A New Adaptive Parameterized Consistency Algorithm

Yunpeng Wu
College of Computer Science and Technology
Jilin University
Changchun, China
E-mail: 2412466779@qq.com

Sibo Zhang
College of Computer Science and Technology
Jilin University
Changchun, China
E-mail: 187097385@qq.com

Yonggang Zhang *
College of Computer Science and Technology
Jilin University
Changchun, China
E-mail: zhangyg@jlu.edu.cn
* Corresponding Author

Abstract—Nowadays all kinds of constraint solver support the same level of local consistency on all problems. The authors proposed a new adaptive parameterized consistency algorithm which adjusted the level of consistency depending on the depth of their supports in there domain. To this end, the authors gave the notion of the improved distance to the end of a value, which is the dynamic distance, and it reflected the current state of the assignment. If the size of the variable domain reached its given threshold or the ratio to the original size reached its given threshold, the constraint solving process used the stronger consistency technique to propagate the assigned value. The faster deletion of invalid value, the easier cause backtracking. It is more likely to find the problem as soon as possible. Finally the authors carry on the experiments on some benchmark instances, the results showed that the new algorithm can outperform original parameterized consistency algorithm on many problems and is a promising method.

Keywords—Constraint Satisfaction Problem; Constraint Propagation; Adaptive; Backtracking; Parameterized Consistency

I. INTRODUCTION

Constraint programming (CP) is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics[1]. Constraint satisfaction problems (CSPs) are the central problems in CP, which are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which were solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time.

Backtracking and constraint propagation are two main and basic constraint solving methods[1]. Arc consistency is the oldest and most famous way of constraint propagation. It can be used to prune infeasible values in the preprocessing and backtracking phase. Arc consistency is the standard level of consistency supported in constraint solvers. In addition, several other local consistencies stronger than arc consistency have been proposed, such as max restricted path consistency[3] or singleton arc consistency[4] and their variants[6,7,8]. Unfortunately, these new consistency techniques do not play the important role in current constraint solvers because of the high computational cost of maintaining them during search.

Stergiou[9] found that choosing the right level of local consistency for solving a problem required making the trade-off between the ability of removing inconsistent values and the cost of the algorithm. And he also suggested take advantage of the power of strong consistencies to reduce the search space while avoiding the high maintaining cost in the whole network. His paper presented a heuristic approach based on the monitoring of propagation events to dynamically adapt the level of local consistency (arc consistency or max restricted path consistency) to individual constraints. This pruned more values than arc consistency and less than max restricted path consistency, but it was more efficient than the arc consistency and max restricted path consistency.

Recently, Balafrej[10] proposed an original
A value $v_i \in D(x_i)$ is a max restricted path consistent (maxRPC) support for $v_i \in D(x_i)$ on $c_i$ if and only if it is an AC support and the tuple $(v_i, v_j)$ is path consistent. A value $v_i \in D(x_i)$ is max restricted path consistent of a constraint $c_i$ if and only if there is an $v_j \in D(x_j)$ maxRPC support for $v_i$ on $c_i$. A value $v_i \in D(x_i)$ is max restricted path consistent if and only if for all $x \in C(x_i)$ it has a maxRPC support $v_j \in D(x_j)$ on $c_i$. A domain $D(x_i)$ is maxRPC if it is non-empty and all values in $D(x_i)$ are maxRPC. A constraint network $N$ is maxRPC if all domains in $D$ are maxRPC.

The problem of finding the solution of a constraint network is called constraint solving, and it is NP-complete. The basic solving method is backtrack search combined with the some level of consistency.

III. PARAMETERIZED CONSISTENCY

Parameterized consistency was based on the concept of stability of values. Balafrej introduced the notion of the distance to end of a value in a domain. This captured how far a value was from the last in its domain. In the following, the definition and the theorem came from [10].

Definition 1. (Distance to end of a value). The distance to end of a value $v_i \in D(x_i)$ is the ratio

$$\text{Dis}(x_i, v_i) = \left( \frac{\text{rank}(v_i, D_i(x_i))}{|D_i(x_i)|} \right)$$

where $D_i(x_i)$ is the internal domain of $x_i$.

