Title: Saving constraint checks in maintaining coarse-grained generalized arc consistency
Authors: Hongbo Li, Ruizhi Li and Minghao Yin
Publication Venue: ©The Natural Computing Applications Forum 2017
Received: 8 December 2016
Accepted: 11 April 2017
Published online: 3 May 2017

I. Introduction

Constraint satisfaction problem(CSP) is a well-studied problem in artificial intelligence. Several algorithms are proposed to maintain arc consistency in a constraint network. A new algorithm is proposed by the paper which avoids unnecessary constraint checks in maintaining arc consistency in a constraint network.

II. Related Works

Maintaining arc consistency(MAC) is very popular to solve binary csp. Nevertheless to solve a non-binary csp generalized arc consistency(GAC) algorithms are used. GAC algorithms are efficient and uses few data structures. Two types of GAC algorithms are well known, (1) Coarse-Grained: Based on constraint and value propagation schemes and (2) Fine-Grained: uses heavy data structures. The most popular coarse-grained AC algorithm is AC3. AC3.1 and AC3.2 avoids repeated constraint checks by recording last support and multi-directional support respectively but they use heavy data structure. MAC3r uses residue supports and performs better than MAC3.1 and MAC3.2. MAC3rm outperforms MAC3r because it uses few data structures during search. MAC3rm2 considers multiple residues and shows better performance than MAC3rm. These coarse-grained AC algorithms can be used in generalized arc consistency(gac) version.

III. Problem Definition

A constraint satisfaction problem is defined as $P = (X, D, C)$ where X is a set of variables x_1, x_2, x_3, x_4, ..., x_n . D is a domain for each of the variables D_1, D_2, D_3, D_4, ..., D_n and C is a set of constraint where each c constraint is defined using $scp(c)$, the variables involved in that constraint in a particular order and a subset of $D_1 \times D_2 \times D_3 \times D_4 \times ... \times D_n$. A constraint involving two variables is called a binary constraint. A variable x_i is called arc consistent with a variable x_j if and only if for every value $a_i \in D_i$ there is at least one value $a_j \in D_j$ which satisfies the binary constraint between x_i and x_j. A network is called arc-consistent if all of its arcs are arc-consistent. When more than two variables are involved in a constraint then it is called a non-binary constraint.

• A tuple of a non-binary constraint, involving a set of variables denoted as scp(c) is an element of $D_1 \times D_2 \times D_3 \times D_4 \times ... \times D_n$. A tuple is denoted as τ and $\tau[x]$ is called the value of variable x in tuple τ .
• If a constraint, c contains r variables then r is called the arity of the constraint. If the domain size of a variable is d then there will be d^r tuples for constraint c. Suppose, t is the number of disallowed tuples by constraint c then t/d^r is the tightness of the constraint c.
• If there is a tuple, τ exists for a constraint c such that $\tau[x] = a$ where $x \in scp(c)$ and $a \in D_x$ then (x, a) is called consistent with c.
• If $\forall x \in scp(c)$, $D_x \neq \emptyset$ and $\forall a \in D_x, (x, a)$ is consistent then c is called a generalized arc consistent.
• A CSP P is generalized arc consistent if all of its constraints are generalized arc consistent.

Generalized arc consistency(GAC) algorithm is a kind of pre processing algorithms. Saving constraint checks is important in maintaining generalized arc-consistency because some constraint checks are relatively expensive than others. Constraint checks can be repeated in two ways. If a tuple that is allowed by a constraint is checked again then it is called positive constraint and if a tuple that is disallowed by a constraint is checked again is called a negative constraint. The problem is to design a generalized arc consistency algorithm which will avoid repeated constraint checks.

IV. Proposed Method

A generalized arc-consistency algorithm named growing tabular reduction **(GTR)** is proposed which avoids all positive repeats and maximum negative repeats. The data structures used by this algorithms are :
• $tupleList(c)$: A dynamic list which containts tuples that have been checked to be allowed by constraint c. The tuples of a list which are not yet checked to be invalid are called active tuples.
• $firstActive(c)$: Index of the first active tuple of the $tuplelist(c)$.
• $levelLast(c)$: An array of size (n+1) where n is the number of variables and $levelLast(c)[p]$ denotes the last invalid tuple of $tupleList(c)$ at the pth level of search. Pre-processing is done at level 0. If $levelLast(c)[p]$ = -1 then no tuples were removed at level p.

