How to Establish Arc-Consistency by Reactive Agents

Ahlem Ben Hassine [1] and Khaled Ghédira [1]

Abstract. The objective of this paper is to obtain the full global arc consistency of a CSP as a result of interactions between simple and reactive agents. Thus, a Multi-Agent model is proposed and discussed in terms of correctness, termination and complexity. This model consists of Constraint Agents in interaction by exchanging inconsistent values. A comparative analysis with AC-7 [2] is also done.

1 INTRODUCTION

Arc consistency techniques have shown a great interest in CSP which are known to be NP-Complete. They reduce the complexity by eliminating domain inconsistencies and consequently pruning the search space.

Informally, a binary CSP [9] 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 $D = \{D_1, .., D_n\}$ and a set of e constraints between these variables $C = \{C_{ij}, ..\}$, C_{ij} being a binary constraint between X_i and X_j. The constraints restrict the values the variable can simultaneously take. $R = \{R_{ij}, .. \}$ is the set of e relations, where R_{ij} is the set of allowed pairs of values for the corresponding C_{ij}. Solving a CSP consists in finding one or all-complete assignments of values to variables satisfying all the constraints.

A value a, from D_i, is supported by a value b, from D_j, along C_{ij} iff (a, b) satisfies C_{ij} (i.e. (a, b) belongs to the relation R_{ij} associated to C_{ij}), b is called a support for a along C_{ij}, we note that $S_{ij}(a, b)$. A value a from D_i is viable iff $\forall X_k$ such that $\exists C_{ik} \in C$, $k = 1 ...n$, there exists a support b for a in D_k.

A CSP is arc-consistent or 2-consistent if and only if for each variable $X_i \in X$ ($i = 1..n$), and for each value $a \in D_i$, a is viable. So arc-consistency achievement consists in transforming a CSP P (X, D, C, R) into another equivalent and more simple CSP P' (X, D', C, R) , where $D'_i \subseteq D_i \in D$. This is obtained by removing all and only arc inconsistent values in order not to affect the set of satisfiable assignments of the CSP. A CSP is consistent if it has at least one solution, otherwise, it is inconsistent.

There are two approaches to achieve arc-consistency: the centralized and distributed ones. Among the former, we quote arc-consistency applied to vision problems [12], AC-1, AC-2 and AC-3 algorithms [8], AC-4 [10], AC-5 [4], AC-6 [1], AC-Inference and AC-7 [2], and AC2000 and AC2001 [3].

In this paper, we are interested in the distributed approaches due to the natural distribution of many real CSP applications and the advents of both distributed computing and networking technologies. The most recent research proceeds by adapting classical arc-consistency techniques to the distributed framework: DisAC4 [11], DisAC6 and DisAC9 [6]. DisAC4 is a coarse-grained parallel algorithm designed on the basis of AC-4 and the DisCSP formalism [13], which defines an agent as responsible of a subset of variables. DisAC4 is used for a distributed memory computer using asynchronous message passing communication. Unfortunately, it has been restricted to diffusion Networks (Ethernet), which leads to an underlying synchronism between processes. The theoretical complexity is $O(\frac{n^2d^2}{k})$, where n is the number of variables, d is the size of the largest domain and k is the number of the processors.

As for DisAC6, it is based on AC-6 and DisCSP. The basic idea of this algorithm is to scatter the problem among autonomous processes and make them asynchronously interact by point-to-point messages containing useful information (in order to perform the global arc-consistency). The worst time complexity is $O(n^2d^3)$ and the space complexity is $O(n^2d)$ with $O(nd)$ the amount of message operations. DisAC9 is an improvement of DisAC6. It is an optimal algorithm in the number of message passing operations. It exploits the bidirectionality property of constraint relations, which allows agents to induce acquaintances relations. The worst time complexity of this algorithm is $O(n^2d^3)$ with nd messages and with a total amount of space in $O(n^2d)$.