Definition 2. (p-stability for AC) A value $v_i \in D(x_i)$ is p-stable for AC on $c_i$ if it has an AC support $v_j \in D(x_j)$ on $c_i$ such that $\text{Dis}(x_i, v_i) \leq p$. A value $v_i \in D(x_i)$ is p-stable for AC if for all $x_i \in C(x_i)$, $v_i$ is p-stable for AC on $c_i$.

Theorem 1. Let LC be a local consistency stronger than AC for which the LC consistency of a value on a constraint is defined. Let $p_1$ and $p_2$ be two parameters in $[0, 1]$. If $p_1 < p_2$ then AC $< \text{constraint-based p}_1$-LC $< \text{constraint-based p}_2$-LC $<$ LC.

Theorem 2. Let LC be a local consistency stronger than AC. Let $p_1$ and $p_2$ be two parameters in $[0, 1]$. If $p_1 < p_2$ then AC $< \text{value-based p}_1$-LC $< \text{value-based p}_2$-LC $<$ LC.

Definition 3. (p-maxRPC). A value and a network are p-maxRPC if and only if they are constraint-based p-maxRPC.

IV. P-MAXRPC \textit{\textsuperscript{\textit{cap}} ALGORITHM}

Definition 4. (Improved Distance to end of a value). The improved distance to end of a value $v_i \in D(x_i)$ is the ratio

$$\text{Dis}(x_i, v_i) = \left( \frac{\text{rank}(v_i, D_i(x_i))}{|D_i(x_i)|} \right)$$

Where, $D_i(x_i)$ is the current domain of $x_i$.

During the process of solving the variable universe, the authors have the priority to implement the principle of priority in the process of the current domain size to reach a certain limit. According to the experimental data summarized the threshold set to a and b. That $D_a(x_i) \leq a$ or $|D_b(x_i) / D_i(x_i)| \leq b$, the $b$ is measure the current on the domain size of the standard, the current on the domain size and initial on the domain size, the ratio is less than beta determine when the domain of the variable value up to standard, does not have to perform comparison $(x_i, v_i)$ with $P$, direct use of strong consistency checking algorithm, thus ensuring more backtracking to the premise with
minimal overhead were judged. Therefore, it is easier to produce a dead node so as to lead to backtracking and improve the efficiency of solving. The followed were the pseudocode of p-maxRPC3 algorithms.

\begin{verbatim}
Algorithm 1 Initialization(X, D, C, Q)
1. Begin;
2.  foreach v ∈ X do
3.      foreach v ∈ D(v) do
4.          foreach v ∈ Γ(v) do
5.                  p-support ← false
6.                foreach v ∈ D(v) do
7.            if (v, v) ∈ γ then
8.                LastCpu,v ← v
9.            USEmaxRPC ← false
10.            if |D(v)| ≤ 3 OR |D(v)|/|D0(v)| ≤ 0.3 OR γ(v, v) ≤ p then
11.                USEmaxRPC ← true
12.            if USEmaxRPC then
13.                p-support ← true
14.                LastCpu,v ← v
15.            break;
16.            if searchPCwitLoss(v, v) then
17.                p-support ← true
18.                LastCpu,v ← v
19.            break;
20.            if ¬p-support then
21.                remove v from D(v)
22.                Q ← Q − {v}
23.            break;
24.        if D(v) = ∅ then return false;
25.    return true;

Algorithm 2 checkPCsupLoss(vj, xj)
1. Begin;
2.  if LastCpu,v,j ∈ D(vj) then b ← ¬ max(3|LastCpu,v,j = 1, LastCpu,v,j = 0|
3.       else
4.         b ← ¬ max(LastCpu,v,j = 1, LastCpu,v,j = 0)
5.       foreach v ∈ D(vj) do
6.           if (v, v) ∈ γ then
7.               if LastCpu,v,j ∈ D(vj) & LastCpu,v,j = LastCpu,v,j then
8.                 USEmaxRPC ← true
9.               if |D(v)| ≤ 3 OR |D(v)|/|D0(v)| ≤ 0.3 OR γ(v, v) ≤ p then
10.                  USEmaxRPC ← true
11.                 if USEmaxRPC then
12.                     return true;
13.                 if searchPCwitLoss(v, v) then
14.                     return false;
15.             return true;

