(j)} = r$ is allowed by constraints
22: with the scope containing $C_{s(j-1)}$ and $C_{s(j-1)}$ **then**
23: **let** h be an index of a layer such that $r \in L_h$
24: **if** $\min(visits[h][j][r], visits[h][j-1][q] + 1) \leq t(r)$ **then**
25: **for every** $i = 1, 2, \ldots, k$ **do**
26: **if** $i = h$ **then**
27: $visits[i][j][r] \leftarrow \min(visits[i][j][r], visits[i][j-1][q] + 1)$
28: **else**
29: $visits[i][j][r] \leftarrow \min(visits[i][j][r], visits[i][j-1][q])$
30: $conflicts \leftarrow \emptyset$
31: **for each** $q \in D(C_{s(m)})$ **do**
32: **let** h be an index of a layer such that $q \in L_h$
33: **if** $visits[h][m][q] = +\infty$ **then**
34: $conflicts \leftarrow conflicts \cup \{p_1, q\}$
35: **return** *conflicts*

5 Estimating Maximum Numbers of Visits

We have not yet discussed how the maximum numbers of visits by a path in the individual layers are calculated. In this little bit technical part we discuss this issue. The following text just describes simple combinatorial calculations.

To be precise, for a given sequence of variables of the model $C_{s(1)},C_{s(2)},\ldots,C_{s(m)}$ and for a given partitioning of supports (values) into layers $L_1,L_2,\ldots,L_k$ we need to calculate $t(p)$ for every support p. Let us have a support $p \in D(C_{s(a)})$ and let L_b be a layer such that $p \in L_b$. We take a set $L' = \{q \mid q \in L_b \,\&\, q \in \bigcup_{j=1}^{a} D(C_{s(j)})\}$ which is a beginning of the layer L_b. Then a clique cover $L' = K_1 \cup K_2 \cup \ldots \cup K_\kappa$ of the set L' is calculated with respect to the edges represented by pairs of forbidden values. The pairs of forbidden values are induced by the constraints of the model. The clique cover consists of pair wise disjoints complete sub-graphs. More formally, K_i is a complete sub-graph with respect to the defined edges for every $i = 1,2,\ldots,\kappa$ and $K_i \cap K_j = \emptyset$ for every $i, j = 1,2,\ldots,\kappa \,\&\, i \neq j$. Observe that at most single support (value) from each clique K_i can be selected to be assigned to the corresponding variable. Hence we can state that $t(p) \leq \kappa$.

Of course, we cannot afford to compute any kind of optimal clique cover since such problem is too complex. Therefore the clique cover is computed greedily. A largest clique is greedily computed and extracted from the graph. Such a clique is inserted into the clique cover and the process is repeated until the graph is non-empty. A single clique itself is constructed by selecting a vertex of the highest degree. This vertex is included into the constructed clique and the set of candidates for vertices to be included into the clique is restricted to the vertices that are connected by edges to all the vertices already included into the constructed clique.

Nevertheless, we can made a more precise estimation for $t(p)$. Notice that we did not took into account edges outside of the clique cover (that is, edges between cliques). If we select just one vertex from each clique (we have κ selected vertices) it still possible to have at most $S = \sum_{i,j=1,\ldots,\kappa;i<j} |K_i||K_j| - 1$ edges between cliques not violating the selection. But it is not possible if the number of edges outside the clique cover is higher than S (this is really happening very often since the greedy method finds often cliques of sizes 1). Let κ' be the size of the largest complete graph that can be constructed with edges outside the clique cover that do not fit under S. Now we know that $t(p) \leq \kappa - (\kappa' - 1)$.

The last important combinatorial trick we apply is following. It not difficult to see that the maximum numbers of visits for supports of a given layer ordered according to the sequence of variables $C_{s(1)},C_{s(2)},\ldots,C_{s(m)}$ must be non-descending. Hence we can further tighten the value of the maximum number of allowed visits associated with a support. We proceed from the end to the beginning of the layer according to the mentioned ordering and if the maximum number of visits for the previous support in the layer is higher than the maximum number of visits for the current support then we set the maximum number of visits for the previous support to the value of the maximum visits of the current support.

Our experimental evaluation showed that it is crucial for efficient filtering to have the constraint graph of the model as dense as possible. We need to have as many as possible forbidden pairs of supports. In other words, we need to have as many as possible edges in above graphical interpretation of the constraint model. To populate

edges we can adopt the preprocessing technique proposed in [22] The author proposes to populate constraints by enforcing singleton arc-consistency (*SAC*) in the model. Having the richer set of constraints after enforcing SAC standard path-consistency as well as our modification is more efficient.

