Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 904768, 10 pages
http://dx.doi.org/10.1155/2013/904768

Research Article

New Algorithms for Bidirectional Singleton Arc Consistency

Yonggang Zhang,[1,2] Qian Yin,[3] Xingjun Zhu,[1,2] Zhanshan Li,[1,2] Sibo Zhang,[1,2] and Quan Liu[2,4]

[1] *College of Computer Science and Technology, Jilin University, Changchun 130012, China*
[2] *Key Laboratory of Symbol Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China*
[3] *College of Computer Science and Information Technology, Northeast Normal University, Changchun 130117, China*
[4] *School of Computer Science and Technology, Soochow University, Jiangsu 215006, China*

Correspondence should be addressed to Yonggang Zhang; zhangyg@jlu.edu.cn

Received 23 June 2013; Accepted 4 September 2013

Academic Editor: William Guo

Bidirectional singleton arc consistency (BiSAC) which is an extended singleton arc consistency (SAC) has been proposed recently. The first contribution of this paper is to propose and prove two theorems of BiSAC theoretically (one is a property of BiSAC and the other is the property of allowing the deletion of some BiSAC-inconsistent values). Secondly, based on these properties we present two algorithms, denoted by BiSAC-DF and BiSAC-DP, to enforce BiSAC. Also, we prove their correctness and analyze the space and time complexity of them in detail. Besides, for special circumstances, we show that BiSAC-DF admits a worst-case time complexity in $O(en^2d^4)$ and a best one in $O(en^2d^3)$ when the problem is an already BiSAC, while BiSAC-DP also has the same best one when the tightness is small. Finally, experiments on a wide range of CSP instances show BiSAC-DF and BiSAC-DP are usually around one order of magnitude faster than the existing BiSAC-1. For some special instances, BiSAC-DP is about two orders of magnitude efficient.

1. Introduction

Constraint satisfaction problem (CSP) is an important researching area in artificial intelligence. Many combinatorial problems can be solved by modeling as CSP. However, it is the NP-complete task of determining whether a given CSP instance has a solution. Fortunately, consistency techniques can reduce the search space and speed up solving process by removing the inconsistent values [1]. Many notions and algorithms of consistency have been proposed to achieve these goals so far. Among these techniques, arc consistency (AC) [2] is the oldest and most well-known way of filtering domain. This is indeed very simple and natural. More importantly, it is an effective technique to solve CSP instances. Now there are many algorithms to enforce it. For example AC-3 [3], AC-4 [4], and AC-2001 [5] are all very efficient methods and they are very easy to implement. Besides, nowadays singleton arc consistency (SAC) [6] has been focused on widely. It can remove redundant values like arc consistency and path consistency [7], but the power of pruning the inconsistent values is stronger than that of AC. In fact, it ensures that the problem is also arc consistent after assigning any value to any variable. Debruyne and Bessière proposed a straightforward algorithm SAC-1 [6], and Prosser studied SAC again [8]. Consiquently, the second algorithm SAC-2 [9] has been proposed by Bartok. It is based on AC-4 and uses external data structures to avoid invoking the AC procedure to test SAC condition more times than SAC-1. Bessière and Debruyne presented the algorithms SAC-OPT [10] and SAC-SDS [11]. SAC-OPT has the optimal time complexity with a high space complexity. SAC-SDS has a better space complexity though it is not optimal in time. However, it tries to avoid redundant work. Lecoutre proposed the first depth-first greedy algorithm SAC-3 [12]. Bessière and Debruyne (2008) studied the comprehensive theoretical of SAC and proposed a new local consistency—bidirectional singleton arc consistency (BiSAC) [13] by extending the SAC. They have

also proved BiSAC is stronger than SAC and presented the first algorithm BiSAC-1 to enforce it. Zhang et al. proposed the algorithms BiSAC-DF and BiSAC-DP (Algorithms 4 and 5), both of them were the enforced algorithms of BiSAC.

In this paper, we firstly propose two properties of BiSAC and then prove their correctness. Based on them, we present two algorithms to enforce BiSAC, denoted by BiSAC-DF and BiSAC-DP. BiSAC-DF is inspired by algorithm SAC-3 proposed by Lecoutre, and it is also a depth-first greedy algorithm. The BiSAC-DP also uses the previous properties as a theoretical foundation and combines domain partition and divide-and-conquer strategy. Besides, we propose operator ρ on subdomain to improve the efficiency of it. Also we prove their correctness and analyze their complexity. Finally, in our experiments on random CSPs, classical problems, and benchmarks, the results show that our algorithms perform better than the BiSAC-1.

In terms of complexity, BiSAC-DF and BiSAC-DP have a space complexity in $O(nd)$ and a worst-case time complexity in $O(en^3d^5)$, where n is the number of variables, d is the greatest domain size, and e is the number of constraints. In particular, when applied to an already bidirectional singleton arc consistent problem, BiSAC-DF admits the worst-case time complexity in $O(en^2d^4)$ and a best one in $O(en^2d^3)$. BiSAC-DP has a best-case time $O(en^2d^3)$ if the tightness of the problem is relatively small. Notice that it does not mean the problem is BiSAC if the tightness of it is small. Actually, the problem which is an already BiSAC may have a high tightness.

2. Preliminaries

A constraint satisfaction problem $P = (V, D, C)$, where V is a finite set of n variables, D is a finite set of domains for every variable, and C is a finite set of e constraints. For each variable $X \in V$, its domain is denoted by $D(X) \in D$. m will denote the size of the greatest domain. We can use the label D^P to represent the domain set of the problem P. Each $c_i \in C$ has an associated relation denoted by $\text{rel}^P(c_i)$ which represents the set of tuples allowed for the variables involved in c_i. We can say the tightness of P is small if $|\text{rel}^P(c_i)|$ is large enough for $\forall c_i \in C$, respectively. Problem Q is a subproblem of P if $\forall X \in V$, $D^Q \subseteq D^P$and $\forall c_i \in C$, $\text{rel}^Q(c_i) \subseteq \text{rel}^P(c_i)$. A solution of a constraint satisfaction problem is an assignment of value from its domain to each variable such that every constraint is satisfied. A problem is said to be satisfiable if it admits at least one solution. We denote $P = \perp$ if there is a variable X of P and $D^P(X)$ wipes out.

A constraint satisfaction problem can be characterized by properties called partial consistency. It always removes some inconsistent values and some inconsistent pairs of values. There are many methods to do this. Now, we introduce several important notions. In this paper, we focused on binary CSP.