In a different way:

- Our approach (that we call DRAC for Distributed Reinforcement of Arc Consistency) does not rely on any existing centralized algorithm.
- It is based on a Multi-Agent system associating an agent per constraint.
- It uses dual constraint-graphs to represent CSPs. A binary 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

[1] Laboratoire URIASIS, Institut Supérieur de Gestion de Tunis-Tunisie, email: {Ahlem.BenHassine, Khaled.Ghedirag@isg.rnu.tn

constraint-graph [5]. The primal constraint-graph represents variables by nodes and associates an arc with any two nodes residing in the same constraint. A dual constraint-graph represents each constraint by a node and associates a labeled arc with any two nodes that share at least a variable. The arcs are labeled with the shared variables.

- It directly addresses generalized CSPs without transforming the initial problem into a binary one. It is known that this transformation procedure increases both the temporal and spatial complexity. So, we expect that the use of the dual graph associated to both the "agent , constraint" assignment and the point-to-point asynchronous message passing protocol would be very appropriate in order to directly achieve arc-consistency for generalized CSPs.

Note that the goal of DisAC9 is essentially to reduce the total amount of messages by doing more local computations, because of the high cost of messages passing in a distributed multiprocessor architecture. As we intend to use a mono-processor machine, we ignore the cost of messages passing, and rather focus on reducing the local agent computation. So, our objective is di®erent: we look for obtaining the full global arc-consistency as a result of the interactions between the Constraint Agents by exchanging inconsistent values. In other words, the full global arc-consistency is obtained as a side e®ect of the interactions between reactive agents; each having a local goal. As a starting point of our whole research, we focus on binary CSPs.

This paper is organized as follows. First we present the Multi-Agent architecture and its global dynamic. Second, we prove the correctness and the termination properties, then we compute the complexity. Finally, we exhibit the experimental results.

2 MULTI-AGENT ARCHITECTURE

This approach involves two kinds of agents (Constraint agents and Interface agent) communicating by asynchronous point-to-point messages. The last agent has been added in order to detect whether the full global arc-consistency has been achieved and, especially, to inform the user of the result.

Each agent has a simple structure: acquaintances (the agents that it knows), a local memory composed of its static and dynamic knowledge, a mailBox where it stores the received messages and a behavior. In the proposed model, agents communicate by sending messages. An agent can send a message to another one only if it knows it (it belongs to its acquaintances). For the transmission between agents, we assume that messages are received in the order they are sent. The messages delivering time is ¯nite.

2.1 Constraint agents

Each agent has its own [2] variables, its acquaintances consist of both all the agents with which it shares a variable, and the Interface agent. Its acquaintances and its associated relation de¯ne its static knowledge, while its dynamic knowledge concerns its internal state, the domains of its own variables and a parameter called EndBehavior which speci¯es whether its behavior is completed or not.

[2] The variables implied in this constraint.

2.2 Interface agent

The Interface agent has as acquaintances all the Constraint agents of the system, denoted by $_i$, which represent its static knowledge. Its dynamic knowledge consists of the internal state of all its constraints.

3 MULTI-AGENT DYNAMIC

The objective is to transform a CSP P (X, D, C, R) into another equivalent CSP P'(X, D', C, R). P' is obtained as a result of the interactions between the Constraint agents which are trying to reduce their domains.
Before detailing these interactions and the underlying global dynamic, we present the communication protocol, the data structures and the basic primitives relative to an agent C_{ij}.

3.1 Communication protocol

The communication protocol is based on the two following message passing primitives.

- SendMsg(Sender, Receiver, "Message") where Receiver can be more than one.
- GetMsg() extracts the ¯rst message from the mailBox.

As far as the exchanged messages are concerned, the Multi-Agent dynamic involves three types (without considering the messages relative to the detection of the equilibrium state) namely:

- "Start" message, sent by the interface to all the agents in order to activate them,
- "ReduceDomains of" message, sent by a Constraint agent to its acquaintances in order to propagate its deleted values.
- "StopBehavior" message sent by a Constraint agent, which has a domain wipe-out, to the interface.
- "StopLocalBehavior" message sent by the interface to all the agents of the system to make them stop their local behavior.