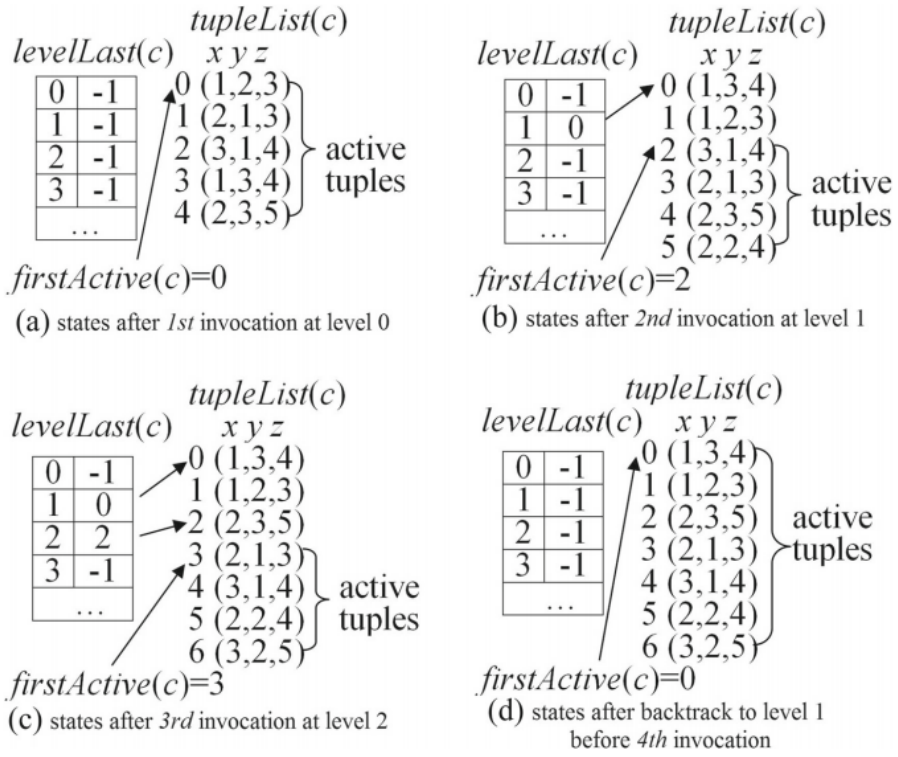

Figure 1: Data Structures of GTR algorithm

When the algorithm GTR is at level p then all the tuples of $tupleList(c)$ before $activeLast(c)$ are verified to be invalid in previous level. For each level, all the tuples with residue supports are identified in the first part of the algorithm and in the second part the residue supports for rest of the values are found. If a value from a variable's domain is removed by other constraint then the tuples of the indices from $levelLast(c).size() - 1$ to $firstActive(c)$ are checked to find the invalid tuples and the $firstActive(c)$ and $leveLast(c)$ are updated. Since the algorithm checks only the tuples up to $firstActive(c)$ so repeated tuple checking is avoided. If the search backtracks, then $firstActive(c)$ can be updated using the array $levelLast(c)$.

Suppose, we have a constraint $c : x = y + z$ involving three variables x, y, z where the domains of x,y and z are $D_x = D_y = \{1, 2, 3\}$ and $D_z = \{3, 4, 5\}$

(a) At the 0^{th} label the algorithm finds supports for all values of each variables domain and the tuples are added to the $tupleList(c)$. Since all the tuples are yet to be tested so $firstActive(c)$ = 0 [Figure 1(a)]

(b) Suppose, (x,1) is removed by other constraint at level 1 so index 4 to 0 of $tupleList(c)$ are checked and tuples (1,2,3) and (1,3,4) are removed. Therefore, $firstActive(c)$ = 2 and $levelLast(c)[1] = 0$. For (y,2) the algorithm finds a new support (2,2,4) in part 2. Hence, (2,2,4) is added to the end of the tuple list.

(c) Let, (y,3) is removed by another constraint. So the algorithm checks from index 5 to index 2 at level 2. (2,3,5) is removed from the list. Now, $firstActive(c) = 3$ and $levelLast(c) = 2$. (3,2,5) is found as a new support for (z,5) and added to the end of the $tupleList(c)$.

(d) If the constraint c is never checked again and the search backtracks to level 1, then both (x,1) and (y,3) are restored. Besides, $firstActive(c)$ is restored as 0 and $levelLast(c)[2]$ and $levelLast(c)[1]$ are set to -1.

V. Results

The constraints on which the proposed method are implemented are tight non-binary constraints. For simplicity I have implemented the method for randomly generated binary-constraints and compared with other algorithms like AC1,AC2,AC3 and AC4. GTR algorithm performs as good as other algorithms in terms of running time and also pass the Anova's test with hsd tuckey.

Treatment pairs	Tukey HSD Q statistic	Tukey HSD p-value	Tukey HSD inferfence
AC3 vs AC3GTR	2.8209	0.0468329	* p <0.05

TABLE 1: Anova's Test with hsd tuckey result

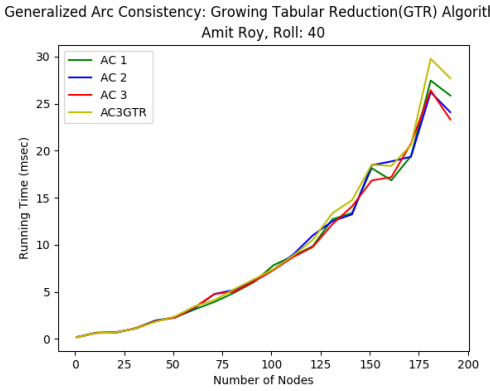

Figure 2: Running time vs number of nodes (1~200)

Treatment $\rightarrow$	AC3	AC3GTR	Pooled Total
observations N	180	180	360
sum $\sum x_i$	196.6006	204.1778	400.7784
mean $\overline{x}$	1.0922	1.1343	1.1133
sum of squares $\sum x_i^2$	221.7041	238.9809	460.6851
sample variance s^2	0.0389	0.0412	0.0404
sample std. dev. s	0.1974	0.2030	0.2010
std. dev. of mean $SE_{\overline{x}}$	0.0147	0.0151	0.0106

TABLE II Descriptive statistics of k=4 independent treatments

VI. Conclusion

The worst-case time complexity of GTR to establish GAC at the preprocessing step is $O(er^3d^{r+1})$ with space complexity $O(er^2d)$. The experiment is performed on binary-constraint but it is more suitable for non-binary tight constraints because GTR algorithm has more improvement in non-binary constraints than in binary constraints. But still this algorithm performs as good as AC algorithms for binary constraints and avoids repeated constraints.