Definition 1. The value a of variable X is arc consistent (AC) if and only if, for each variable Y connected by the constraint c, there exists a value b of Y such that $(a, b) \in \text{rel}^P(c)$. The CSP is arc consistent if and only if every value of every variable is arc consistent.

We use AC(P) to represent the problem which is obtained after enforcing AC on a given problem P. If there is a variable with an empty domain in AC(P), denote AC(P) = $\perp$.

Definition 2. The value a of variable X is singleton arc consistent (SAC) if and only if the problem restricted to $X = a$ is arc consistent ($\text{AC}(P|_{X=a}) \neq \perp$). The CSP is singleton arc consistent if and only if every value of every variable is singleton arc consistent.

From the definition of AC and SAC, we can clearly see SAC is stronger than AC. In 2008, Bessière extended SAC to a stronger level of local consistency and proposed a new technique—bidirectional SAC (BiSAC) [13]. Now we introduce BiSAC below.

Definition 3. The value a of variable X is bidirectional singleton arc consistent (BiSAC) if and only if $\text{AC}(T_{Xa}) \neq \perp$, $T_{Xa} = (V, D_{Xa}, C)$, where $D_{Xa}(Y) = \{b \in D^P(Y) \mid (X, a) \in \text{AC}(P|_{Y=b})\}$. The CSP is bidirectional singleton arc consistent if and only if every value of every variable is bidirectional singleton arc consistent.

In [13] Bessière proposed a straightforward algorithm BiSAC-1 to enforce bidirectional singleton arc consistency. Its correctness and its space and time complexity are $O(1)$ and $O(en^3d^5)$, respectively (see Algorithm 1).

3. New Algorithms to Enforce BiSAC

In this paper, we propose two algorithms to enforce BiSAC. Before describing them, we need to present two important theorems of BiSAC. They not only provide a theoretical basis for correctness of our algorithms, but also can improve efficiency for our algorithms.

3.1. Theoretical Analysis of BiSAC

Theorem 4. *Let $P = (V, D^P, C)$ be a CSP and let Q be a subproblem of P. Given $(X, a) \in Q$,*

(1) (X, a) is bidirectional singleton arc consistent in P, if (X, a) is bidirectional singleton arc consistent in Q;

(2) (X, a) is bidirectional singleton arc consistent in P, if $AC(T_{Xa}) \neq \perp$, where $T'_{Xa} = (V, D'_{Xa}, C)$, $D'_{Xa}(Y) = \{b \in D^Q(Y) \mid (X, a) \in AC(P|_{Y=b})\}$.

Proof. (1) Let $T''_{Xa} = (V, D''_{Xa}, C)$, where $D''_{Xa}(Y) = \{b \in D^Q(Y) \mid (X, a) \in \text{AC}(Q|_{Y=b})\}$. Because (X, a) is BiSAC in Q, thus $\text{AC}(T''_{Xa}) \neq \perp$ is clearly concluded by the definition of BiSAC. For every $Y \in V$, the fact that Q is a subproblem of P can produce $D^Q(Y) \subseteq D^P(Y)$. Moreover, according to the correctness of AC procedure, for $(Y, b) \in Q$, if $(X, a) \in \text{AC}(Q|_{Y=b})$, then $(X, a) \in \text{AC}(P|_{Y=b})$. Above all, it implies $D''_{Xa}(Y) \subseteq D_{Xa}(Y)$, where $D_{Xa}(Y) = \{b \in D^P(Y) \mid (X, a) \in \text{AC}(P|_{Y=b})\}$. Let $T_{Xa} = (V, D_{Xa}, C)$; hence, T''_{Xa} is a

```
(1)  repeat
(2)      CHANGE ← false;
(3)      foreach (X, a) ∈ D^P do
(4)          D^T ← ∅;
(5)          foreach (Y, b) ∈ D^P do
(6)              if (X, a) ∈ AC(P|_{Y=b}) then D^T ← D^T ∪ {(Y, b)};
(7)          if AC(V, D^T, C) = ⊥ then
(8)              D(X) ← D(X)\{a};
(9)              if D(X) = ∅ then return false;
(10)             CHANGE ← true;
(11) until not CHANGE;
(12) return true;
```

Algorithm 1: BiSAC-1.

subproblem of the problem T_{Xa}. According to the correctness of AC procedure again, $\mathrm{AC}(T''_{Xa}) \neq \perp$ implies $\mathrm{AC}(T_{Xa}) \neq \perp$. So (X, a) is also bidirectional singleton arc consistent in P by the definition of BiSAC.

(2) Let $T_{Xa} = (V, D_{Xa}, C)$, where $D_{Xa}(Y) = \{b \in D^P(Y) \mid (X, a) \in \mathrm{AC}(P|_{Y=b})\}$. Thus, T'_{Xa} is a subproblem of T_{Xa}. Now, using the correctness of AC and the fact that $\mathrm{AC}(T'_{Xa}) \neq \perp$, $\mathrm{AC}(T_{Xa}) \neq \perp$ is deduced correspondingly. Hence conclusion (2) is obtained by the definition of BiSAC. □

Actually conclusion (1) exploits BiSAC of the subproblem Q to ensure that (X, a) is BiSAC in P. So it can avoid revoking the redundant AC procedure (see Definition 3). Conclusion (2) can be a similar result through a combination of definition of BiSAC and subproblem Q. It is not enough although Theorem 4 as a property of BiSAC can conclude that (X, a) is BiSAC in P. We also need another property to ensure (X, a) is bidirectional singleton arc inconsistent in P.

Theorem 5. *Let $P = (V, D^P, C)$ be a CSP and $(X, a) \in P$.*

(1) (X, a) is bidirectional singleton arc inconsistent in P, if $AC(P|_{X=a}) = \perp$.

(2) (X, a) is bidirectional singleton arc inconsistent in P, if $Q = AC(P|_{X=a}) \neq \perp$ and $AC(T'_{Xa}) = \perp$, where $T'_{Xa} = (V, D'_{Xa}, C)$, $D'_{Xa}(Y) = \{b \in D^Q(Y) \mid (X, a) \in AC(P|_{Y=b})\}$.