3.2 Data structures

- Acquaintances$_{X_i}$ (resp. Acquaintances$_{X_j}$) = the set of Constraint agents sharing the variable X_i (resp. X_j) with C_{ij}.
- $D_i^{C_{ij}}$ and $D_j^{C_{ij}}$ represent the local view of respectively D_i and D_j. Both are supposed to be totally ordered. $D_i^{C_{ij}}$ (resp. $D_j^{C_{ij}}$) is called the occurrence of D_i (resp. D_j).
  Note that some occurrences of a given D_i may be di®erent, but all occurrences of D_i $8i \in 1..n$ must be identical when the full global arc-consistency is reached (this property will be proved in the subsection 4.1). At this stage, let us refer to the ¯nal obtained domain $D_i^{C_{ij}}$ (resp. $D_j^{C_{ij}}$) by $fD_i^{C_{ij}}$ (resp. $fD_j^{C_{ij}}$).

- $SP_{x_i x_j}$ = f(a b y) such that a 2 $D_i^{C_{ij}}$, b 2 $D_j^{C_{ij}}$ and y 2 f0, 1g j if y =0, b is the ¯rst support of a. Otherwise b is one support of ag.
- $TestedValue_{X_i}$ (resp. $TestedValue_{X_j}$): the set of the current viable values of X_i (resp. X_j).
- $InconsistentValue_{X_i}$ (resp. $InconsistentValue_{X_j}$): the set of the current non-viable values of X_i (resp. X_j).
- EndBehavior : a Boolean parameter that indicates whether the agent behavior is ¯nished or not.

3.3 Basic Primitives

- addTo($SP_{x_i x_j}$, (a b y)) : insert (a b y) in the set $SP_{x_i x_j}$,
- First($D_i^{C_{ij}}$) : returns the ¯rst value in the domain of $D_i^{C_{ij}}$,
- Last($D_i^{C_{ij}}$) : returns the last value in $D_i^{C_{ij}}$ if $D_i^{C_{ij}}$ 6= ;,
- Next(a, $D_i^{C_{ij}}$) : returns the ¯rst viable value occurring after a in $D_i^{C_{ij}}$ if a 6= Last($D_i^{C_{ij}}$) else returns nil,
- FirstSupport(a, $D_j^{C_{ij}}$, h) : returns the ¯rst support of a value a in $D_j^{C_{ij}}$ greater or equal to h according to C_{ij} , if it exists, else returns nil.

3.4 Global Dynamic

At the initial state, the Interface agent creates all the Constraint agents and activates them (¯gure 1.). Each agent C_{ij} reduces the domains ($D_i^{C_{ij}}$ and $D_j^{C_{ij}}$) of its own variables X_i and X_j by computing local viable values (see x1) for both X_i and X_j. To achieve this, C_{ij} looks for one support (the ¯rst one) for each value of its variables. When the ¯rst support b 2 $D_j^{C_{ij}}$ of a value a 2 $D_i^{C_{ij}}$ relatively to C_{ij} is found, then (a b 0) is added to the list of supports $SP_{x_i x_j}$ (¯gure1.line7.), and respectively, when the ¯rst support c 2 $D_i^{C_{ij}}$ of a value b 2 $D_j^{C_{ij}}$ is found then (c b 1) is added to the list of supports $SP_{x_i x_j}$ (¯gure2.line15.), i.e. b could not be the ¯rst support of c. A value a is deleted from $D_i^{C_{ij}}$ if and only if a has no support in $D_j^{C_{ij}}$.

Each agent uses the bidirectionality property of constraints relations: a 2 $D_i^{C_{ij}}$ supports b 2 $D_j^{C_{ij}}$ (S_{ji}(b, a)) if and only if b 2 $D_j^{C_{ij}}$ supports a 2 $D_i^{C_{ij}}$ (S_{ij}(a, b)). This property, already used by AC-7, allows us to avoid checking for S_{ji}(b, a) if S_{ij}(a, b) has already been successfully checked, i.e. a is also a support for b.

