A CSP Search Algorithm with Reduced Branching Factor

Igor Razgon and Amnon Meisels

Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel
{irazgon,am}@cs.bgu.ac.il

Abstract. This paper presents an attempt to construct a "practical" CSP algorithm that assigns a variable with 2 values at every step. Such a strategy has been successfully used for construction of "theoretical" constraint solvers because it decreases twice the base of the exponent of the upper bound of the search algorithm.
We present a solver based on the strategy. The pruning mechanism of the algorithm resembles Forward Checking (FC), therefore we term it 2FC. According to our experimental evaluation, 2FC outperforms FC on graph coloring problems and on non-dense instances of randomly generated CSPs.

1 Introduction

Partition of domains is a strategy that allows to reduce the size of the search space explored by a constraint solver. Consider a method based on the strategy that partitions the domain of every variable into subsets of two values (if a domain has an odd size than one subset is a singleton). The algorithm scans all constraint networks obtained by restriction of the domain of every variable to one of the partition classes. If at least one constraint network being scanned is soluble, the algorithm returns the solution found. Otherwise, it reports failure.

Assume that the considered constraint network has n variables and the maximal domain size is d. Then a simple analysis shows that the algorithm scans $O((d/2)^n)$ constraint networks if d is even and $O(((d+1)/2)^n)$ if d is odd. Taking into account that constraint networks with domains of size at most two can be solved efficiently [7], we get that these upper bounds determine the search space size of the considered algorithm. Clearly, this size is much smaller than $O(d^n)$ of FC, MAC, and others.

The strategy of domain partition has been successfully applied to the design of constraint solvers with exact upper bounds. Sophisticated techniques based on the strategy are presented in [5, 1]. However, the strategy attracted little attention of researchers that investigate "practical" approaches to construction of complete constraint solvers.

This paper introduces an attempt to construct a "practical" complete constraint solver based on domain partition. The main problem with the strategy

is that the resulting algorithm is not guaranteed to have a solution when all variables are assigned. Therefore the algorithm could generate many "long" insoluble 2-CNs (constraint networks with domain sizes at most 2) which makes the actual time of its work close to the theoretical upper bound. To overcome this difficulty, we introduce the following modifications.

- Variables are assigned one by one. A variable is assigned with 2 values if its current domain contains at least 2 values. Otherwise, the variable is assigned with one value. To present this situation in a more general form, we say that at every step of the algorithm, a variable v is assigned with a subset S of its current domain.
- Whenever a variable v is assigned with a set of values S, the algorithm removes all values of unassigned variables that are incompatible with all the values of S.
- The algorithm backtracks when the current domain of some variable becomes empty.
- Whenever a variable v is assigned with a set $\{val_1, val_2\}$, a new conflict is added between any pair $(\langle v_1, val'_1\rangle, \langle v_2, val'_2\rangle)$ of values of different unassigned variables v_1 and v_2 such that $\langle v_1, val'_1\rangle$ conflicts with $\langle v, val_1\rangle$ and $\langle v_2, val'_2\rangle$ conflicts with $\langle v, val_2\rangle$.

The proposed algorithm resembles FC with the main difference that a variable can be assigned with 2 values. Therefore we call the algorithm 2FC. The main property of 2FC is that whenever all variables are assigned, the resulting 2-CN is *guaranteed* to have a solution.

In our experimental analysis, we compared 2FC to FC. The main experimental observation is that 2FC strictly outperforms FC on graph k-coloring problem. The rate of runtime improvement changes from a factor of 2 to 10. 2FC also outperforms FC on randomly generated constraint networks with low density.

The experimental result suggest that the proposed approach could be useful in the area of graph coloring. Another possible application of 2FC follows from the fact that it returns a 2-CN which could have many solutions that can be efficiently generated. Therefore the algorithm could be useful for dynamic environments where constraints frequently change: it might allow quick replacing of an inconsistent solution by another solution without performing search.

The rest of the paper is organized as follows. Section 2 provides necessary background. Section 3 presents the 2FC algorithm. Section 4 proves correctness of the 2FC algorithm. Section 5 demonstrates result of experimental evaluation. Section 6 outlines directions of further investigation.

2 Preliminaries

The model we consider in the paper is binary constraint network (CN). A CN $Z = \langle V, D, C\rangle$ is a triple consisting of a set of *variables* V, a set of *domains* D and a set of *constraints* C. Let $V = \{v_1, \ldots, v_n\}$. Then $D = \{D(v_1), \ldots, D(v_n)\}$,

where $D(v_i)$ is the domain of values of v_i, $C = \{C(v_i, v_j) | i \neq j, 1 \leq i, j \leq n\}$, where $C(v_i, v_j) \subseteq D(v_i) \times D(v_j)$ is the set of all *compatible* pairs of values of v_i and v_j. We refer to the parts of Z as V_Z, D_Z, and C_Z. To emphasize that a value val belongs to the domain of a variable v, we refer to this value as $\langle v, val \rangle$. In this paper we consider a special case of constraint network, *2-CN*, that is a CN for which every domain has at most 2 values.