Proof. (1) Let $T_{Xa} = (V, D_{Xa}, C)$, where $D_{Xa}(Y) = \{b \in D^P(Y) \mid (X, a) \in \mathrm{AC}(P|_{Y=b})\}$ and $R = P|_{X=a}$. It means that $D_{Xa}(X) = D^R(X) = \{a\}$ and $D_{Xa}(Y) \subseteq D^P(Y) = D^R(Y)$ for every $Y \in V$ s.t. $Y \neq X$. Thus, according to the correctness of AC, $\mathrm{AC}(R) = \mathrm{AC}(P|_{X=a}) = \perp$ can lead to $\mathrm{AC}(T_{Xa}) = \perp$. Therefore, (X, a) is bidirectional singleton arc inconsistent in P by the definition of BiSAC.

(2) Let $S = \{(Z, c) \mid (Z, c) \in R = P|_{X=a} \bigwedge (Z, c) \notin Q = \mathrm{AC}(P|_{X=a})\}$; it means that every value in S is arc inconsistent when the problem P is restricted to $X = a$. Besides, let $T_{Xa} = (V, D_{Xa}, C)$ where, $D_{Xa}(Y) = \{b \in D^P(Y) \mid (X, a) \in \mathrm{AC}(P|_{Y=b})\}$. It implies $D_{Xa}(X) = D'_{Xa}(X) = \{a\}$ and $D'_{Xa}(Y) \subseteq D_{Xa}(Y)$ for every Y in V and $Y \neq X$. Thus every element in $D_{Xa} \backslash D'_{Xa}$ must be in $D^R \backslash D^Q$, so $D_{Xa} \backslash D'_{Xa} \subseteq S$ and T'_{Xa} is a subproblem of T_{Xa}. However, every value in $D_{Xa} \backslash D'_{Xa}$ is arc inconsistent in T_{Xa} by the correctness of AC. Again, this correctness and $\mathrm{AC}(T'_{Xa}) = \perp$ will lead to $\mathrm{AC}(T_{Xa}) = \perp$. Thus (X, a) is bidirectional singleton arc inconsistent in P. So, conclusion (2) is obtained. □

Theorem 5 is a property of bidirectional singleton arc inconsistency. Conclusion (1) of it, using the fact that problem P restricted to $X = a$ is arc inconsistent, can infer (X, a) is bidirectional singleton arc inconsistent in P. Another conclusion also can tell us (X, a) is bidirectional singleton arc inconsistent in P through arc inconsistency of a special subproblem. Actually, these theorems together provide a theoretical basis for soundness and completeness of our algorithms. More importantly, they are also approaches to speed up the efficiency of consistency technique.

Now, we propose two algorithms to enforce BiSAC, denoted by BiSAC-DF and BiSAC-DP, which rely on Theorems 4 and 5.

3.2. BiSAC-DF Algorithm. The first algorithm BiSAC-DF is inspired by SAC-3, which is also depth-first greedy approach. Moreover the above theorems can improve efficiency and ensure correctness of BiSAC-DF. The function BiSAC-DF firstly puts all variable-value pairs in the structure *Queue* and then calls the function Judge-DF to establish BiSAC. If the Judge-DF returns $\perp$, it means there exists a domain that wipes out. So the BiSAC returns false. Otherwise, this processing continues until a fix-point is reached.

The function Judge-DF performs a depth-first search in order to instantiate variables, but it does not have a backtrack mechanism when dead ends occur. As long as, the current branch, no inconsistency is found, we try to extend it. In other words, the current problem BP is restricted to $X = a$ and then arc consistency is maintained. So a subproblem $\mathrm{AC}(BP|_{X=a})$ is generated correspondingly. The other value pairs will be judged whether they are BiSAC on this current subproblem. Theorem 4 concludes that (X, a) is BiSAC in P if it is BiSAC in sub-problem or $\mathrm{AC}(BP|_{X=a}) \neq \perp$ (line 15 and line 25). In particular, it maybe finds a lucky solution (line 18). For bidirectional singleton arc inconsistency, a dead end must occur. In that case, Theorem 5 can tell us that (X, a) is bidirectional singleton arc inconsistent in P if the structure

occur is null (line 6 and line 14). Therefore, the value a should be removed from the domain of the variable X. If this domain wipes out, the algorithm terminates and returns $\perp$ (line 23). Otherwise, we cannot judge whether (X, a) is BiSAC or not. So, it needs further propagations.

Theorem 6. *BiSAC-DF is a correct algorithm to enforce bidirectional singleton arc consistency.*

Proof. Consider the following.

Soundness. Suppose (X, a) is the first BiSAC value removed by BiSAC-DF. It is necessarily removed in line 22 of the function Judge-DF. It means that the structure *occur* is $\emptyset$, and at this time it goes to line 22 only from line 6 or line 14. Hence, if it jumps from line 6, it implies $\mathrm{AC}(BP|_{X=a}) = \perp$. The main loop in Judge-DF was executed only once because the structure *occur* is $\emptyset$, so $BP = P$. Therefore, (X, a) is bidirectional singleton arc inconsistent in P according to conclusion (1) of Theorem 5. Jumping from line 14, it is implied that $Q = \mathrm{AC}(BP|_{X=a}) \neq \perp$ and $\mathrm{AC}(T^{Q}) = \perp$; conclusion (2) of Theorem 5 admits that (X, a) is also bidirectional singleton arc inconsistent in P. Therefore, this contradicts the assumption that (X, a) is the first BiSAC value removed by BiSAC-DF. So, BiSAC-DF is sound.

Completeness. When the BiSAC-DF terminated, for every $(X, a) \in D$, (X, a) is removed from *Queue* if the clause repeat (in Judge-DF) is executed for the first time. It means $\mathrm{AC}(BP) \neq \perp$, where $BP = (V, D^{BP}, C)$, $D^{BP}(Y) = \{b \in D^{P}(Y) \mid (X, a) \in \mathrm{AC}(P|_{Y=b})\}$. According to the definition of BiSAC, (X, a) is bidirectional singleton arc consistent in P. Otherwise, it is concluded that $\mathrm{AC}(BP) \neq \perp$, where $BP = (X, D^{BP}, C)$, $D^{BP}(Y) = \{b \in D^{Q}(Y) \mid (X, a) \in \mathrm{AC}(P|_{Y=b})\}$. Theorem 4 can ensure that (X, a) is also bidirectional singleton arc consistent in P. Thus, BiSAC-DF is complete. □

Theorem 7. *BiSAC-DF admits a space complexity in $O(nd)$ and a worst-case time complexity in $O(en^3d^5)$.*