At the end of this computation, deleted values are announced to related acquaintances (¯gure1.line 21. and 23.). Each agent that has received this message starts processing it. It ¯rst updates the domains of its variables by deleting non viable received values (¯gure2. line3.). Afterwards, it updates computed support information (¯gure2. line 5.). In the case where a is a non-viable value, and if the value of y is 0, the agent looks for another support for b in $D_i^{C_{ij}}$ (¯gure2.line9.and 11.) that occurs after a (as AC-7). Otherwise it looks for a support from scratch i.e. the ¯rst value in $D_i^{C_{ij}}$ (¯gure2. line10. and11.). This can lead to a new deletion of values (¯gure2. line14.) and by consequence to new outgoing messages (¯gure2. line19.).

Thus reducing domains on an agent may, consequently, cause an eventual domain reductions on another agent. Therefore, these interactions must carry on until the stable equilibrium state, where all the agents are de¯nitely satis¯ed and consequently no more reduction is possible.

```
Begin
1.  SP_{x_i x_j} ← ∅; EndBehavior ← False;
2.  InconsistentValue_{X_i} ← D_i^{C_ij}; InconsistentValue_{X_j} ← D_j^{C_ij};
3.  TestedValue_{X_i} ← ∅; TestedValue_{X_j} ← ∅
4.  For each (a, b) ∈ R_ij do
5.    If ((a ∈ D_i^{C_ij}) AND (b ∈ D_j^{C_ij}))
6.    Then If (a ∉ TestedValue_{X_i})
7.           Then addTo (SP_{x_i x_j}, (a b 0));
8.                TestedValue_{X_i} ← TestedValue_{X_i} ∪ {a};
9.                InconsistentValue_{X_i} ← InconsistentValue_{X_i} /{a};
10.               If (b ∉ TestedValue_{X_j})
11.               Then InconsistentValue_{X_j} ← InconsistentValue_{X_j} /{b};
12.                    TestedValue_{X_j} ← TestedValue_{X_j} ∪ {b};
13.          Else If (b ∉ TestedValue_{X_j})
14.               Then addTo (SP_{x_j x_i}, (a b 1));
15.                    TestedValue_{X_j} ← TestedValue_{X_j} ∪ {b};
16.                    InconsistentValue_{X_j} ← InconsistentValue_{X_j} /{b};
17. D_i^{C_ij} ← TestedValue_{X_i}; D_j^{C_ij} ← TestedValue_{X_j};
18. If D_i^{C_ij} = ∅ OR D_j^{C_ij} = ∅
19. Then SendMsg(C_ij, Interface, " StopBehavior"); EndBehavior ← True;
20. Else For each C_ik ∈ Acquaintances_{X_i} do
21.          SendMsg(C_ij, C_ik, "ReduceDomains: InconsistentValue_{X_i} of X_i");
22.      For each C_jk ∈ Acquaintances_{X_j} do
23.          SendMsg(C_ij, C_jk, "ReduceDomains: InconsistentValue_{X_j} of X_j");
End
```

Figure 1. "Start" message (Executed by each Agent (C_{ij}))

```
ReduceDomains: DelVal of X_i
Begin
1.  InconsistentValue_{X_i} ← ∅;
2.  For each (a ∈ DelVal) AND (a ∈ D_i^{C_ij}) do
3.    D_i^{C_ij} ← D_i^{C_ij} / {a};
4.    S → the set of all values b in SP_{x_i x_j} having a as a support;
5.    Delete from SP_{x_i x_j} all tuples (a b y);
6.    For each b ∈ S do
7.      If b has not another support in SP_{x_i x_j}
8.      Then If a is the first support of b (y = 0)
9.           Then h ← Next (a, D_j^{C_ij});
10.          Else h ← First (D_j^{C_ij});
11.          c ← FirstSupport (b, D_i^{C_ij}, h);
12.          If c = nil
13.          Then InconsistentValue_{X_j} ← InconsistentValue_{X_j} ∪ {b};
14.               D_j^{C_ij} ← D_j^{C_ij} / {b};
15.          Else addTo (SP_{x_j x_i}, (c b 1));
16. If D_i^{C_ij} = ∅ OR D_j^{C_ij} = ∅
17. Then SendMsg(C_ij, Interface, " StopBehavior"); EndBehavior ← True;
18. Else For each C_jk ∈ Acquaintances_{X_j} do
19.      SendMsg(C_ij, C_jk, " ReduceDomains.InconsistentValue_{X_j} of X_j");
End
```