Given a CN Z as above, the task of Constraint Satisfaction Problem (CSP) is to find a set $P = \{\langle v_1, val_1 \rangle, \ldots, \langle v_n, val_n \rangle\}$ such that every val_i belongs to the domain of v_i and all values of P are mutually compatible (consistent), or to report that there is no such a set. The set P is called a *solution* of Z.

A typical CSP search algorithm like Forward Checking (FC) [4] usually creates a solution in an iterative manner. During its work it maintains a consistent set of values of a subset of variables and tries to extend it to a solution. This set of values is called a *partial* solution. Variables whose values are contained in the partial solution are called *assigned*. Other variables are called *unassigned*. The present paper introduces an algorithm that can assign a variable with more than one value. We refer to a set of assigned values maintained by the algorithm as an *extended partial solution*. When all variables are assigned, the algorithm has an *extended solution*.

The following notion is frequently used further in the paper.

Definition 1. *Let Z be a CN and let S be a set of values of Z. A subnetwork Z' of Z induced by S is obtained as follows:*

- *take to Z' only those variables of Z whose values appear in S;*
- *the domain of every variable v of Z' is the intersection of the domain of v in Z with S;*
- *two values are compatible in Z' if and only if they are compatible in Z.*

To illustrate the notion, consider Figure 1. On the left side of the figure there is a CN Z. Ellipses represent variables, black points represent values, conflicting values are connected by arcs. The CN Z' is a subnetwork of Z induced by the set of values encircled by additional circles. Note that Z' does not contain variable V_4 because no value of the domain of V_4 is included in the inducing set of values. Also note that there are conflicts between $\langle V_1, 2 \rangle$ and $\langle V_2, 2 \rangle$ and between $\langle V_2, 3 \rangle$ and $\langle V_3, 3 \rangle$ because these conflicts appear in the original CN.

Finally, we recall the notions of directional arc and path-consistency.

Definition 2. *A value $\langle v_i, val \rangle$ is consistent with a set of values S if it is compatible with at least one value of S.*

Definition 3. *A CN Z is called directionally arc-consistent with respect to an order $v_1, \ldots, v_n$ of its variables if every value $\langle v_i, val \rangle$ is consistent with a domain of a variable v_k whenever $k < i$.*

Definition 4. *A pair of values $\{\langle v_i, val_i \rangle, \langle v_k, val_k \rangle\}$ is consistent with a set of values S if at least one value of S is compatible with both $\langle v_i, val_i \rangle$ and $\langle v_k, val_k \rangle$.*

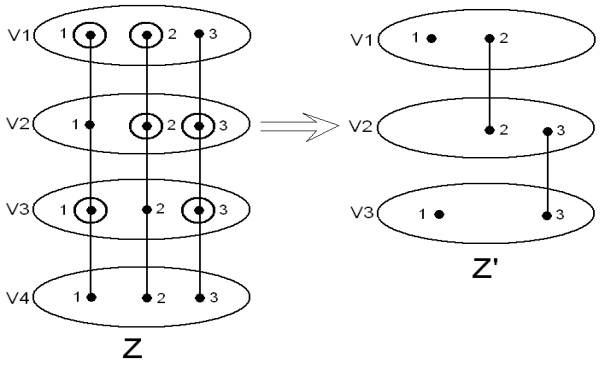

Fig. 1. Illustration of a CN induced by a set of values.

Definition 5. *A CN Z is called directionally path-consistent with respect to the order $v_1, \ldots, v_n$ of its variables if every pair of compatible values $\{\langle v_i, val_i\rangle, \langle v_k, val_k\rangle\}$ is consistent with the domain of a variable v_l whenever $l < i$ and $l < k$.*

3 The 2FC Algorithm

In this section we introduce a modification of FC that explores much smaller search space than FC. In particular, processing a CN with n variables and maximal domain size d, the algorithm generates a search tree with $O((d/2)^n)$ nodes if d is even and $O(((d+1)/2)^n)$ nodes if d is odd.

The first 3 steps of the modification are the following.

1. Variables are assigned with subsets of their current domains. In particular, let v be the variable being assigned currently. If the current domain of v is of size at least 2 then v is assigned with a subset S of its domain such that $|S| = 2$. Otherwise, if the domain is a singleton, v is assigned with the domain itself.
2. Once a variable v is assigned with a set S, the algorithm removes from the domains of unassigned variables all the values that are incompatible with all the values of S.
3. Unassigning a variable v at the backtrack stage, the algorithm removes from the current domain of v *all* values of the set assigned to v.

These 3 steps naturally generalize FC, providing the ability to assign a variable with one value as well as with two values. However the resulting algorithm has an essential drawback: when all variables are assigned, the CN induced by the assigned values may be insoluble and this is in contrast to the standard FC that has a solution when all variables are assigned. Thus the modified FC has a restricted ability of early recognition of dead-ends.