Proof. The additional data structure in BiSAC-DF is *Queue* and *BP*, while the space of them is also $O(nd)$, so the overall space complexity of BiSAC-DF is $O(nd)$.The worst-case of running BiSAC-DF is that there is exactly one value removed from its domain when the loop (see line 5 in Algorithm 2) terminates. In that case, the number of elements in *Queue* is $(nd - i + 1)$ when the loop repeat is executed i $(1 \leq i \leq nd)$ times. Notice that the number of variable-value pairs in P is also $(nd - i + 1)$. Let k be the total times of calling Judge-DF at i iterative times in the loop repeat (in BiSAC-DF), and let del_j be the number of elements removed from *Queue* at j times of calling Judge-DF. From the function Judge-DF, it is easily concluded that the times of calling AC are at most $(nd - i + 1)$ when a value is removed from *Queue*. Thus, at i iterative times in the loop repeat (in BiSAC-DF), the total number of calling AC is

$$\sum_{j=1}^{j=k} \mathrm{del}_j \times (nd - i + 1) = (nd - i + 1)^2. \quad (1)$$

```
(1) repeat
(2)     CHANGE ← false;
(3)     foreach (X, a) ∈ P do
(4)         Queue ← Queue ∪ {(X, a)};
(5)     while Queue ≠ ∅ do
(6)         if Judge-DF() = ⊥ then return false;
(7) until not CHANGE
(8) return true;
```

Algorithm 2: BiSAC-DF.

Therefore, the times of AC called by BiSAC-DF are $(nd)^2 + (nd-1)^2 + \cdots + (nd-i+1)^2 + \cdots + 1^2 = 1/6 \times (nd) \times (nd+1) \times (2nd+1)$. If an optimal AC is used, then the time complexity is $O(ed^2)$, such as AC-2001. So, the worst-case time complexity of BiSAC-DF is $1/6 \times (nd) \times (nd+1) \times (2nd+1) \times O(ed^2)$, namely, $O(en^3d^5)$. □

Theorem 8. *Applied to a constraint network which is BiSAC, the worst-case time complexity of BiSAC-DF is $O(en^2d^4)$ and the best-case time complexity is $O(en^2d^3)$.*

Proof. When it is applied to a BiSAC network, there will be no value removed. It implies that the loop repeat (in Algorithm 2) is executed only once. During this procedure, the number of elements in *Queue* is nd. For every element in *Queue*, in the worst-case, it needs call AC $2 \times nd$ times and the total is $2 \times (nd)^2$. Thus, worst-case time complexity is $2 \times (nd)^2 \times O(ed^2) = O(en^2d^4)$. For the best case, it only executes Judge-DF exactly once (finding a lucky solution). So the number of calling AC is $d(n - i) + i$ at i $(1 \leq i \leq n)$ iterative times in the loop repeat (in Algorithm 3). Note that the number of elements in *BP* is $d(n-i)+i$. Thus, the total of calling AC is

$$\sum_{i=1}^{i=n} (d(n-i)+i) = n/2 \times (n(d+1) + (1-d)). \quad (2)$$

So the best-case time complexity is $n/2 \times (n(d+1) + (1-d)) \times O(ed^2) = O(en^2d^3)$. □

3.3. BiSAC-DP Algorithm. Another algorithm BiSAC-DP is also based on the previous propositions, but it is not depth-first greedy approach. So it cannot find a lucky solution. However, it combines domain partition technique and uses the operator ρ presented by us to find a special subproblem so that it saves lots of redundant work. Now we propose operator ρ: let $P = (V, D, C)$; operator $\rho(P, X)$ must remove every value b from the domain of every variable Y connected with X by the constraint c, where the value b s.t. $\exists a \in D(X)$ such that $(a, b) \notin \mathrm{rel}^{P}(c)$. During removing values, if a domain wipes out, operator ρ returns $\perp$; otherwise, it returns the new problem. From the definition, we know operator ρ can run in $O(nd^2)$ at worst without additional data structure. Actually, if $|D(X)| = 1$, the function of operator ρ is the

```
(1)  occur ← ∅;
(2)  BP ← P;
(3)  repeat
(4)      get (X, a) from Queue s.t. X ∉ vars(occur) and (X, a) ∈ BP;
(5)      BP ← AC(BP|_{X=a});
(6)      if BP = ⊥ then goto L;
(7)      flag ← false;
(8)      foreach (Y, b) ∈ BP do
(9)          if (X, a) ∉ AC(P|_{Y=b}) then
(10)             BP ← BP \ {(Y, b)}
(11)             flag ← true;
(12)     if flag = true then
(13)         BP ← AC(BP|_{X=a});
(14)         if BP = ⊥ then goto L;
(15)             else occur ← occur ∪ {(X, a)};
(16)     else occur ← occur ∪ {(X, a)};
(17) until vars(occur) = vars(P);
(18) return Solution;
(19) L:
(20) if occur = ∅ then
(21)     CHANGE ← true;
(22)     P ← P \ {(X, a)};
(23)     if P = ⊥ then return ⊥;
(24) else
(25)     Queue ← Queue \ occur;
(26)     Queue ← Queue ∪ {(X, a)};
```

Algorithm 3: Judge-DF.

```
(1) repeat
(2)     CHANGE ← false;
(3)     foreach Y ∈ V do
(4)         D_1 :: D_2 ← D(Y);
(5)         if Judge-DP(Y, D_1) = ⊥ ‖ Judge-DP(Y, D_2) = ⊥ then
(6)             return false;
(7) until not CHANGE;
(8) return true;
```

Algorithm 4: BiSAC-DP.

same as forward checking (FC). For $|D(X)| > 1$, it can be considered as the extended forward checking. Note that the operator ρ is only used to improve efficiency through finding an appropriate special subproblem. However $\rho(P, X) = \perp$ when $|D(X)| > 1$; we cannot say that the current problem has no solutions. Actually, in that case all values in $D(X)$ are also necessary to be propagated for BiSAC. So, a "good" domain $D(X)$ for $\rho(P, X) \neq \perp$ is very important for efficiency of our algorithm. Therefore, we introduce divide-conquer strategy to automatically find an appropriate subdomain.

The function BiSAC-DP partitions the domain of every variable into two subdomains firstly and then calls the function Judge-DP to establish the BiSAC. This process terminates until the fix-point is reached. It means that every value in the current subdomain D_X is bidirectional singleton arc consistent when function Judge-DP returns true. Otherwise, it means that a domain has wiped out when it returns $\perp$.