Figure 2. "ReduceDomains of" message (Executed by each Agent (C_{ij}))

An agent is satis¯ed when it has no more reduction to do on its variable domains or when one of its reduced domain wipe-out (¯gure1. line18. and ¯gure2. line16.). But it is clear

that this satisfaction state is not definitive. Indeed, if there exists at least one unsatisfied Agent C_{ik}, it may cause the unsatisfaction of other Constraint agents and this is due to the propagation of constraints. So, interactions and especially reductions must carry on. Note that this dynamic allows a premature detection of failure: absence of solutions. Thus, in the case of failure, the "StopBehavior" message is sent by the constraint (which has detected this failure) to the interface in order to stop the whole process. In this case, the Interface agent in turn send a "StopLocalBehavior" message to each constraint to make them stop their local activity (their attribute EndBehavior is set to true) and informs the user of the absence of solutions.

The maximal reinforcement of global arc-consistency is obtained as a side effect from the interactions described above.

3.5 Agent behaviors

3.5.1 Constraint agent behavior

There are two cases where a Constraint agent is satisfied:

- When one of its domains is empty. In this case, it asks the interface to stop the whole process and to communicate the failure result to the user.
- When all possible local reductions are done to take into account the just received messages containing the values deleted by the other Constraint acquaintances. In this case, it updates its internal state.
- Otherwise, i.e. in the case of unsatisfaction behavior, it sends a message containing inconsistent values to the concerned acquaintances: SendMsg(self, Acquaintances , "ReduceDomains of").

3.5.2 Interface agent behavior

When all the agents are satisfied or when it has received a failure message, the Interface agent is satisfied and in this case it makes all the agents stop their local behavior: SendMsg(self, i , "StopLocalBehavior"), and communicates the obtained result to the user.

Otherwise, i.e. in the case of unsatisfaction behavior, it checks the system state, using the algorithm described in [7].

4 CORRECTNESS, TERMINATION AND COMPLEXITY

4.1 Correctness

The objective of this sub-section is to show that our approach leads to the full global arc-consistency. For this, we have to prove the following assertions:

- $\forall i \in 1..n, \forall j \neq k, f D_i^{C_{ij}} = f D_i^{C_{ik}}$:
- $\forall i \in 1..n, \forall C_{ij} \in C, \forall val \in f D_i^{C_{ij}}$ (resp. $f D_j^{C_{ij}}$); val is viable.
- $\forall i \in 1..n, \forall C_{ij} \in C, \forall val \in D_i^{C_{ij}}$ (resp. $D_j^{C_{ij}}$), if val is viable then $val \in f D_i^{C_{ij}}$ (resp. $val \in f D_j^{C_{ij}}$).

In fact, the first assertion concerns the process of deleted values propagation. Since $C_{ik} \in Acquaintances_{X_i}$ of C_{ij} (and conversely) and since all the messages are received in a finite period of time and in the same order as they were sent, C_{ij} (resp. C_{ik}) has to be informed by each deleted value[3]. Then the agents will have the same final domains $f D_i^{C_{ij}}$ and $f D_i^{C_{ik}}$.