To illustrate this drawback, consider a CN with variables $\{v_1, v_2, v_3, v_4\}$ and the domain of every variable $\{1, 2, 3\}$. Every 2 variables are connected by the

inequality constraint. Assume that the algorithm assigns v_1 with $\{1, 2\}$. No value is deleted from the domains of the unassigned variables because there are values incompatible with 1, values incompatible with 2, but none that are incompatible with both of them. In the same way, the algorithm can assign v_2, v_3, v_4 with $\{1, 2\}$. The CN induced by the assigned values is clearly insoluble.

The following claim (proved in the next section) suggests a simple way to overcome the drawback.

Lemma 1. *Let Z be a 2-CN with no empty domain which is directionally arc-consistent and directionally path consistent with respect to an order $v_1, \ldots, v_n$ of variables of Z. Then Z is soluble.* [1]

Thus, to guarantee that whenever all the variables are assigned, the CN induced by the assigned values is soluble, it is enough to ensure that it is directionally arc-consistent and directionally path-consistent.

Note that directional arc-consistency is already ensured by the modification number 2 described above. To ensure directional path-consistency, it is possible to perform the following operation: *Every time a variable v is assigned with a set S, add a conflict between every pair P of compatible assignments of future variables such that P is inconsistent with S.*

We call the resulting algorithm 2FC. Algorithm 1 introduces its pseudocode.

The algorithm is presented in the form of a recursive procedure that gets a CN Z as input. In the first 6 lines the termination conditions are checked. In particular, if Z has no variables, the procedure returns $\emptyset$ (lines 1-3), if Z has a variable with the empty domain, the procedure returns $FAIL$.

In line 7 a variable u is selected. In line 8 a CN Z' is created by removing from Z the variable u and all its values. Before exploring the domain of u, the algorithm removes from it all values that conflict with domains of some other unassigned variable (line 9).

The loop described in lines 10-27 explores the domain of u. In lines 11-15 a subset S of the current domain of u is selected. The set S contains 2 values unless there is only one value in the current domain of u. After u is assigned by S, the algorithm removes from the domains of variables of Z' all values inconsistent with S (line 16). If the size of S is 2, the algorithm adds conflicts between compatible pairs of values of Z' that are inconsistent with S (lines 18-20). Then the function 2FC is applied recursively to Z' (line 22). If the output of the recursive application is not $FAIL$, the procedure returns the union of the output with S (line 24). (The operation is correct because the output of 2FC is either $FAIL$ or a set of values.) Otherwise, if the recursive application returns $FAIL$, the algorithm removes S from the domain of u (line 26) and starts a new iteration of the loop or finishes it if the domain of u is wiped out. In the latter case the algorithm returns $FAIL$ in line 28.

Note that when 2FC returns a set of values, a solution can from it by the process described in the proof of Lemma 1. (See the next section.)

[1] Note that the suggested sufficient condition of solubility of 2-CNs is weaker than path-consistency whose sufficiency is proved in [7].

Algorithm 1 FUNCTION 2FC(Z)

1: **if** $V_Z = \emptyset$ **then**
2: Return $\emptyset$
3: **end if**
4: **if** There is a variable with the empty domain **then**
5: Return $FAIL$
6: **end if**
7: Select a variable u
8: $Z' \leftarrow Z \setminus u$
9: Remove from $D_z(u)$ all values that are inconsistent with domains of unassigned variables
10: **while** $D_z(u) \neq \emptyset$ **do**
11: **if** $|D_z(u)| \geq 2$ **then**
12: $S \leftarrow \{val_1, val_2\}$, where val_1 and val_2 are two values of $D_z(u)$
13: **else**
14: $S \leftarrow D_z(u)$
15: **end if**
16: Remove from the domains of Z' the values that are inconsistent with S
17: **if** $|S| \leq 2$ **then**
18: **for** every pair $\{\langle v_1, val'_1\rangle, \langle v_2, val'_2\rangle\}$ of compatible values of Z' that is inconsistent with S **do**
19: $C_{Z'}(v_1, v_2) \leftarrow C_{Z'}(v_1, v_2) \setminus \{\{\langle v_1, val_1\rangle, \langle v_2, val_2\rangle\}\}$
20: **end for**
21: **end if**
22: $R \leftarrow 2FC(Z')$
23: **if** $R \neq FAIL$ **then**
24: Return $R \cup S$
25: **end if**
26: $D_Z(u) = D_Z(u) \setminus S$
27: **end while**
28: Return $FAIL$