6 Experimental Evaluation

We performed a competitive comparison of the modified path-consistency with the standard path-consistency on a set of SAT benchmarks [1, 16]. Although both methods are substantially different (standard PC is a local consistency while our modified version uses a bit of global approach) the standard path-consistency is closest existing comparable method to out proposal. We took several problems from the Satisfiability Library [16]. The selection of problems was guided by two requirements - the problems should be small and they should be non-trivial. The requirement on the size of problems is imposed by our current experimental implementation.

Table 1. *Comparison of the modified path-consistency and standard path-consistency* on several SAT benchmark problems from SAT Library. Numbers of discovered pairs of inconsistent values are compared.

SAT Problem	Number of **variables**	Number of **clauses**	Pairs filtered by **standard PC**	*Pairs filtered by **modified PC***
bw_large.a	495	4675	22	*22*
hanoi4	718	4934	9	***10***
huge	459	7054	12	*12*
jnh2	100	850	135	***147***
logistics.a	828	6718	192	*192*
medium	116	953	177	***227***
par8-1-c	64	254	0	***19***
par8-2-c	68	270	0	***9***
par8-3-c	75	298	0	***100***
par8-4-c	67	266	0	***42***
par8-5-c	75	298	0	***24***
par16-1-c	317	1264	0	***11***
par16-2-c	349	1392	0	***7***
par16-3-c	334	1332	0	***7***
par16-4-c	324	1292	0	***8***
par16-5-c	341	1360	0	***4***
ssa0432/003	435	1027	81	***1598***
ssa2670/130	1359	3321	4	***2656***
ssa2670/141	986	2315	20	***8871***
ssa7552/038	1501	3575	16	***5652***
ssa7552/158	1363	3034	49	***2371***
ssa7552/159	1363	3032	37	***2379***
ssa7552/160	1391	3126	2	***2344***

We implemented the presented approximation algorithm and the single source shortest path algorithm [8] in C++. The second algorithm is used to simulate standard path-consistency. We do not enforce the consistency for all the sequences of variables in the problem but only for selected sequences of variables that seems to be promising. We select sequences of variables that are highly constrained. This selection increases the probability of filtration. For both methods - for the standard path consistency and for the modified version - we use the same set of sequences of variables. Such selection ensures equal conditions for both methods in comparison.

We list results for benchmark problems where either the modified PC or the standard PC gave non-trivial results (non-zero number of inconsistent pairs of values is discovered) in table 1. Consistencies were enforced for 2048 selected sequences of variables. The length of sequences were uniformly distributed between 4 and 10 (for the length of 3 both consistency techniques are equal). For majority of benchmark problems we did not obtained any non-trivial result (both consistencies inferred 0 inconsistencies). This is caused by the fact that majority of benchmark problems has sparse constraint graphs even after enforcing singleton arc-consistency. Contrary to this problems listed in table 1 has quite dense constraint graphs.

We select only 2048 sequences for that we enforce consistencies since theoretically both consistencies are computationally too costly. We cannot afford to enforce path-consistency for all sequences of length 3 (this implies path-consistency for all sequences of variables) since there are too many such triples. So much the more this holds for the modified path-consistency. In this case we would have to enforce the consistency for all sequences of variables which is not possible.

The results show that the proposed modified version of path-consistency provides stronger filtration than the standard version (even if we enforce an approximation of it). In some cases the modified version is significantly stronger. This is especially the case of problems with dense constraint graphs.

Regarding performance the shortest path algorithm for simulating path-consistency is about 25% faster than the proposed approximation algorithm (the comparison was made on a 1600MHz AMD Opteron machine with 1GB of memory under Mandriva Linux 10.2). However, our implementation is far from well optimized so this result is estimative only. Moreover both methods are substantially different - normally we should use some of the PC-_ algorithms [19] instead of single source shortest path for enforcing PC. Therefore this experimental comparison should be treated as comparison of filtration strengths of both methods.

In the second experimental setup, we are concentrating on an application of enforcing the modified path-consistency for preprocessing of SAT problem instances. We are trying to answer the question whether the modified path-consistency provides some advantage in comparison with standard path-consistency on SAT problem preprocessing.

We chose four state-of-the-art SAT solvers for evaluation of the effect of preprocessing by path-consistency and by the modified path-consistency. The SAT solvers of our choice were zChaff [14, 20], HaifaSAT [15, 23], RSAT [21], and MiniSAT (a version with SATElite preprocessing integrated) [11, 12]. Our choice was guided by the results of several last SAT competitions [17] in which these solvers belonged to the winners and by the availability of the solvers (we need to run all the solvers under Linux system).

