Restricted Path Consistency Enforcement for any Constraint Network

Ahlem Ben Hassine† and Tu Bao Ho†

Local consistency techniques (LC) are in the core of the constraint programming paradigm due to their preminent role in its success. The main objective of these techniques is to prune the search space and consequently to enhance the efficiency of the constraints solver. Several levels were proposed among which arc consistency (AC) is the most used one due to its low time and space complexities. However, recently few efforts were directed to enforce local consistency in an entirely distributed manner. Nevertheless, most of these works are limited only to AC property due to the effective-cost of the other existing more powerful levels. For some hard CNs applying only AC enforcement may be fruitless, case of problems initially arc-consistent.

In an attempt to overcome these limitations, the main contribution of this paper is to propose a refinement of the DRAC approach (Distributed Reinforcement of Arc-Consistency) to achieve higher level of local consistency, the restricted path consistency (RPC) in a distributed manner with the minimal amount of additional constraint checks. A comprehensive empirical study was performed to highlight the benefit of using the collected knowledge for enforcing arc-consistency on any binary constraint network (CN), especially for hard arc-consistent problems.

1. Introduction

Constraint Satisfaction Problem (CSP) formalism is ubiquitous in representing and solving many practical combinatorial applications, such as planning, resource allocation, time tabling, frequency allocation and scheduling. The great success of the CSP paradigm is due to its simplicity in specifying many kinds of real-world applications. A CSP is defined by a set of variables, a domain of values for each variable and a set of constraints between these variables. Solving a CSP involves finding assignments of values to variables that satisfy all the constraints. This type of problems is known as NP-Complete for which the solving task is hard. Many significant researches are focused on improving the efficiency of finding solution. Reinforcing local consistency is worthwhile for pruning the search space and consequently improving the efficiency of constraints' solvers.

Hence, several levels of local consistency have been proposed in the literature. The majority of the proposed enforcing techniques are centralized and addresses mainly arc-consistency[11] and partial forms of arc consistency. This can be vindicated by two main reasons: the first is that arc consistency cheaply removes some values that cannot belong to any solution for binary CSP. The second reason is that higher levels of consistency (path consistency or k-consistency with k>3) require high space and time complexities and also can change the structure of the constraint network (CN). However, the best centralized proposed algorithms for enforcing PC proposed in the literature are PC-5 in 13) with $O(n^3d^3)$ worst-case time complexity and $O(n^3d^2)$ worst-case space complexity, and PC-8 in 7) with $O(n^2d)$[☆] worst-case time complexity and $O(n^3d^4)$ worst-case space complexity (n is the total number of variables in the problem and d is the size of the largest domain).

Therefore, many lesser levels of local consistency have been defined for binary constraint satisfaction problems[8]. According to the *stronger than* property defined in 8), some of these levels are situated between AC and PC. The main advantage is that they can prune more inconsistent values than AC while avoiding the drawbacks of PC. The restricted path consistency (RPC) property proposed by Berlandier in 2) is one of these levels. The underlying centralized proposed technique for enforcing RPC does not remove only the arc in-

† Knowledge Creating Methodology Laboratory
Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292
email: {hassine, bao}@jaist.ac.jp

☆ As mentioned in 8) this algorithm still requires $O(n^2d^2)$ data structure for the constraints representation.

consistent values but in addition it checks the path consistency of the pairs of values involving a weakly supported value. The RPC centralized algorithm has O(end^3) worst-case time complexity and O(end+ed^2) worst-case space complexity with e is the number of constraints.

In this paper, we are interested in the distributed approaches due essentially to the natural distribution of many real CSP applications. The few distributed approaches proposed in the literature concerns in a great part, the enforcement of AC. One such approach is DRAC approach in 5)∼6) that is a distributed approach to enforce arc consistency on any constraints. However, for some hard CNs performing only arc-consistency is fruitless. Therefore achieving more local consistency pruning levels, with reasonable cost, can be worthwhile. Thus, we should find the best compromise between the cost of the filtering process and the amount of deleted values. The main contribution of this paper is to refine the DRAC approach to perform restricted path consistency (RPC) with the minimum amount of additional constraint checks. The main idea of the new generic approach, that we called DRAC^{++}, is to profit from the collected information while enforcing arc-consistency to prune more non-viable values for any CNs.