Consider an example of application of 2FC. Let Z be the CN described above with variables $\{v_1, \ldots, v_4\}$, each domain equal $\{1, 2, 3\}$ and variables connected by inequality constraint. Assume that v_1 is assigned with $\{1, 2\}$. Then 2FC adds conflicts between every pair of values 1 and 2 of different unassigned variables, that is between $\langle v_2, 1\rangle$ and $\langle v_3, 2\rangle$, between $\langle v_2, 2\rangle$ and $\langle v_3, 1\rangle$ and so on. Assume that in the next iteration, variable v_2 is assigned with $\{1, 2\}$. Then values 1 and 2 are deleted from the current domains of v_3 and v_4. In the next iteration, trying to assign v_3, 2FC backtracks, because the only remaining values $\langle v_3, 3\rangle$ wipes out the current domain of v_4. As a result of backtrack, 2FC unassigns v_2. The only possible next assignment is $\{3\}$. After the assignment, 2FC removes 3 from the domains of v_3 and v_4. In the next iteration, 2FC tries again to assign v_3, but backtracks because both remaining values 1 and 2 wipe out the domain of v_4. This time, after unassigning v_2, the current domain of v_2 is finished; therefore 2FC backtracks again and changes the assignment of v_1 to $\{3\}$. In a few iteration the algorithm finishes with FAIL because of wiping out of the current domain of v_1.

A drawback of 2FC is large overhead spent to addition of conflicts between values of unassigned variables. It is not hard to show that $O(n^2d^2)$ additional consistency checks per iteration must be spent. The overhead can be reduced if we observe that 2FC checks compatibility of values of two variables only if one of these variables is either assigned or selected to be assigned.

Based on the observation we suggest a procedure of adding new conflicts based on the notion of *critical value*. Let u be a variable assigned with a set $\{val_1, val_2\}$. We say that $\langle u, val_1\rangle$ is critical with respect to a value $\langle v, val\rangle$ if $\langle v, val\rangle$ conflicts with $\langle u, val_2\rangle$. Instead of performing lines 18-19 in Algorithm 1, new conflicts can be added in the following "lazy" way. *Whenever a new variable v is selected to be assigned, a conflict is added between every pair of compatible values $\langle v, val\rangle, \langle w, val'\rangle$ that satisfies the following conditions:*

- *w is an unassigned variable other than u;*
- *val' belongs to the current domain of w;*
- *$\langle w, val'\rangle$ conflicts with at least one critical value with respect to $\langle v, val\rangle$.*

One can calculate that the suggested technique of updating of constraints takes $O(n^2d)$ consistency checks per iteration. We use the technique in our implementation of 2FC. We decided not to describe the method directly in the pseudocode because it reduces readability of the code and makes the correctness proof more complicated.

4 Theoretical Analysis.

In this section we prove correctness of 2FC. We start from proving Lemma 1.

Proof of Lemma 1 By induction on n, the number of variables of Z. It is trivial for $n = 1$. For $n > 1$, assign v_n with a value val_n that belongs to its domain. Let Z' be a 2-CN obtained from Z by removing v_n and deleting from the domains of the rest of variables all values that are incompatible with $\langle v_n, val_n\rangle$. Observe the following properties of Z'.

– The domains of all variables of Z' are not empty. Really, an empty domain of some variable v in Z' would mean that $\langle v_n, val_n \rangle$ conflicts with all the values of the domain of v in Z in contradiction to the directional arc-consistency of Z.
– Z' is directionally arc-consistent with respect to the order $v_1, \ldots, v_{n-1}$. Assume by contradiction that a value $\langle v_k, val \rangle$ is inconsistent with the domain of v_i, $i < k$. If the domain of v_i in Z' is the same as in Z, $\langle v_k, val \rangle$ is inconsistent with v_i in Z in contradiction to our assumption about directional arc-consistency of Z. Otherwise, one of the values of the domain of v_i is incompatible with $\langle v_n, val_n \rangle$, the other is incompatible with $\langle v_k, val \rangle$, while $\langle v_n, val_n \rangle$ and $\langle v_k, val \rangle$ are compatible. In this case we get contradiction with out assumption about directional path-consistency of Z.
– Z' is directionally path-consistent. For otherwise, if we have two compatible values $\langle v_k, val_k \rangle$ and $\langle v_l, val_l \rangle$ that wipe out the domain of some variable v_i, $(i < k, l)$, the same situation occurs in Z in contradiction to directional path-consistency of Z.

Thus Z' satisfies all conditions of the lemma and has $n-1$ variables, therefore it is soluble by the induction assumption. Let S be a solution of Z'. Note that all values of Z' are compatible with $\langle v_n, val_n \rangle$. Therefore $S \cup \{\langle v_n, val_n \rangle\}$ is a solution of Z. □.

The next lemma claims that every extended partial solution generated by 2FC satisfies the conditions of Lemma 1.