The main function of Judge-DP is to determine whether all values in subdomain D_X are BiSAC in the special subproblem. If they are all BiSAC, then they return true; otherwise it needs to partition the current subdomain. Firstly, Judge-DP uses operator ρ to find the special subproblem. If $\rho(Q, X) = \perp$ (line 4), the function would go to label L. Otherwise, Judge-DP calls the AC. It would still go to L if AC returns $\perp$ too. Otherwise, it enforces BiSAC on this current problem (line 6–line 10). Theorem 4 can ensure that it is BiSAC in P if it is BiSAC in the current subproblem for every value in subdomain D_X (line 10). However, if not, it also needs to go to L for further processing. In label L, according to the definition of BiSAC, if $|D_X| = 1$, the single value in D_X is bidirectional singleton arc inconsistent, so it must be removed (line 13). However, if $|D_X| \neq 1$, the current subdomain D_X needs to be partitioned further (line 18).

```
(1)  Q ← P;
(2)  D^Q(X) ← D_X;
(3)  if |D^Q(X)| = 0 then return true;
(4)  if ρ(Q, X) = ⊥ then goto L;
(5)  if AC(Q) = ⊥ then goto L;
(6)  foreach (Y, b) ∈ Q s.t Y ≠ X do
(7)      if AC(P|_{Y←b}) ≠ ⊥ then
(8)          if exist a ∈ D_X s.t. (X, a) ∉ AC(P|_{Y=b}) then
(9)              Q ← Q \ {(Y, b)};
(10)  if AC(Q) ≠ ⊥ then return true;
(11) L:
(12)  if |D_X| = 1 then
(13)      P ← P \ {(X, D_X)};
(14)      CHANGE ← true;
(15)      if P = ⊥ then return ⊥;
(16)          else return true;
(17)  D_1 :: D_2 ← D_X;
(18)  if Judge-DP(X, D_1) = ⊥ ‖ Judge-DP(X, D_2) = ⊥ then return ⊥;
(19)  return true;
```

Algorithm 5: Judge-DP (X, D_X).

Theorem 9. *BiSAC-DP is a correct algorithm to enforce bidirectional singleton arc consistency.*

Proof. Consider the following.

Soundness. Suppose (X, a) is the first BiSAC value removed by BiSAC-DP. It is necessarily removed in line 13 of the function Judge-DP. It implies that the subdomain D_X only contains one value $a(D_X = \{a\})$. From the Judge-DP, it goes to line 13 only through line 4, line 5, or line 10 (in this case, $\mathrm{AC}(Q) = \bot$). If it goes through line 5, then $\mathrm{AC}(Q) = \bot$ and $Q = P|_{X=a}$. So conclusion (1) of Theorem 5 admits (X, a) is bidirectional singleton arc inconsistent in P. Through line 4, $\mathrm{AC}(Q) = \bot$ and $Q = P|_{X=a}$ can also be concluded because $D_X = \{a\}$ and $\rho(Q, X) = \bot$. So the same result can be obtained. Otherwise, from line 10 and $\mathrm{AC}(Q) = \bot$, in this case, according to conclusion (2) in Theorem 5, (X, a) is also bidirectional singleton arc inconsistent in P. Above all, this contradicts the assumption. So, BiSAC-DP is sound.

Completeness. When the BiSAC-DP terminated, for every $(X, a) \in D$, (X, a) is checked or D_X is checked for BiSAC, where D_X is subdomain of D and $(X, a) \in D_X$. If (X, a) is checked, then (X, a) is BiSAC in P by Theorem 4. Otherwise, the same result is also obtained by the definition of operator ρ and Theorem 4. Thus, BiSAC-DP is complete. □

Theorem 10. *BiSAC-DP admits a space complexity in $O(nd)$ and a time complexity in $O(\lambda end^3)$, where λ denotes the number of partitions built by the BiSAC-DP.*

Proof. The BiSAC-DP has only the additional data structure Q and the space complexity of it is $O(nd)$. So the space complexity of BiSAC-DP is $O(nd)$. From the function partition, we can see that the number of calling AC is nd when the Judge-DP (from lines 1 to 10) is executed only once. So, the time complexity of BiSAC-DP is $2 \times \lambda \times nd \times O(ed^2) = O(\lambda end^3)$. □

Corollary 11. *The worst-case time complexity of BiSAC-DP is $O(en^3d^5)$, and the best-case time complexity is $O(en^2d^3)$.*

Proof. The worst-case of running BiSAC-DP is that there is exactly one value removed from its domain when the loop (see line 3 in BiSAC-DP) terminates. So it calls the function Judge-DP $2 \times n^2d$ times. The number of partitions produced by calling the function Judge-DP once is at most $2 + 2^2 + \cdots + 2^k = 2^{k+1} - 2$, where $2^k = d$. Consequently, $\lambda = 2 \times n^2d \times (2^{k+1} - 2)$. Thus the worst-case time complexity is $O(en^3d^5)$. However the best case is $\lambda = n$. The best-case time complexity $O(en^2d^3)$ is obtained. □

From the above proofs, the time complexity of the algorithm BiSAC-DP is $O(en^2d^3)$ when $\lambda = n$. In that case, the tightness of problem must be small enough. For another algorithm BiSAC-DF, it has the same time complexity if it can find a solution in an already BiSAC constraint network. Note that a problem whose tightness is small may not be a bidirectional singleton arc consistent. In other words a bidirectional singleton arc consistent problem may have a big tightness.

4. Experiments

To compare the different algorithms mentioned in this paper, we have performed experimentation with respect to random CSPs, classical problems, and benchmarks. Performances have been measured in terms of the CPU time in seconds and the number of checks performed by BiSAC. All algorithms have been implemented in C++ and embedded into the constraint solving platform "Ming Yue" [14–16] designed by

Table 1: Results obtained on classical problems.

Problems	BiSAC-1		BiSAC-DF		BiSAC-DP	
	Time (ms)	#chks (M)	Times (ms)	#chks (M)	Times (ms)	#chks (M)
15-queens	3813	229	1719	104	485	29
20-queens	26140	1763	11297	771	2672	174
25-queens	129031	8543	55781	3636	10750	687
15-pigeons	2672	182	1187	82	218	14
20-pigeons	20250	1491	9547	716	1109	82
25-pigeons	99828	7482	43860	3304	4141	299
$\langle 60, 6, 177\rangle$	6234	329	2484	155	1718	102
$\langle 80, 6, 316\rangle$	17093	1043	8938	544	5500	328
$\langle 100, 5, 495\rangle$	26156	1514	15171	914	9717	576

Table 2: Results obtained on benchmarks.