This paper is organized as follows. First we present some preliminaries. Second, we describe DRAC^{++} generic approach. Third, we discuss the experimental results. Finally, we conclude the paper.

2. Preliminaries

In this section we will introduce some useful definitions and notations.

Definition 1. Informally, a CSP[12)] is composed of a finite set of n variables X={X_1, ..., X_n}, each of which is taking values in an associated finite domain D={D(X_1), ..., D(X_n)} with |D(X_i)|=d, i.e. d is the size of the largest domain and a set of e constraints between these variables C={C_{ij}, ...}. We assume a total order $\prec_d$ on the values of each domain without loss of generality.

Solving a CSP consists in finding one or all-complete assignments of values to variables satisfying all the constraints.

A CSP can be associated to a constraint-graph, the nodes of which (respectively arcs) represent variables (respectively constraints). As for high-order constraints, they can be represented according to primal constraint-graph or dual constraint-graph[9)].

Definition 2. A binary CN is *Restricted Path Consistent* (RPC)[2)] if and only if:

- For all variable $X_i \in X$, D(X_i) is a non-empty arc consistent domain and,
- For all value $a \in$ D(X_i), for all $X_j \in X$ such that a has a *unique* support $b \in$ D(X_j),
- For all $X_k \in X$ linked to both X_i and X_j, there exists $c \in$ D(X_k) such that (a, c) satisfies C_{ik} AND (b, c) satisfies C_{jk} ($C_{ik}(a, c) \wedge C_{jk}(b, c)$)

Following Montanari in 12), a binary relation C_{ij} between variables X_i and X_j can be represented by a (0, 1)-matrix with |D(X_i)| rows and |D(X_j)| columns by imposing an order on the domains of the variables. A *zero* entry at row a column b means that the pair consisting of the a^{th} element of D(X_i) and the b^{th} element of D(X_j) is not permitted; a *one* entry means that the pair is permitted. However for the case of constraints in intension, to determine all the allowed couples of values requires high time and space cost.

In the following, we propose a new property based on *restricted path inconsistency* that we will use in the proposed protocol in order to prune more inconsistent values from the CN. The main objective is to improve the efficiency of DRAC approach without loss of correctness.

Property 1. For each path of three variables P_3={X_i, X_j, X_k} of an arc-consistent CN. For each $X_i \in P_3$, for each value $a \in$ D(X_i) and its arc-consistent support* $b \in$ D(X_j), a is an inconsistent value and consequently should be removed from D(X_i) if and only if:

- There is no common support $c \in$ D(X_k) such that $SP_{X_iX_k}[a]$=$SP_{X_jX_k}[b]^{\star\star}$=$c$.
- For all $b' \in$ D(X_j) with $b' \neq b$ and for all $c' \in$ D(X_k), $SP_{X_iX_j}[b']$=a' with a' first support of b' and $a' > a$ and $SP_{X_iX_k}[c']$=a" with $a"$ first support of c'

* This support can be the first support or one support, by using bidirectionality property in 3)∼4), depending on the used order between variables.

** $SP_{X_iX_j}$ represents the collected knowledge result of performing AC.

and $a'' > a$.

Note that by using bidirectionality between two variables X_i and X_j while enforcing AC, we can have knowledge about the *first* support of X_i in X_j and not the inverse. Therefore, in case it is not possible to check this condition, we need to perform more constraint checks to verify the inconsistency of a by applying RPC property as mentioned in the following condition,

- The value $b \in D(X_j)$ is the only support of $a \in D(X_i)$ and there is no common value support $c \in D(X_k)$ such that the pair (a, c) and the pair (b, c) are simultaneously arc-consistent.

The two first conditions of the above mentioned property are used to infer the oneness of supports for the value a to detect whether it is path inconsistent or not without performing extra constraint checks.