Lemma 2. *Every extended partial solution generated by 2FC induces a 2-CN without empty domains, directionally arc-consistent with respect to the chronological order of assignment of variables, and directionally path consistent with respect to the same order.*

Proof. By induction on the length n of the extended partial solution. It is clear that the lemma is valid for $n = 1$. For $n > 1$, let S be the considered extended partial solution and let $v_1, \ldots, v_n$ be the order according to which the variables were assigned. By the induction assumption, the extended partial solution obtained by removing v_n satisfies the conditions of the lemma. Therefore, if S violates these conditions then the values assigned to v_n violate either the directional arc-consistency or the directional path-consistency. Assume that the former holds. That is, a value val of assigned to v_n is inconsistent with all values assigned to v_k $(k < n)$. However, such a situation cannot happen because 2FC would remove val from the current domain of v_n when v_k has been assigned. For the latter, assume that a value val assigned to v_n, together with a compatible value val' assigned to some v_k are inconsistent with the set of values assigned to some v_i $(i < k, n)$. However, such a situation cannot happen as well because 2FC would add a conflict between $\langle v_k, val' \rangle$ and $\langle v_n, val \rangle$ when v_i was selected to be assigned. Thus the lemma holds for S. □

Now we are ready to claim the correctness of 2FC.

Theorem 1. *The 2FC algorithm is correct.*

Proof. To prove correctness, we have to prove that the algorithm terminates and also that it is sound and complete.

Termination is easy to verify by induction. If the underlying CN has 0 variables, the algorithm clearly terminates. Otherwise, 2FC selects a variable, assigns it with some partition class of its domain, and applies recursively to a CN created by the rest of variables (with updated constraints). By the induction assumption, every recursive application eventually finishes and also the number of partition classes in the first variable is finite so the algorithm terminates.

Soundness (solubility of an extended solution returned by 2FC) directly follows from Lemmas 1 and 2.

It follows from termination and soundness that 2FC always returns $FAIL$ when processes an insoluble CN. It remains to prove completeness, that is to show that 2FC always returns a solution when processes a soluble CN. In essence, all we have to show is that the additional conflicts generated by 2FC do not cause missing of a solution.

We prove completeness by induction on the number n of variables of the CN. Completeness follows immediately for $n = 1$. For $n > 1$, let v be the variable that 2FC selects to be assigned first. If the underlying CN is soluble then 2FC eventually assigns v with a set of values S that belongs to an extended solution. Then 2FC removes from the domains of the rest of variables all values that are inconsistent with S and adds conflicts between pairs of compatible values that wipe S out. Note that neither the removed values can be in the same solution with any value of S nor pairs of values that are made incompatible. Therefore 2FC is applied recursively to a soluble CN where it finds an extended solution by the induction assumption. □

5 Experimental Evaluation

It is not hard to show that 2FC explores $O(\lceil d/2 \rceil^n)$ nodes of the search tree and its running time is the bound multiplied by a polynomial. Clearly, this bound is much smaller than $O(d^n)$ upper bound for FC. However, we are interested to evaluate the practical merits of 2FC. To do this we compare in this section actual running times of 2FC and FC.

We implemented the algorithms in Microsoft Visual C++ 6.0 and tested them on a computer with CPU 2.4GHz and 0.2GB RAM. We used two measures of computation effort: the number of nodes visited and runtime (in seconds). For every tuple of parameters of the tested instances, the computation effort measures were obtained as average of 50 runs.

In our implementation, variables are ordered by the Fail-First heuristic [6] which takes first a variable with the smallest domain. The values of the variable being assigned are ordered according to the min-conflict heuristic, that is, values that conflict with the less number of values in the domains of unassigned variables are assigned first. (FC assigns the values one by one, while 2FC assigns them in pairs.)

We compared these algorithms on graph k-coloring problem and on randomly generated binary CNs.

Given a graph G with n vertices and k-colors, the CN that encodes the k-coloring problem for G has n variables corresponding to the vertices of G. The domain of every variable is $\{1, \ldots, k\}$. Pairs of variables that correspond to adjacent vertices of G are connected by the inequality constraint.

We generated 3 sets of instances: the first with 60 and 6 colors, the second with 45 vertices and 8 colors and the third with 30 vertices and 10 colors. For every set of instances we tried densities from 10% to 90% by steps of 5%.

The results of comparison of 2FC and FC on the first set of instances are shown on Figures 2 and 3. In this set of instances the phase transition region falls to the area of small density. Clearly, 2FC performs better than FC on this set of instances.

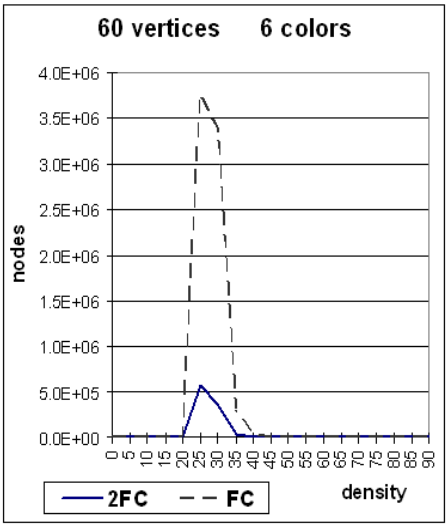

Fig. 2. 2FC vs. FC for graphs with 60 vertices and 6 colors (nodes visited)