The experimental setup was as follows. We took the same set of problems as in the previous experimental setup. We performed a preprocessing step with each benchmark problem by path-consistency and by the modified path-consistency. The preprocessing consists in enforcing the selected consistency in the problem (that is, inconsistent pairs of literal occurrences are filtered out; the filtration itself is represented by adding a new clause to the original formula that forbids the simultaneous assignment of the value *true* to both inconsistent literals). Again only 2048 sequences of variables are selected in each problem for enforcing the consistency.

Each preprocessed instance was then submitted to the all the general SAT solvers of our choice. We collected number of decision steps made by the solvers on preprocessed problems. The results are presented in table 2. The ratios of the number of decision steps made by the solver on an instance preprocessed by path-consistency and on an instance preprocessed by the modified path-consistency are shown (higher ratio means that the modified path-consistency eventually achieved lower number of decision steps).

Table 2. *Experimental comparison of the standard path-consistency and the modified path-consistency on several SAT problems from SAT Library.* The ratios of the number of decision steps made by the solver on an instance preprocessed by standard path-consistency / the number of decision steps made by the solver on an instance preprocessed by the modified path-consistency are listed. Ratios where the modified path-consistency was better are shown as bold.

Problem	#variables	#clauses	HaifaSat	Minisat2	Rsat_1_03	zChaff
bw_large.a	459	4675	1.0	1.0	1.0	1.0
hanoi4	718	4934	1.0	1.0	1.0	1.0
hanoi5	1931	14468	1.0	1.0	1.0	1.0
huge	459	7054	1.0	1.0	1.0	1.0
jnh2	100	850	1.0	1.0	1.0	**1.3**
logistics.a	828	6718	1.0	1.0	1.0	1.0
medium	116	953	1.0	1.0	0.8	0.9
par8-1-c	64	254	1.0	1.0	0.9	0.7
par8-2-c	68	270	0.9	**1.2**	0.7	0.8
par8-3-c	75	298	0.8	**1.4**	0.6	0.8
par8-4-c	67	266	**1.3**	0.9	0.5	**1.3**
par8-5-c	75	298	**1.3**	**1.9**	**3.4**	**1.4**
par16-1-c	317	1264	0.1	0.4	**2.2**	0.1
par16-2-c	349	1392	1.1	**2.3**	0.8	0.8
par16-3-c	334	1332	0.8	**1.4**	**6.6**	**1.6**
par16-4-c	324	1292	**1.3**	**1.8**	**18.6**	0.9
par16-5-c	341	1360	**2.5**	**11.9**	**22.9**	**1.2**
qg7-09	729	22060	1.0	1.0	1.0	1.0
ssa0432-003	435	1027	1.0	**228.0**	**155.0**	**122.0**
ssa2670-130	1359	3321	**51.0**	**411.0**	**371.0**	**323.0**
ssa2670-141	986	2315	**289.0**	**429.0**	**455.0**	**489.0**
ssa7552-038	1501	3575	**190.0**	**226.0**	**173.0**	**238.0**
ssa7552-158	1363	3034	**114.0**	**129.0**	**151.0**	**312.0**
ssa7552-159	1363	3032	**123.0**	**141.0**	**163.0**	**309.0**
ssa7552-160	1391	3126	**141.0**	**130.0**	**275.0**	**264.0**

The modified path-consistency proved to be a better option than the standard path-consistency on the majority of tested problems (ratio of >1.0). Moreover, the improvements reached by the new consistency technique are in orders of magnitudes on several problems.

However, both preprocessing methods has the same impact on the ability of the solver to solve the problem on several instances (ratio of 1.0). There are even the problems where the standard path-consistency is a better option for preprocessing (ratio of <1.0).

7 Related Works

The proposed modification of path-consistency is in fact a generalization of the consistency technique presented in [22] . Moreover, the modified path-consistency is able to decide by itself the same set of benchmark problems on which the technique from [22] is successful.

If we compare our consistency enforcing algorithm and the proposed consistency with existing local consistencies such as $(i, j)-$consistency [9] we may identify that our approach is more global (larger part of the problem is considered at once). Moreover, the time complexity growth for increasing i and j compared to the complexity growth for increasing portion of the problem we consider (length of sequence of variables) makes our approach more promising.

The study of local consistencies for Boolean satisfaction problems is given in [3] and in [4].

8 Concluding Remarks and Future Work

In this paper we present a new consistency technique which we call a modified path-consistency. The new consistency is based on strengthening of the condition required by the standard path-consistency. We show that enforcing the modified path-consistency exactly is *NP*-complete problem. Therefore we propose an approximation algorithm for enforcing the consistency. We evaluate the new consistency in comparison with the standard path-consistency on a set of SAT benchmark problems taken from SAT Library [16]. The results show that the proposed new consistency is stronger than the original path-consistency. This advantage is especially visible on highly constrained SAT problems.