3. DRAC^{++} Generic Approach

3.1 Global Dynamic

The proposed model for the new approach DRAC^{++} involves (as for DRAC model) two kinds of agents, Constraint agents and the Interface agent, communicating by exchanging asynchronous point-to-point messages. For transmission of messages, we assume that they are received in the same order they were sent and in a finite delivering time.

The main goal of DRAC++ is to transform any CSP P (X, D, C) into another CSP P'(X, D', C) equivalent via interactions among the Constraint agents, which are trying to reduce their domains. The underlying new proposed protocol is divided into two steps

- First, enforce arc consistency on the problem (the same as DRAC protocol),
- Second, use the knowledge collected from the previous step to remove some additional values that cannot belong to any solution by enforcing RPC property.

At the initial state, the Interface agent creates all the Constraint agents and activates them. Each agent C_{ij} reduces the domains of its own variables by computing local first viable value for each variable.

Lets recall that for each variable X_i, for each value $a \in D(X_i)$, if its *first* support $b \in D(_j)$ is found, then $(a\ b\ y)$ is added to the list of tuple supports $SP_{X_iX_j}$, i.e. y=0 (*resp.* y=1), if $b \in D(X_j)$ (*resp.* $a \in D(X_i)$) is the *first* support of $a \in D(X_i)$ (*resp.* $b \in D(X_j)$).

We must note that b is the first value support for a but they are also values support for each other by applying the bidirectionality property of relations associated to constraints. A value a is deleted from $D(X_i)$ if and only if it has no viable value support. Each obtained set of deleted values for a variable should be announced immediately to the concerned acquaintances in order to save fruitless consistency checks for these values by the other agents. Obviously, reducing domains on an agent may cause an eventual domains' reductions on other agents. The same process, domains' reduction and exchange of deleted values, should resumes until the full global arc-consistency is achieved or a domain wipes out, i.e. the problem is then detected as inconsistent.

Hence, all the agents starts the second step in order to prune more non-viable values. Each agent C_{ij} checks first if it belongs to a path formed by three variables. This is can be done by checking its list of constraint acquaintances. The same agent may belong to more than one path. First for each path, each agent C_{ij} asks its path acquaintance agents[☆] (C_{ik} and C_{kj}) for their sets of first support ($SP_{X_iX_k}$ and $SP_{X_kX_j}$). For each received set, the agent C_{ij} determines first the boolean matrices M_{ik} and M_{kj} corresponding to $SP_{X_iX_k}$ and $SP_{X_kX_j}$ respectively. Second, performs the multiplication of these two matrices. Each entry of the obtained matrix $M_{Prod_{ij}}$ indicates the existence (entry equal to 1) or not (entry equal to 0) of a path of length 2 between the two variables X_i and X_j of the agent C_{ij} through the variable X_k. Finally the agent performs the convolution of $M_{Prod_{ij}}$ and its first support matrix M_{ij} by applying the multiplication operator as follows:

$$\forall\ m \in \{1, \ldots, D(X_i)\} \text{ and}$$
$$\forall\ l \in \{1, \ldots, D(X_j)\}$$
$$M_{Res}[m][l] = M_{Prod_{ij}}[m][l] * M_{ij}[m][l].$$

To illustrate the principle of the proposed protocol, let us consider the example in Figure 1 formed by three variables (X_1, X_2 and X_3). Figure 2 shows the proposed model correspond-

[☆] All the constraint agents belonging to the same path.

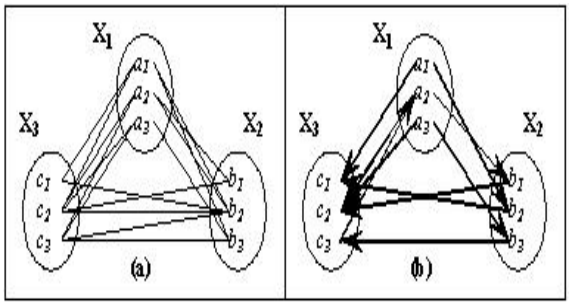

Fig. 1 (**a**) Example of arc-consistent problem, (**b**) the corresponding graph of first support values.