Algorithm 3 checkPCwitLoss(vj, xj)
1. Begin;
2.  foreach v ∈ Γ(vj) do
3.        witness ← false
4.    if sv = LastCpu,v,j ∈ D(vj) then
5.        USEmaxRPC ← false
6.    if |D(v)| ≤ 3 OR |D(v)|/|D0(v)| ≤ 0.3 OR γ(v, v) ≤ p then
7.        USEmaxRPC ← false
8.    if USEmaxRPC then
9.        witness ← true
10. else
11.     if LastCpu,v,j ∈ D(vj) & LastCpu,v,j = LastCpu,v,j then
12.        OR LastCpu,v,j ∈ D(vj) & LastCpu,v,j = LastCpu,v,j else
13.        OR LastCpu,v,j ∈ D(vj) & LastCpu,v,j = LastCpu,v,j
14.        then witness ← true
15.        else
16.        if searchPCwitLoss(v, v) then
17.        return true;
18.        return false;
19.    return true;

Figure 1. Algorithm 1 Initialization

Figure 2. Algorithm 2 checkPCsupLoss and Algorithm 3 checkPCwitLoss

The line 9 of the algorithm 1 initialization (X, D, C, Q) (Fig. 1), line 9 of the algorithm 2 checkPCsupLoss (v, x) and the line t of the algorithm 3 checkPCwitLoss (x, v, x), the authors add the |D (x)| ≤ 3 and |D (x)|/|D0 (x)| ≤ 0.3 judgment, namely, to determine the current domain size, if on a smaller domain, use the
delete value ability strongly consistent algorithm, so more easily produced dead node as soon as possible so as to lead to backtracking, improve the efficiency of the algorithm.

V. EXPERIMENTS

In this section, the authors will show the experimental evaluation of the proposed algorithm. All the problem instances come from the website (http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html) maintained by Lecoutre. All algorithms have been implemented in C++ and embedded into the constraint solving platform designed by ourselves. In this platform, the authors use dom/deg variable ordering because it is easy to implement and very efficient. The experiments were carried out on a DELL Intel i7-3770 3.40GHz CPU/8GB RAM with Windows 7.0/Visual C++6.0.

<table>
<thead>
<tr>
<th>Instances (sat)</th>
<th>Measure</th>
<th>maxRPC</th>
<th>p-maxRPC</th>
<th>apc-maxRPC</th>
<th>apx-maxRPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>qwh-15-106-0_ext</td>
<td>cpu(s)</td>
<td>307.551</td>
<td>Time-out</td>
<td>479.477</td>
<td>302.539</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>4348296</td>
<td></td>
<td>2857255</td>
<td>3945798</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>1856</td>
<td></td>
<td>3893</td>
<td>1856</td>
</tr>
<tr>
<td>qwh-15-106-2_ext</td>
<td>cpu(s)</td>
<td>302.173</td>
<td></td>
<td>310.1</td>
<td>275.95</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>4299125</td>
<td></td>
<td>4296012</td>
<td>1879</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>1879</td>
<td></td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td>qwh-15-106-3_ext</td>
<td>cpu(s)</td>
<td>3585.64</td>
<td></td>
<td>1179.71</td>
<td>524.15</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>28010711</td>
<td></td>
<td>5660938</td>
<td>7091034</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>24135</td>
<td></td>
<td>9287</td>
<td>2801</td>
</tr>
<tr>
<td>qwh-15-106-4_ext</td>
<td>cpu(s)</td>
<td>56.097</td>
<td></td>
<td>34.947</td>
<td>23.508</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>518713</td>
<td></td>
<td>124863</td>
<td>566.563</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>2990</td>
<td></td>
<td>2990</td>
<td>2801</td>
</tr>
<tr>
<td>qwh-15-106-6_ext</td>
<td>cpu(s)</td>
<td>73.039</td>
<td></td>
<td>27.368</td>
<td>22.564</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>409856</td>
<td></td>
<td>409847</td>
<td>446423</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>230</td>
<td></td>
<td>230</td>
<td>230</td>
</tr>
<tr>
<td>qwh-15-106-7_ext</td>
<td>cpu(s)</td>
<td>1.859</td>
<td></td>
<td>1.675</td>
<td>1.652</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>60896</td>
<td></td>
<td>1.822</td>
<td>97603</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>114</td>
<td></td>
<td>138</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qwh-15-106-8_ext</td>
<td>cpu(s)</td>
<td>69.254</td>
<td></td>
<td>179.123</td>
<td>62.659</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>2538845</td>
<td></td>
<td>5171642</td>
<td>952177</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>1517</td>
<td></td>
<td>690</td>
<td>422</td>
</tr>
<tr>
<td>qwh-15-106-9_ext</td>
<td>cpu(s)</td>
<td>1.571</td>
<td></td>
<td>1.573</td>
<td>1.685</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>71664</td>
<td></td>
<td>1.573</td>
<td>107256</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>104</td>
<td></td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>qwh-15-106-10_ext</td>
<td>cpu(s)</td>
<td>61.54</td>
<td></td>
<td>66.831</td>
<td>66.828</td>
</tr>
<tr>
<td></td>
<td>#ccks</td>
<td>1039289</td>
<td></td>
<td>1092199</td>
<td>6672117</td>
</tr>
<tr>
<td></td>
<td>#nodes</td>
<td>421</td>
<td></td>
<td>421</td>
<td>2900</td>
</tr>
</tbody>
</table>