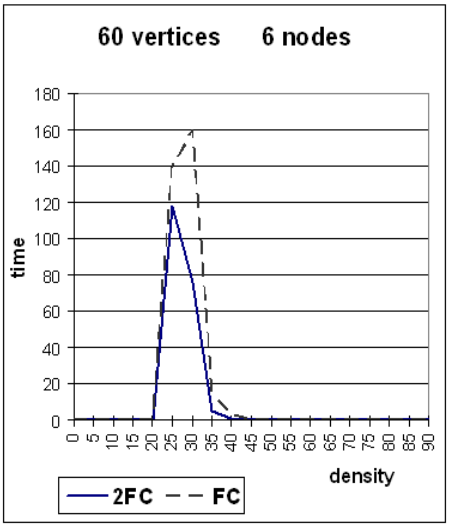

Fig. 3. 2FC vs. FC for graphs with 60 vertices and 6 colors (runtimes)

The results of comparison of 2FC and FC on the second set of instances are shown on Figures 4 and 5. In this experiment the graph that are most hard for coloring have an average density. Note that for denser graphs the rate of improvement of 2FC with respect to FC grows.

The results of comparison of 2FC and FC on the third set of instances are shown on Figures 6 and 7. In this experiments the phase transition region falls to the area of dense graphs. Note that here 2FC exhibits a larger factor of improvement as compared to the previous cases.

Comparing 2FC and FC on randomly generated CNs, we generated them using 4 parameters: the number of variables, the domain size, density, and tight-

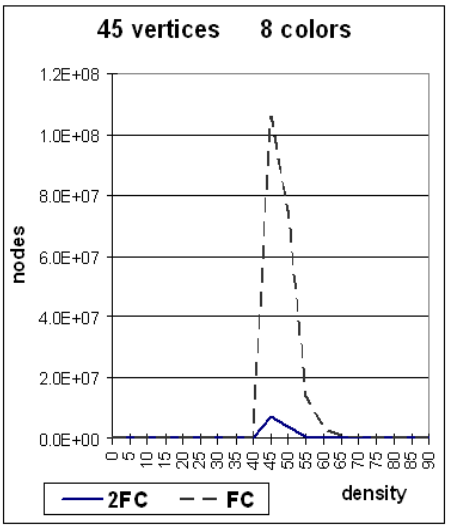

Fig. 4. 2FC vs. FC for graphs with 45 and 8 colors (nodes visited)

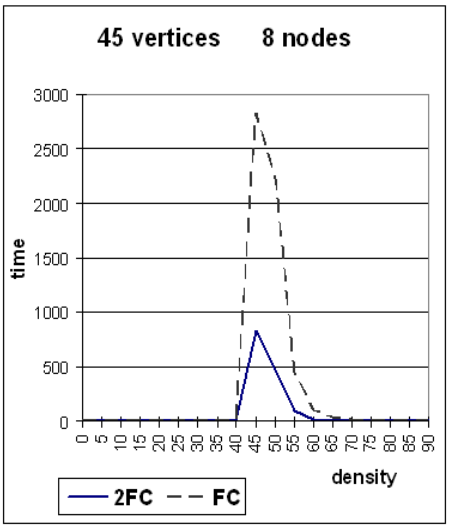

Fig. 5. 2FC vs. FC for graphs with 45 and 8 colors (runtimes)

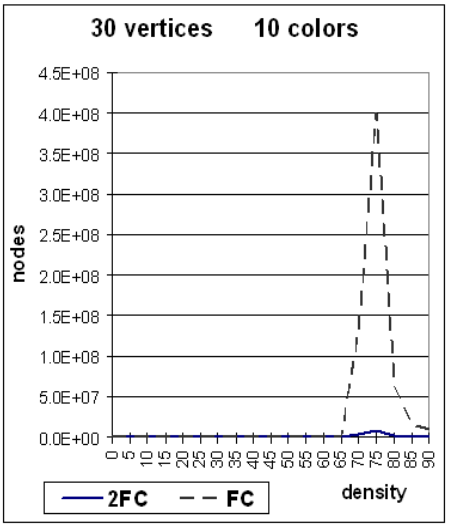

Fig. 6. 2FC vs. FC for graphs with 30 and 10 colors (nodes visited)

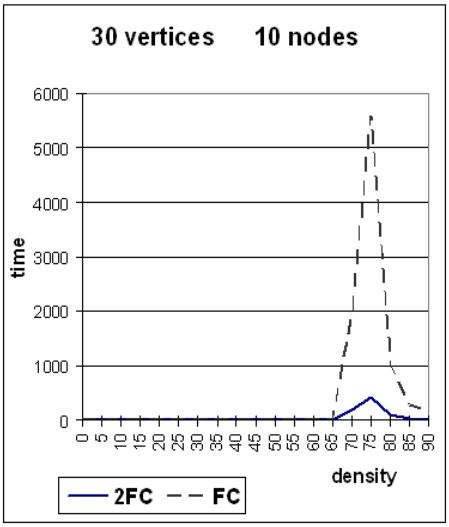

Fig. 7. 2FC vs. FC for graphs with 30 and 10 colors (runtimes)

ness [9]. To generate a set of instances, we fixed the number of variables, the domain size, and the density and varied the tightness from 10% to 90% by steps of 5%.