Problems	BiSAC-1		BiSAC-DF		BiSAC-DP	
	Time (ms)	#chks (M)	Times (ms)	#chks (M)	Times (ms)	#chks (M)
frb30-15-1	40688	2106	25141	1294	15766	796
frb30-15-2	42235	2195	25297	1312	15782	803
Lsq_dg_7	19094	1298	5868	392	4016	273
Lsq_dg_8	64235	4712	37218	2769	12563	912
qk_8_0_5	1828	69	16	0.107	0	0.1
qk_10_0_5	9844	389	16	0.296	16	0.277
qk_12_0_5	38187	1621	47	0.664	31	0.624
composed-25_1_25_0	406	23	1516	89	375	21
composed-25_1_80_9	484	26	1766	104	422	24

us. In this platform, we use dom/deg variable ordering and choose the AC with heuristic information [17] because it is easy to implement and very efficient. The experiments were carried out on a DELL Intel PIV 3.0 GHz CPU/512 MB RAM with Windows XP Professional SP2/Visual C++6.0.

Random constraint satisfaction problems have become a standard for testing constraint satisfaction algorithms and heuristic algorithms because of their randomness [18]. A binary random constraint satisfaction problem (http://www.lirmm.fr/~bessiere/generator.html) is specified by four parameters $\langle n, m, p_1, p_2\rangle$, where n defines a number of variables, m defines the size of domains, p_1 (*density*) defines how many constraints appear in the problem, and p_2 (*tightness*) defines how many pairs of values are inconsistent in a constraint. This paper chooses $m = n = 20$, while p_1 and p_2 range from 0.05 to 0.95 (interval is 0.05). Figure 1 shows the difference of running times for BiSAC-1 and BiSAC-DF and a ratio BiSAC-1/BiSAC-DF. From Figure 1(a) we can see that the average running time required by BiSAC-DF is much less than that of BiSAC-1. Actually, from the right graph of Figure 1, we can know that BiSAC-DF is around 2–5 times faster the BiSAC-1 for small *tightness*. When the *tightness* is relatively big, BiSAC-DF is about 10–25 times than BiSAC-1. For some special combination of the *density* and *tightness*, BiSAC-DF can reach about 30 times the BiSAC-1. All of them can be explained by the fact that the times of revoking AC procedure of BiSAC-DF are much less than BiSAC-1, and BiSAC-DF can determine whether (X, a) is BiSAC earlier than BiSAC-1. Figure 2 tells us that the number of checks performed by BiSAC-DF is also much smaller than BiSAC-1. For the other algorithm BiSAC-DP, the running time is also less than BiSAC-1. In fact, it is less than BiSAC-DF too (see Figure 3). For small *tightness* BiSAC-DP is about 2–20 times the BiSAC-1 because the BiSAC-DP can find the appropriate subproblem earlier and more easily so that lots of values can be inferred BiSAC without more BiSAC checks. For the other *tightness*, BiSAC-DP is on average 40–60 times the BiSAC. BiSAC-DP is around 100 times faster than BiSAC-1 when *density* ranges from 0.5 to 0.6, and *tightness* is set to about 0.7, which is not surprising since BiSAC-DP not only uses the operator ρ to find a good subproblem successfully, but also avoids much redundant work at testing BiSAC checks. Figure 4 also shows that BiSAC-DP performs a visibly smaller number of checks than BiSAC-1.

Besides random CSPs, we have made our algorithms apply to classical problems like Queens, Pigeons, and Color problems. From Table 1, we can see BiSAC-DF and BiSAC-DP also outperform BiSAC-1 on range of CSP instances. On average, the efficiency of BiSAC-DF is about 3 times that of BiSAC-1, and BiSAC-DP is around 8–20 times faster than BiSAC-1.

Table 2, built from other instances (http://cpai.ucc.ie/05/Benchmarks.html), shows that our algorithms also have a better performance than BiSAC-1. Especially, BiSAC-DP and BiSAC-DP are about 1000 times more efficient than BiSAC-1 for the instances—"qk." For frb and Lsq_dg, our

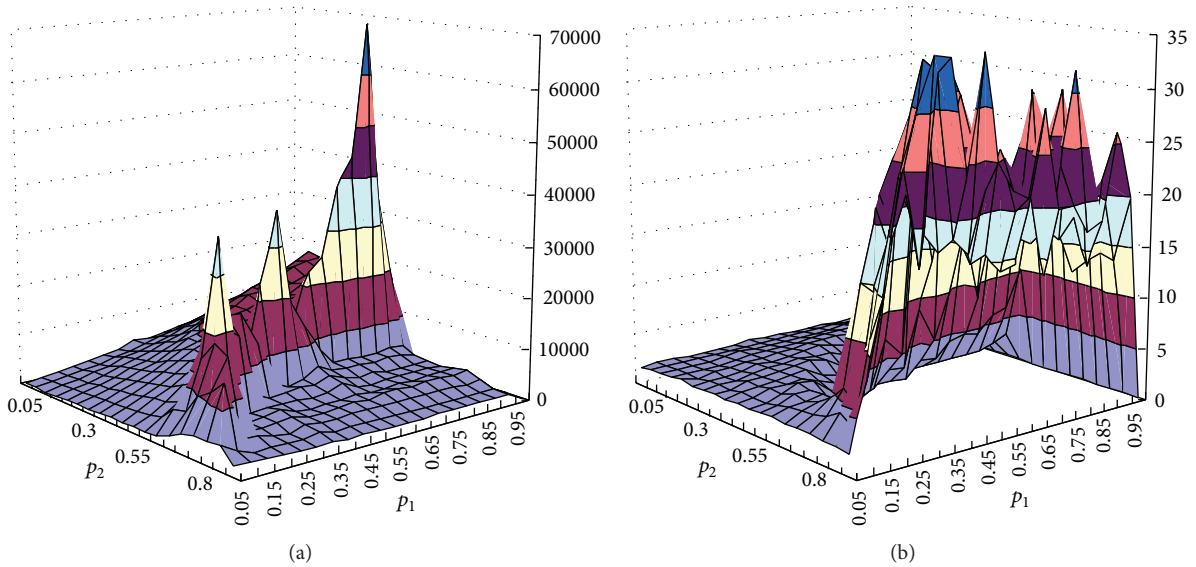

Figure 1: A comparison of running times for BiSAC-1 and BiSAC-DF. The left shows time difference (BiSAC-1 – BiSAC-DF), while the right shows time ratio (BiSAC-1/BiSAC-DF). Time is measured in milliseconds.