TABLE I. PERFORMANCE (CPU TIME, NODES AND CONSTRAINT CHECKS) OF MAXRPC, VARIABLE-BASED AP-MAXRPC(APX-MAXRPC), CONSTRAINT-BASED AP-MAXRPC(APC-MAXRPC) AND THE CORRESPONDING IMPROVED VERSION ON QWH INSTANCES
All the instances in TABLE I are belong to the SAT (satisfaction type), where each instance including 100 variables and 900 constraints. The experiments showed that the improved algorithms have advantage both in solving time and visited nodes. Compared with the classical maxRPC algorithm, the adaptive version of maxRPC algorithms were not as good as the classical maxRPC on the most instances. It showed that the adaptive version algorithm increase the computational cost though they decreased the strong consistency cost. The authors could observe the data about p-maxRPC, apc-maxRPC, apx-maxRPC algorithm, and could find that apx-maxRPC was the best one. It was because that this algorithm adjusted threshold p using variable-based method, the value of p became bigger and bigger with the domain reduced, thus it enforced strong consistency more frequently and got the solution more quickly. The improved p-maxRPC, apc-maxRPC, apx-maxRPC had stable advantage both on the CPU time and number of the visited node. The execution time is generally less than the original algorithms, and the number of nodes is the same as those of the original algorithms.

In the original three parameters of the algorithm can be seen based on the adaptive variable is a absolute advantage, the reasons above have given analysis. By comparison, the new three parameter apx-maxRPC algorithm is more efficient than the algorithm based on variable in the same optimization algorithm. Although the overall execution time is less than before optimization, most of the algorithms and the visit to the number of nodes is minimized, but apx-maxRPC algorithm on based on the method of adaptive variable optimization is not perfect. This is mainly because the idea is to optimize the value of the idea is to update the value of the end of the distance, while the variable is also the variable adaptive and it is worth to cut and continue to change. Based on the variation of the threshold parameters of the variable is by all variables where the median constraint deletion caused, but terminal distance changes indeed within the limited scope changes, this range is the initial domain size. Therefore, change from the end of distance speed to change the parameters of the threshold p value range. The implementation of the algorithm is less than the value of the p value of the implementation of the strong compatibility check. In front of the initial domain value because of distance is greater than the threshold; it is not strong compatibility check. However, due to the rapid increase of P value, it is not required to perform the strong compatibility check of the value of the end of the scale is less than P, which makes a partial value of the early implementation of the strong compatibility check. As a result, the strong consistency checks led to the increase in the implementation of the time.

VI. CONCLUSIONS

State-of-the-art constraint solver maintains uniformly maintain the same level of local consistency on all the instances. The authors proposed a new parameterized local consistency based on the notion of the improved distance to the end of a value, which is the dynamic distance, and it reflected the current state of the assignment. If the size of the variable domain reached its given threshold or the ratio to the original size reached its given threshold, the constraint solving process used the stronger consistency technique to propagate the assigned value. The faster deletion of invalid value, the easier cause backtracking. It is more likely to find the problems as soon as possible. Finally the authors carry on the experiments on some benchmark instances, the results showed that the new algorithm can outperform original parameterized consistency algorithm on many problems and is a promising method.

ACKNOWLEDGMENT

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 No. 61170314, No. 61373052.

REFERENCES