In the first set of experiments, the generated CNs have 60 variables domains of size 10 and density 10%. Figures 8 and 9 compare the number of nodes visited and the runtimes, respectively. We can see that 2FC outperforms FC on this set of instances.

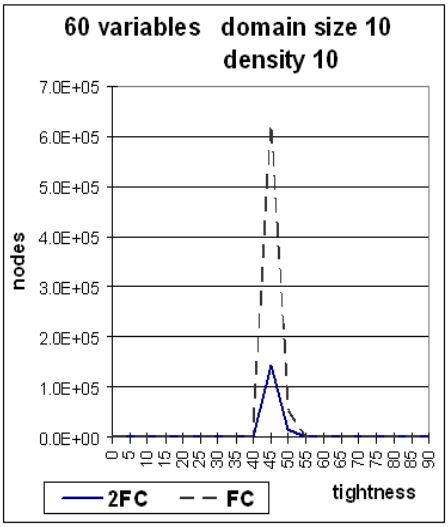

Fig. 8. 2FC vs. FC for CNs with 60 variables, domain size 10, and density 10 (nodes visited)

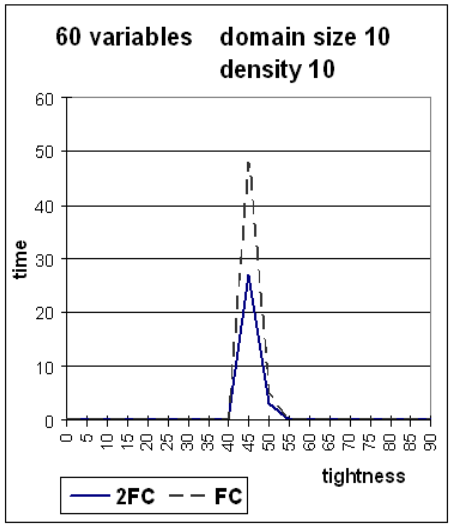

Fig. 9. 2FC vs. FC for CNs with 60 variables, domain size 10, and density 10 (runtimes)

Unfortunately, on denser instances of randomly generated CNs, 2FC works worse than FC. Moreover, 2FC becomes worse and worse compared to FC as the underlying CN gets denser. To see this, consider the following two sets of sets of experiments (Figures 10, 11, 12, and 13). On the set of instances with density 0.2, 2FC continues to perform better in the number of nodes visited while spends more runtime. However, on the instances with density 0.8, it looks worse with respect to the both measures.

Thus, according to our experiments, 2FC performs better than FC on graph coloring problems and non-dense instances of randomly generated CN, while works worse on denser random CNs.

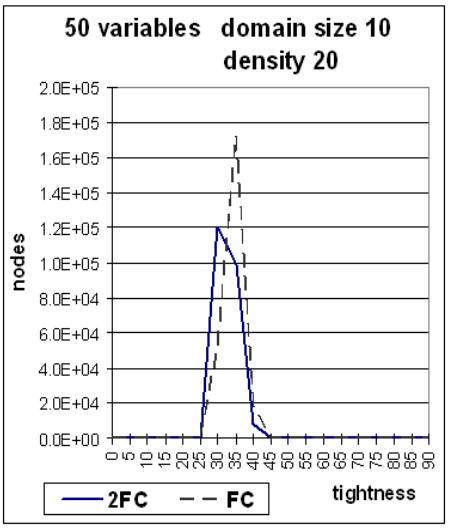

Fig. 10. 2FC vs. FC for CNs with 50 variables, domain size 10, and density 20 (nodes visited)

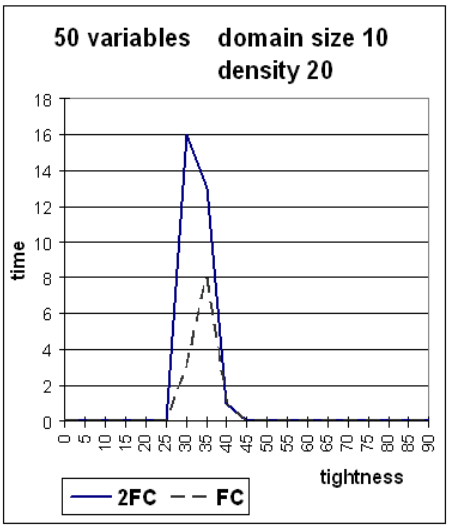

Fig. 11. 2FC vs. FC for CNs with 50 variables, domain size 10, and density 20 (runtimes)