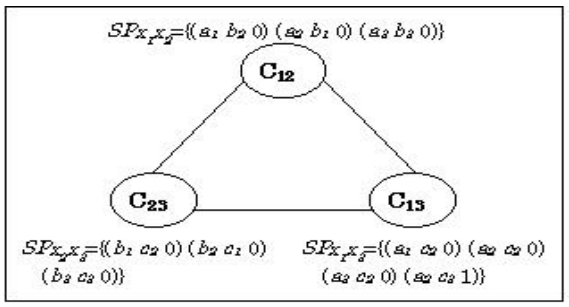

Fig. 2 The corresponding model for the proposed approach.

ing to the above example. Let us consider the agent C_{13} ($SP_{X_1X_3}$={(a_1 c_1 0) (a_2 c_2 0) (a_3 c_2 0) (a_2 c_3 1)}), it will receive the set of first support from its two acquaintances ($SP_{X_1X_2}$={(a_1 b_2 0) (a_2 b_1 0) (a_3 b_3 0)} and $SP_{X_2X_3}$={(b_1 c_2 0) (b_2 c_1 0) (b_3 c_3 0)}). It will determine the corresponding following matrices:

$$M_{12} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, M_{23} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and $M_{13} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Then, it will settle the product of the two first matrices:

$$M_{12} * M_{23} = M_{Prod_{13}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Finally the agent should determine M_{Res} using $M_{Prod_{13}}$ and M_{13} as mentioned above.

$$M_{Res} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

For each $a \in D(X_i)$ (*resp.* $b \in D(X_j)$)
If $\Sigma_{h=1}^{|D(X_j)|} M_{Res}[a][b_h]<1$ (*resp.* $\Sigma_{h=1}^{|D(X_i)|} M_{Res}[a_h][b]<1$)

Then the agent should check if the value $a \in D(X_i)$ (*resp.* $b \in D(X_j)$) is restricted path inconsistent or not and this by using the third criterion of the property 1. Each value that does not satisfy the property conditions should be deleted and consequently propagated. For our example, we have to check only a_3 and c_3.

The same process is repeated for the other paths. However, enforcing local RPC on an agent may lead to AC enforcement, which in its turn leads to more RPC enforcement. Thus the same process should continue until the stable equilibrium state is reached. This state can be defined by the satisfaction of all the agents of the system. An agent is satisfied if and only if it has no arc inconsistent or restricted path inconsistent value. It is noticeable that we can be content with enforcing *Lazy* RPC, one pass of RPC, in order to reduce the complexity of the pruning process.

Note that this dynamic allows a premature detection of failure: absence of solutions. Thus, in the case of failure, the constraint (which has detected this failure) sends a message to the interface in order to stop the whole process. The maximal reinforcement of global restricted path-consistency is obtained as a side effect from the interactions described above.

3.2 Termination

The global dynamic of DRAC^{++} approach stops when the system reaches its finite stable equilibrium state. The state where all the restricted path inconsistent values are pruned or when one of the domains wipes out. At this state, all the agents are satisfied. However, in this second case, the problem is inconsistent i.e. no instantiation satisfies all the constraints.

The detection of the stable equilibrium state in a distributed system can be achieved by taking a snapshot of the system, using the well known algorithm of 10). Termination occurs when all the agents are waiting for a message and there are no messages in the transmission channels. The cost, of the termination process, can be mitigated by combining snapshot messages with our protocol messages.

4. Experimental Comparative Evaluation

In this section, we provide experimental

tests on the performance and efficiency of the distributed filtering new generic approach DRAC^{++}. The experiments were performed over randomly generated instances using four parameters: n is the number of variables, d is the domain size of each variable, p is the graph connectivity (the proportion of constraint in the network, p=1 corresponds to the complete graph) and q is the constraint looseness (the proportion of allowed pairs of values in a constraint). The implementation was developed with Actalk[1)] under Smalltalk-80 environment.