Next we demonstrated the usefulness of the proposed modified path-consistency as a tool for preprocessing the SAT problems. We again compared the modified path-consistency and the standard version in preprocessing of SAT problems. We obtained significantly better results for the modified path-consistency. Namely, the tested SAT solvers need lower number of steps to solve problems preprocessed by the modified path-consistency.

As we stated above we made the comparison with path-consistency since it is the most similar existing method. However modified PC and the standard PC are substantially different. While standard PC is a local consistency technique the modified version is partially global since it considers a larger part of the problem (a sequence of variables) at once.

For future work we plan to develop a propagation algorithm in the style of existing propagation algorithms for local consistencies such as AC-_ or PC-_.

References

1. Aloul, F. A.: *Fadi Aloul's Home Page - SAT Benchmarks.* Personal Web Page. http://www.eecs.umich.edu/~faloul/benchmarks.html, University of Michigan, USA, 2007, (March 2007).
2. 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.
3. Bacchus, F.: *GAC Via Unit Propagation.* Proceedings of the 13th CP Conference, Providence, USA, 133-147, LNCS 4741, Springer Verlag, 2007.
4. Bessière, C., Hebrard, E., Walsh, T.: *Local Consistencies in SAT.* Proceedings of the 6th SAT Conference, Santa Margherita Ligure, Italy, LNCS 2919, 299-314, Springer Verlag, 2004.
5. Chvátal, V.: *On Hamilton's Ideals.* Journal of Combinatorial Theory 12, 163-168, Elsevier, 1972.
6. Chvátal, V.: *Tough Graphs and Hamiltonian Circuits.* Discrete Mathematics 306, Volume 10-11, 910-917, Elsevier, 2006.
7. Cook, S. A.: *The Complexity of Theorem Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 151-158, USA, ACM Press, 1971.
8. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: *Introduction to Algorithms, second edition.* MIT Press, 2003.
9. Dechter, R.: *Constraint Processing.* Morgan Kaufmann Publishers, 2003.
10. Dowling, W., Gallier, J.: *Linear-time algorithms for testing the satisfiability of propositional Horn formulae.* Journal of Logic Programming, 1(3), 267-284, Elsevier Science Publishers, 1984.
11. 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), Scotland, 2005.
12. Eén, N., Sörensson, N.: The MiniSat Page. Research Web Page. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html, Chalmers University, Sweden, (January 2008).
13. Freuder, E. C.: *A Sufficient Condition for Backtrack-Bounded Search.* Journal of ACM 32, Volume 4, 755-761, ACM Press, 1985.
14. Fu, Z., Marhajan, Y., Malik, S.: zChaff. Research Web Page. http://www.princeton.edu/~chaff/ zchaff.html, Princeton University, USA, (January 2008).
15. Gershman, R., Strichman, O.: HaifaSat – a new robust SAT solver. Research Web Page. http://www.cs.technion.ac.il/~gershman/HaifaSat.htm, Technion Haifa, Israel, (January 2008).
16. Hoos, H. H., Stützle, T.: *SATLIB: An Online Resource for Research on SAT.* In Proceedings of SAT 2000, 283-292, IOS Press, 2000. http://www.satlib.org/, 2000, (January, 2008).
17. Le Berre, D., Roussel, O., Simon, L.: The international SAT Competitions web page. http://www.satcompetition.org/, (January 2008).
18. Mackworth, A. K.: *Consistency in Networks of Relations.* Artificial Intelligence 8, 99-118, AAAI Press, 1977.
19. Mohr, R., Henderson, T. C.: *Arc and Path Consistency Revisited.* Artificial Intelligence, Volume 28 (2), 225-233, Elsevier Science Publishers, 1986.
20. Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. Proceedings of the 38th Design Automation Conference (DAC-2001), 530-535, USA, ACM Press, 2001.
21. Pipatsrisawat, K., Darwiche, A.: RSat - ...veRSATile... Research Web Page. http://reasoning.cs.ucla.edu/rsat/, University of California Los Angeles, USA, (January 2008).

22. Surynek, P.: *Solving Difficult SAT Instances Using Greedy Clique Decomposition.* Proceedings of the 7th SARA Symposium, Whistler, Canada. 359-374, LNCS 4612, Springer Verlag, 2007.
23. Strichman, O., Gershman, R.: HaifaSat: a New Robust SAT Solver. Proceedings of the 1st International Haifa Verification Conference, LNCS 3875, 76-89, Israel, Springer-Verlag. 2005.
24. 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.