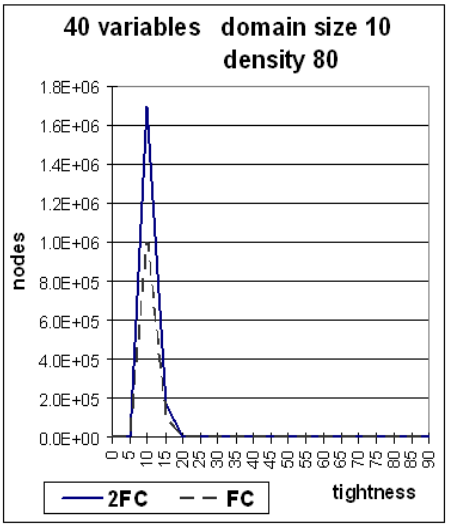

Fig. 12. 2FC vs. FC for CNs with 40 variables, domain size 10, and density 80 (nodes visited)

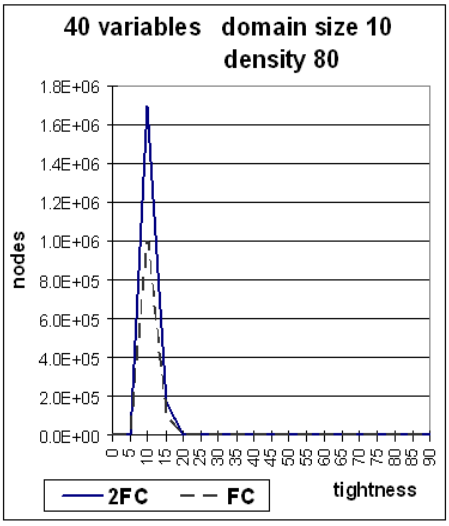

Fig. 13. 2FC vs. FC for CNs with 40 variables, domain size 10, and density 80 (runtimes)

6 Discussion

We introduced the 2FC algorithm which is based on the idea of assigning a variable with two values instead of one. In this section we discuss possible applications of the proposed approach and directions of further development.

According to our experimental results, 2FC performs very well on graph k-coloring problem. This result suggests the possibility of combining the proposed approach (of assigning a vertex with 2 colors) with branch-and-bound algorithms that find chromatic numbers of graphs (like [2]). The proposed approach could also be useful in the area of resource allocation problems because many of such problems, like timetabling [8], have binary constraint networks with inequality constraints.

On the other hand, 2FC is not very successful on random constraint networks. A natural way of improvement of its pruning ability is replacing FC by MAC, that is design of 2MAC. We expect that 2MAC would behave better with respect to MAC than 2FC does with respect to FC. This is because maintaining arc-consistency has a better ability than FC to utilize the conflicts that are added after every new assignment.

An interesting direction of further research is the application of the approach to CNs with non-binary constraints. Note that the application cannot be straightforward because solving a 2-CN with non-binary constraints is NP-complete in general (it can be shown by reduction from SAT). A possible way to recognize dead-ends early is maintaining the current extended partial solution S together with a solution T of the CN induced by S. Every time when S grows by assigning a new variable, T must grow also. Solving a 2-CN with non-binary constraints at every iteration of the algorithm could require too much time, therefore one has to develop heuristic methods of quick solving of such CNs.

2FC could also be useful in dynamic environments where solutions are frequently discarded because of updating of constraints. The set of values returned by 2FC can contain many solutions and they can be efficiently extracted. Therefore, there is a chance that once a single solution is discarded, another solution could be found instantly without applying search.

Finally, there is an intriguing connection of the proposed approach with the technique of bucket elimination [3]. In its simplest form, the principle of bucket elimination states that whenever there is an unassigned variable v conflicting with at most two other unassigned variables, variable v can be eliminated. To preserve consistency, conflicts must be added between the pairs of values that wipe out the current domain of v. Note that the described bucket elimination technique as well as assigning a variable with two values have the following common paradigm: *assign a variable v with a subset s of its domain such that every minimal consistent partial solution on the "future" variables that wipes s out has the size at most 2.* Bucket elimination is an "extremal" realization of the paradigm where a variable is assigned with the whole domain. Assigning a variable with only one value is another case of extremal realization. Then assigning a variable with two values can be considered as some "intermediate" case. Continuing the reasoning, we derive that there may be other "intermediate"

realizations of the paradigm. For example, a more flexible version of bucket elimination can be considered, where a variable is assigned with a subset of its values that conflict with at most two unassigned variables.

References

1. O. Angelsmark and P. Jonsson. Improved algorithms for counting solutions in constraint satisfaction problems. In *CP 2003*, pages 81–95, 2003.
2. M. Caramia and P. Dell'Olmo. Constraint propagation in graph coloring. *Journal of Heuristics*, 8:83–107, 2002.
3. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.
4. R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
5. D. Eppstein. Improved algorithms for 3-coloring, 3-edge coloring and constraint satisfaction. In *SODA-2001*, pages 329–337, 2001.
6. R. M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
7. P. Jeavons, D. Cohen, and M. Cooper. Constraints, consistency, and closure. *Artificial Intelligence*, 101:251–265, 1998.
8. A. Meisels and A. Schaerf. Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence*, 39:41–59, 2003.
9. P. Prosser. An empirical study of phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.