We randomly generated a list of instances according to the following parameters, n=20; d=10; $p \in \{0.2; ...; 0.9\}$ with a step of 0.1 and $q \in \{0.35; 0.38; 0.42; 0.45; 0.47; 0.48; 0.5; 0.5\}$. We carried out our experiments on the most complex arc-consistent problems. The main goal of these experiments is to highlight the usefulness of using meta-knowledge inferred from the set of first support to prune more inconsistent values on hard CN, with the minimum amount of additional constraint checks. For each $\langle p, q \rangle$, 10 CNs instances were tested, for which we performed only *lazy* RPC (one pass RPC) in order to show that for some instances only partial RPC is enough to prove the inconsistency especially of most hard over-constrained problems.

The results reported below represent the average of the obtained outcomes in terms of three criteria: the CPU time in milliseconds, the percentage of pruned values, and the number of constraint checks (ccks).

We bought out two versions of DRAC^{++}: DRAC^{++}-1 and DRAC^{++}-2 for respectively without and with the proposed property (see Section 2). At first glance the result in Figure 3a shows that DRAC^{++}-2 required little more CPU time ($\simeq$14%) than DRAC^{++}-1. This additional CPU time is used in order to decrease the number of constraint checks. Figure 4 shows that the use of the proposed property leads to save almost the half ($\simeq$45%) of the needed number of ccks (Figure 4). The saving of ccks increases hand-in-hand with the hardness of the problem.

The difference in the percentage of deleted values noticed between the two versions of DRAC^{++} (Figure 3b) is vindicated by the fact that the used instances include restricted path

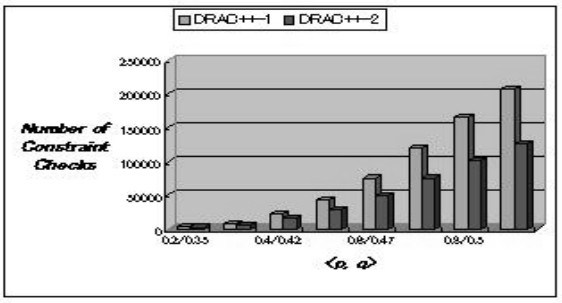

Fig. 4 Results in terms of the required number of ccks.

consistent instances and inconsistent instances. Therefore, the number of pruned values vary for both approaches. Table 1 shows that almost in all cases DRAC^{++}-2 prunes less values to prove the inconsistency of the instances. While for restricted path consistent instances, the two approaches prunes the same non-viable values.

Thus, performing partial RPC on arc-consistent hard instances allow us to discard more values (less than 10.00% for DRAC and more than 55.00% for DRAC^{++}).

For the exchanged number of messages DRAC^{++} may require a higher amount of messages to enforce RPC; this result can be vindicated by the fact that in the beginning of the second step, all the agents implied in at least one path should exchange their set of first support.

At this point, we can say that DRAC^{++} is worthwhile especially for over-constrained problems. This can be justified by the fact that for such problems, the probability of having a path of 3 variables in the CN is high compared to under-constraint problems leading, to the discovery of more path inconsistent values and consequently to more reductions.

5. Conclusion

Enforcing local consistency techniques has been shown to be very useful in improving solving process especially for hard CNs. However, only few distributed techniques are proposed but the majority of them tackle only arc-consistency property. Nevertheless, the harder the CN, the more useful and worthwhile the higher pruning level of local consistency techniques. Therefore, the objective of this paper is to achieve full global restricted path consis-

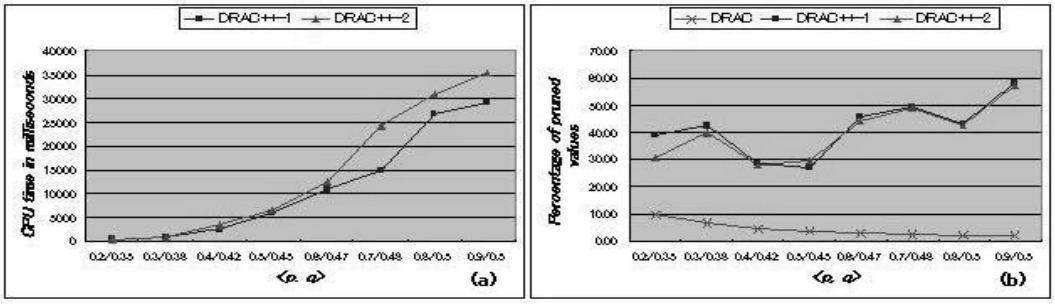

Fig. 3 Results in mean of CPU time and percentage of deleted inconsistent values.