The second assertion concerns the correctness of the ReduceDomains procedure. Each time the deletion of a value (from $D_j^{C_{ij}}$) leads to a non-viable value in the domain of a variable X_i. The agent C_{ij} sends a message to all the concerned acquaintances C_{ik} in order to update their X_i domain. So, all the non-viable values are deleted from the domain of all the agents.

For the third assertion, there are two cases where a value a is deleted from the domain of a variable X_i. The first is that the agent C_{ij} has detected that a has no support in $D_j^{C_{ij}}$. Therefore, a is a non-viable value and must be discarded. The second case is when the agent C_{ij} has received a message to update the domain of X_i by deleting the value a. Thus, this value has been detected as non-viable by the agent which sends the message. Consequently, only non-viable value will be deleted.

4.2 Termination

The dynamic of DRAC approach stops when the system reaches its stable equilibrium state. At this state, all the agents are satisfied. An agent is satisfied when it has no more reductions to do on its variable domains or when one of its related new reduced domains is wipe-out. The detection of the stable equilibrium state is achieved by using the well known algorithm of [7], a state where all agents are waiting for a message and there is no message in the transmission channels. If all the agents of the system are in the state of waiting, and there exists only one agent C_{ij} which has deleted one value a from the domain of one of its variables (X_i or X_j). We assume that this agent shared this altered variable with another agent C_{ik}. The latter must be informed of the loss of the value a in order to propagate the constraints. Hence, there is a message in transit for it, which invalidates our transmission hypothesis.

4.3 Complexity

Let us consider a CSP P having n for the total number of variables, d for the size of the variable domains and e for the total number of constraints. The number of Agents is e. If we consider a fully connected constraint network, we will have e-1 acquaintances for each Constraint agent. Each agent C_{ij} maintains a list $SP_{x_i x_j}$ of supports, with the size of 2d-1 in the worst case. Since there are e agents, the total amount of space is (2d-1)e (for a fully connected graph, e will be set to n(n-1)/2, in the worst case). So the space needed is (n(n-1)/2)*(2d-1) ' $O(n^2 d)$. This space is the same as that of AC7 one's.

[3] Let us recall that the deleted values must be immediately transmitted to the concerned acquaintances.

The worst case in the execution time of a distributed algorithm occurs when it proceeds with a sequential behavior. For our model, this occurs when only one value is deleted at a time. This leads to nd successive deletions. Our approach is composed of two steps; the ¯rst one is the initializing step, where each agent performs d^2 operations to generate the support sets. For each deleted value, the agent will perform $O(d^2)$ operations to search another support for this value. Thus, each agent performs $O(d^2)$ operations.

So the total time complexity of DRAC (with e agents and nd successive deletions), in the worst case, is $O(end^3)$. This complexity is equal to that of DisAC-9 down to the number of variables.

5 EXPERIMENTAL RESULTS

The implementation was developed with Actalk, an object based on concurrent programming language with Smalltalk-80 environment. In this language framework, an agent is implemented as an actor having the Smalltalk object structure enriched by an ability to send/receive messages to/from its acquaintances, bu®ering the received messages in its own mailbox. The DRAC e±ciency is assessed through a comparison with AC7 [2](¯gure 3.) on the basis of randomly generated samples which belong to the transition phase, which consists of both arc consistent problems and arc inconsistent problems. Each sample is designed on the base of the following parameters : n = 20, d = 10 , hp=qi where p (resp. q) represents the density (resp. the tightness). The ¯gure 3. shows that AC7 performs more constraint checks than DRAC, especially for the problems where arc-consistency establishing often succeed (for example h0; 4=0; 4i and h0; 9=0; 5i). Despite the use of the bidirectionality by both AC7 and DRAC, DRAC requires less constraint checks than AC7. This advantage is due to the fact that the main operation of AC7 consists in seeking a support for each value. Thus, a constraint check is needed for each value except for the case where the bidirectionality property can be applied, whilst the DRAC init step (Figure 1.) consists in exploring the relation of each constraint only once in order to generate the $SP_{x_i x_j}$ sets. Note that this advantage is due to the fact that the relations are expressed in extension, i.e. by authorized couples of values, which can be examined without any additional computations. Therefore, DRAC is more appropriate than AC7 in this case of relations. Otherwise, the experimental results would be expected to be similar.