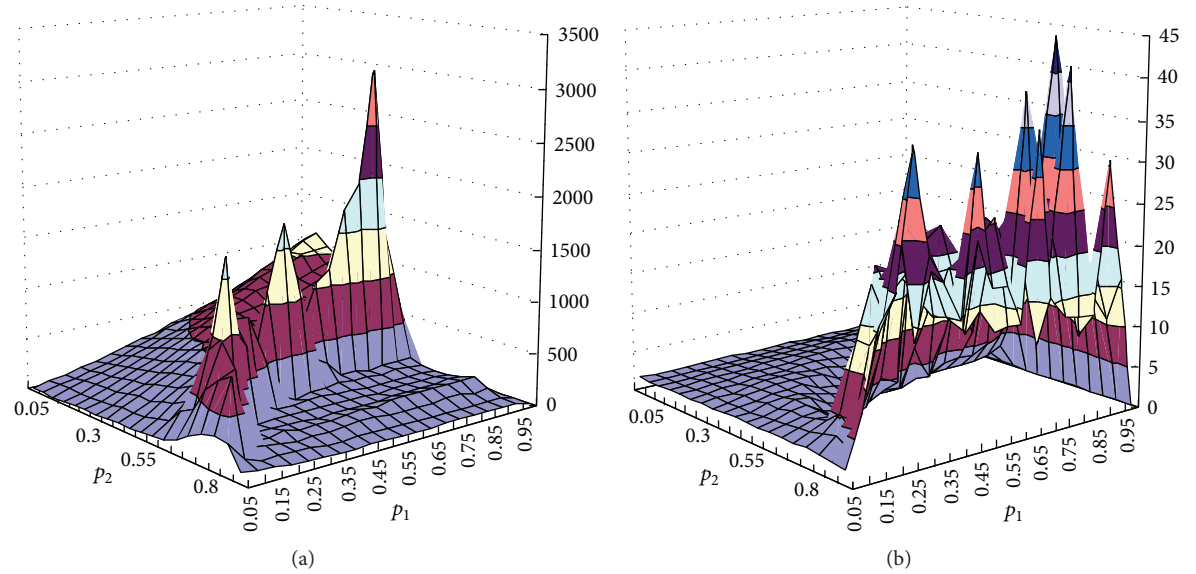

Figure 2: A comparison of number of checks performed by BiSAC-1 and BiSAC-DF. The left shows a relative comparison (BiSAC-1 – BiSAC-DF), while the right shows number of checks ratio (BiSAC-1/BiSAC-DF). Check is measured in million.

algorithms are 2-3 times more efficient than BiSAC-1. However, BiSAC-DF is slower than BiSAC-1. Maybe this is because that the procedure of depth-first instantiating variables delays the time of finding the domain which has been wiped out.

5. Conclusion

In this paper, we made a study of bidirectional singleton arc consistency and then proposed two properties. Based on these properties, two algorithms (BiSAC-DF and BiSAC-DP) were proposed. Besides, we theoretically prove that these algorithms have the same space complexity $O(nd)$ and worst-case time complexity $O(en^3d^5)$. For different cases, they also have the same best-case time complexity $O(en^2d^3)$. Because our algorithms do not invoke the AC procedure to test the BiSAC condition more times than the original BiSAC-1, so they avoid lots of redundant works and have a better performance. Our experimental results also show that our approaches significantly outperform BiSAC-1 on range of CSP instances, and it will be applied to solve large scale real life problems coming from the scheduling and configuration areas.

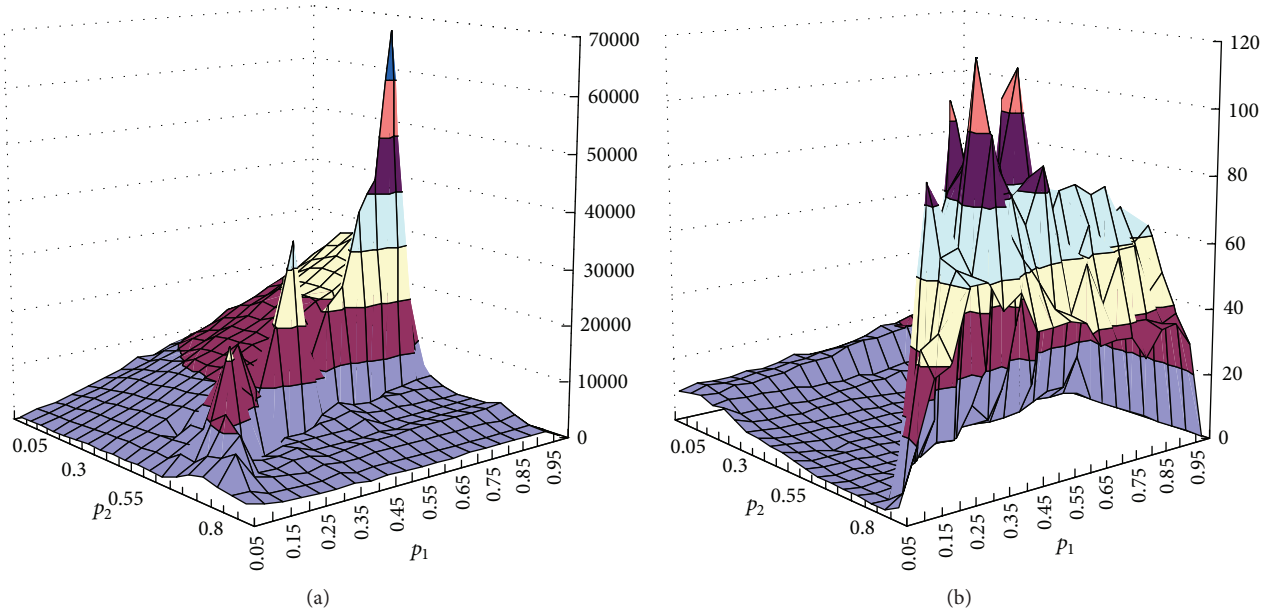

Figure 3: A comparison of running times for BiSAC-1 and BiSAC-DP. The left shows time difference (BiSAC-1 – BiSAC-DP), while the right shows time ratio (BiSAC-1/BiSAC-DP). Time is measured in milliseconds.