Table 1 Results of the percentage of deleted values for the inconsistent instances

	$\langle$**0.2; 0.35**$\rangle$	$\langle$**0.3; 0.38**$\rangle$	$\langle$**0.4; 0.42**$\rangle$	$\langle$**0.5; 0.45**$\rangle$
DRAC^{++}-1	54.33%	71.4%	65%	62.5%
DRAC^{++}-2	49.16%	67%	65.5%	76%
	$\langle$**0.6; 0.47**$\rangle$	$\langle$**0.7; 0.48**$\rangle$	$\langle$**0.8; 0.5**$\rangle$	$\langle$**0.9; 0.5**$\rangle$
DRAC^{++}-1	71.7%	71.83%	78%	74.07%
DRAC^{++}-2	70.4%	70.83%	77.12%	72.85%

tency (RPC), for any binary problem and with the minimal amount of constraint checks.

Our approach consists of Constraint agents, which exchange their local restricted path inconsistent values in order to help themselves to reduce the domains of their variables. This process is performed until an equilibrium state is reached and corresponds to a failure relative to an absence of solutions or to a full global RPC. The experimental comparative evaluation shows that this approach is worthwhile especially for over-constrained problems.

In a future work we will try first to apply this protocol to any general CN, second to improve the enforced level of local consistency with low complexities.

Acknowledgments This work is supported, in a part, by the Japanese Foundation for C&C Promotion Grant for non-Japanese Researcher to Ahlem Ben Hassine.

References

1) Briot, J.P.: Actalk. A Framework for Object-Oriented Concurrent Programming - Design and Experience. *In Object-Oriented Parallel and Distributed Programming, Hermes Science Publications*, pp. 209-231 (2000).

2) Berlandier, P.: Improving Domain Filtering Using Restricted Path Consistency. *Proc. of IEEE CAIA-95*, pp. 32-37 (1995).

3) Bessiere, C., Freuder, E.C., Regin, J-C.: Using Inference to Reduce Arc Consistency Computation. *Proc. IJCAI-95*, pp. 592-598 (1995).

4) Bessiere, C., Freuder, E., Regin, J-C.: Using Constraint Metaknowledge to Reduce Arc Consistency Computation. *Artificial Intelligence*, Vol. 107, pp. 125-148 (1999).

5) BenHassine, A., Ghedira, K.: How to Establish Arc-Consistency by Reactive Agents. *Proc. of the 15th ECAI-02*, pp. 156-160 (2002).

6) BenHassine, A., Ghedira, K.: Using Reactive Agents to Establish Arc-Consistency. *Proc. of the 7th PRICAI-02*, pp. 97-107 (2002).

7) Chmeiss, A. and Jegou, P.: New Constraint Propagation Algorithms Requiring Small Space Complexity. *Proc. of the 8th IEEE ICTAI*, pp. 286-289 (1996).

8) Debruyne, R. and Bessiere, C.: Some Practical Filtering Techniques for Constraint Satisfaction Problems. *Proc. of IJCAI-97*, pp. 412-417 (1997).

9) Dechter, R., Pearl, J.: Tree-Clustering Schemes for Constraint-Processing. *Proc. AAAI-88*, pp. 150-154 (1988).

10) Lamport, L. and Chandy, K. M.: Distributed Snapshots: Determining Global States of Distributed Systems. *TOCS*, Vol. 3, No. 1, pp. 63-75, (1985).

11) Mackworth, A. K.: Consistency in Networks of Relations. *Artificial Intelligence*, Vol. 8, pp. 99-118 (1977).

12) Montanari, U.: NetWorks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences*, Vol. 7, pp. 95-132 (1974).

13) Singh, M.: Path Consistency revisited. *Proc. of the 7th IEEE ICTAI-95*, pp. 318-325 (1995).