	<0,2/0,3>	<0,3/0,3>	<0,4/0,4>	<0,5/0,4>
		(Phase Transition)		
AC7	1981,1	2107,9	2937,5	3879,4
DRAC	461,5	1343,2	287,7	661,4
	<0,6/0,4>	<0,7/0,4>	<0,8/0,4>	<0,9/0,5>
		(Phase Transition)		
AC7	4909,1	5416,3	5797,6	5626,2
DRAC	2006,6	2991,4	3579,9	321,8

Figure 3. DRAC vs. AC7 Results in mean number of Constraint Checks on a Pentium III, 800Mhz
(10 instances are generated for each set of hp=qi parameters)

6 CONCLUSION

The objective of this paper is to achieve full global arc-consistency in a totally distributed way without any help from centralized algorithms. A Multi-Agent approach, that we have called DRAC, has been proposed. Its correctness and termination have been proved. The spatial complexity is similar to AC7's and the temporal complexity is equal to DisAC-9's down to the number of variables.

Our approach consists of Constraint agents, which exchange their local inconsistent values in order to help themselves reduce the domains of the variables that they involve. 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 arc-consistency. Thus, this state is obtained as a side e®ect of the interactions between the Constraint agents whose behaviors are simple and reactive.

As we associate an agent per Constraint, the dual constraint-graph is proved to be well appropriate to represent the agent network. Consequently, any generalized CSP can be naturally and directly (without any non-binary =) binary transformation) handled by DRAC. This will be the main object of our perspectives.

REFERENCES

[1] Bessiere, C. 1994. Arc consistency and arc consistency again. Arti¯cial Intelligence 65 (1994), 179-190.

[2] Bessiere, C.; Freuder, E. and Régin, J-C. 1999. Using constraint Metaknowledge to Reduce Arc Consistency Computation. Arti¯cial Intelligence (1999), Vol. 107, p125-148.

[3] Bessiere, C.; Régin, J-C. 2001. Re¯ning the Basic Constraint Propagation Algorithm. In Proceedings IJCAI'01.

[4] Deville, Y. and Hentenryck, P. V. 1991. En e±cient arc consistency algorithm for a class of CSP problems. In Proceedings IJCAI'91, pp. 325-330, Sydney, Australia.

[5] Dechter, R. and Pearl, J. 1988. Tree-Clustering Schemes for Constraint-Processing. In AAAI-88.

[6] Hamadi, Y. 1999. Optimal Distributed Arc-Consistency. Constraint Programming 1999, p. 219-233.

[7] Lamport, L. and Chandy, K. M. 1985. Distributed snapshots: Determining global states of distributed systems. TOCS, 3(1) : 63-75, Feb 85.

[8] Mackworth, A. K. 1977. Consistency in networks of relations. Arti¯cial Intelligence, 8, 99-118.

[9] Montanari, U. 1974. NetWorks of Constraints: Fundamental properties and applications to picture processing. Information Sciences, 7, P.95-132.

[10] Mohr, R. and Henderson, T. C. 1986. Arc and path consistency revisited. Arti¯cial Intelligence, 28, p225-233.

[11] Nguyen, T. and Deville, Y. 1995. A Distributed Arc-Consistency Algorithm. In : First Int. Workshop on concurrent Constraint Satisfaction.

[12] Waltz, D. L. 1972. Understanding Line Drawings of Scenes with Shadows. in: the Psychology of Computer Vision, McGraw Hill, 1975, 19-91 (¯rst published in:Tech.Rep. AI271, MIT MA, 1972).

[13] Yokoo, M.; Ishida, T. and Kuwabara, K. 1990. Distributed Constraint Satisfaction for DAI Problems. In the 10th Int. Workshop on Distributed Arti¯cial Intelligence.