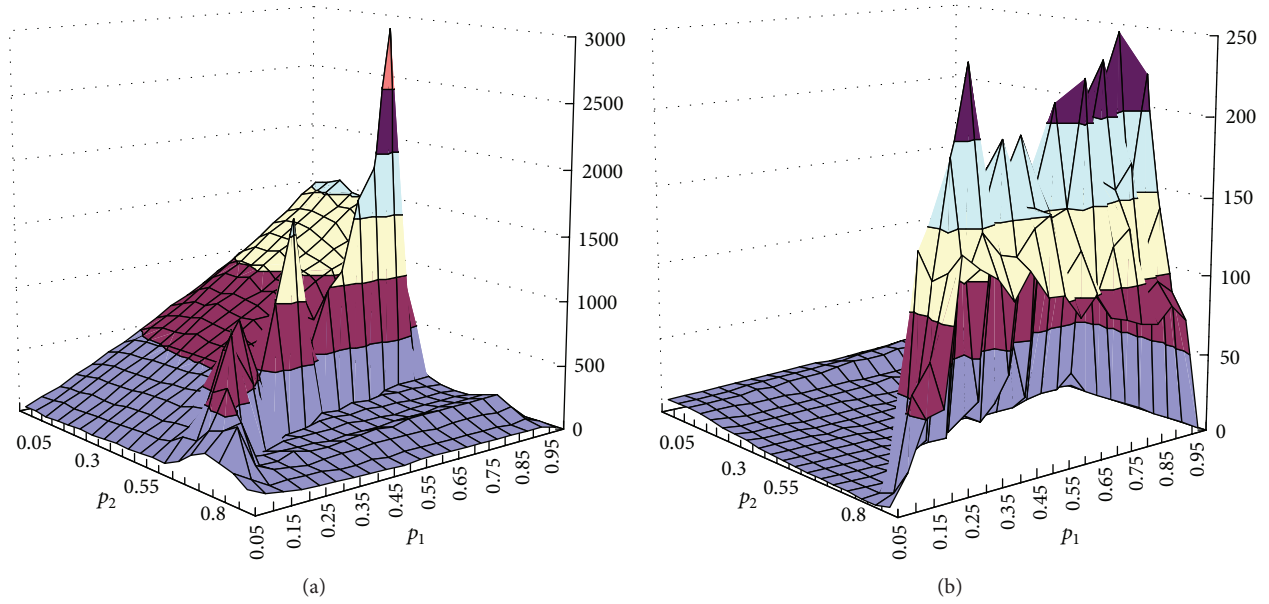

Figure 4: A comparison of number of checks performed by BiSAC-1 and BiSAC-DP. The left shows a relative comparison (BiSAC-1 – BiSAC-DP), while the right shows number of checks ratio (BiSAC-1/BiSAC-DP). Check is measured in million.

Acknowledgments

The authors of this paper express sincere gratitude to all the anonymous reviewers for their hard work. This work was supported in part by NSFC under Grant nos. 61170314, 61272208, and 61373052.

References

[1] R. Barták, "Theory and practice of constraint propagation," in *Proceedings of the 3rd Workshop on Constraint Programming in Decision and Control*, J. Figwer, Ed., pp. 7–14, Gliwice, Poland, 2001.

[2] C. Bessière, J.-C. Régin, R. H. C. Yap, and Y. Zhang, "An optimal coarse-grained arc consistency algorithm," *Artificial Intelligence*, vol. 165, no. 2, pp. 165–185, 2005.

[3] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[4] R. Mohr and T. C. Henderson, "Arc and path consistency revisited," *Artificial Intelligence*, vol. 28, no. 2, pp. 225–233, 1986.

[5] C. Bessière and R. Régin, "Refining the basic constraint propagation algorithm," in *Proceedings of the 17th International Joint*

Conference on Artificial Intelligence (IJCAI '01), pp. 309–315, 2001.

[6] R. Debruyne and C. Bessière, "Some practical filtering techniques for the constraint satisfaction problem," in *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI '97)*, pp. 412–417, 1997.

[7] A. Chmeiss and P. Jégou, "Efficient path-consistency propagation," *International Journal on Artificial Intelligence Tools*, vol. 7, no. 2, pp. 121–142, 1998.

[8] P. Prosser, K. Stergiou, and T. Walsh, "Singleton consistencies," in *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP '00)*, pp. 353–368, 2000.

[9] R. Barták, "A new algorithm for singleton arc consistency," in *Proceedings of the Forging Links and Integrating Resources Conference (FLAIRS '04)*, pp. 257–262, 2004.

[10] C. Bessière and R. Debruyne, "Theoretical analysis of singleton arc consistency," in *Proceedings of the ECAI Workshop on Modelling and Solving Problems with Constraints*, pp. 20–29, 2004.

[11] C. Bessière and R. Debruyne, "Optimal and suboptimal singleton arc consistency algorithms," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, pp. 54–59, 2005.

[12] C. Lecoutre and S. Cardon, "A greedy approach to establish singleton arc consistency," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, pp. 199–204, 2005.

[13] C. Bessière and R. Debruyne, "Theoretical analysis of singleton arc consistency and its extensions," *Artificial Intelligence*, vol. 172, no. 1, pp. 29–41, 2008.

[14] J. G. Sun and S. Y. Jing, "Solving non-binary constraint satisfaction problem," *Chinese Journal of Computers*, vol. 26, no. 12, pp. 1746–1752, 2003.

[15] J.-G. Sun, X.-J. Zhu, Y.-G. Zhang, and Y. Li, "An approach of solving constraint satisfaction problem based on preprocessing," *Chinese Journal of Computers*, vol. 31, no. 6, pp. 919–926, 2008.

[16] X.-J. Zhu, J.-G. Sun, Y.-G. Zhang, and Y. Li, "A new preprocessing technique based on entirety singleton consistency," *Acta Automatica Sinica*, vol. 35, no. 1, pp. 71–76, 2009.

[17] J. G. Sun, X. J. Zhu, Y. G. Zhang, and J. Gao, "Fail first principle for constraint propagation," *Chinese Journal of Mini-Micro Systems*, vol. 29, no. 4, pp. 678–681, 2008.

[18] I. P. Gent, E. Macintyre, P. Prosser, B. M. Smith, and T. Walsh, "Random constraint satisfaction: flaws and structure," *Constraints*, vol. 6, no. 4, pp. 345–372, 2001.