ANNALS OF THE FACULTY OF ENGINEERING HUNEDOARA

2004, Tome II, Fascicole 2

ON THE COMPLETENESS OF THE ALGORITHMS DISTRIBUTED WITHIN THE DCSP MODELING (DISTRIBUTED CONSTRAINT SATISFACTION PROBLEM)

MUSCALAGIU Ionel, PĂNOIU Manuela, OSACI Mihaela

UNIVERSITY „POLITEHNICA” TIMISOARA,
FACULTY OF ENGINEERING OF HUNEDOARA, ROMANIA

Abstract

One of the basic characteristics to be met by any acceptable algorithm is its completeness. The problem makes more sense within the DCSP distributed framework, where the agents act concurrently and asynchronously, each agent being in the position of making decisions that influence the decisions of the other agents. In this article we will analyze the completeness of some algorithms distributed asynchronously, to be found in the reference literature: the classic backtracking algorithm distributed asynchronously (Asynchronous Backtracking), its improved variant (Asynchronous Weak–Commitment Search) and Hamadi's distributed backtracking algorithm (Distributed Backtracking). We will point out to the fact that the first two are complete algorithms, unlike the third, for which the completeness has not been demonstrated yet.

Keywords

Artificial intelligence, distributed programming, constraints, agents.

1. INTRODUCTION

One of the basic characteristics to be met by any acceptable algorithm is its completeness. It consists in knowing whether the algorithm has an end, i.e. whether it does not lead to an infinite loop, without giving an answer. In other words, we are interested in finding out whether the algorithm, for a certain instance, can lead to an answer, either affirmative (i.e. to find a solution) or negative (pointing out that there is no solution).

This characteristic is very important for any algorithm. The problem makes sense particularly in the case of the distributed framework, as, for instance, in the situation of agent modeling, if one agent keeps on changing its values and never reaches a stable condition, the algorithm will enter a loop. Things become even more complex in the distributed context, where agents act concurrently and asynchronously, each agent being in the position of making decisions that influence the decisions of the other agents.

In this chapter we will analyze the completeness of the most important asynchronous distributed algorithms . We will introduce the way in which each DCSP technique ensures the completeness of the algorithm.

We are talking about Yokoo's classical asynchronously distributed backtracking algorithm (AB–Asynchronous Backtracking), (AWCS-Asynchronous Weak–Commitment Search) published in [6] and about Hamadi's distributed backtracking algorithm (DIBT - Distributed Backtracking) [2,3] . For the first two, the authors have demonstrated their completeness, whereas for the third one, there is no such demonstration.

2. THE FRAMEWORK.

In order to carry out the completeness analysis, we will introduce in this paragraph a few notions that are to be found in the reference literature in relation to DCSP modeling [5] .

Definition 1.- CSP model. The model based on constraints CSP-Constraint Satisfaction Problem, existent for centralized architectures, consists in:

- *n* variables x_1, x_2, …, x_n, which can take finite values, within several finite, discrete domains D_1, D_2,…, D_n .
- a set of constraints among these variables .

Solving a CSP supposes finding an association of values for all variables so that all constraints be met (realized).

Definition 2.- DCSP model. One problem of meeting the distributed constraints (DCSP) is a CSP, in which the variables and the constraints are distributed among autonomous agents that communicate by message exchange.

3. COMPLETENESS IN CASE OF ABT ALGORITHM

The first algorithm we are going to analyze from the point of view of its completeness is Yokoo's classical backtracking algorithm. In this algorithm, every agent instantiates its variable concurrently and sends its value to the agents it is directly connected to, further waiting for the messages to be answered. This behavior, in which each agent keeps changing the value, leads to the question whether the algorithm has an end, i.e. if it does not end up in a loop. For example, a looping instance is the one where x_1 obliges x_2 to change its value, the changing of x_2 causing the value of x_3, to change and x_3 determining the change of x_1. Such situations can arise in the case of applying an asynchronous algorithm.

In [5, 6], the authors suggest one way of eliminating the loops in a network of constraints, based on a technique using a unique identifier, technique used for avoiding dead ends in systems with distributed databases. This technique consists in using a relationship of complete order among the knots. If a knot has a unique identifier, one can define a priority order of the agents by using the alphabetical order of the respective identifiers (the alphabetically preceding agent having a higher priority). If a link is directed by means of this priority order (from the agent having a higher priority to the one having a lower priority), then no loop can appear in the network. This means that for each constraint, the lower priority agent will be the estimator and the higher priority agent will OK the message of the estimator.

The authors of the algorithm demonstrate in [5,6] that the algorithm is complete. They point out that if there is a solution, the algorithm leads to a stable solution where all the values of the variables meet all the constraints and all the agents are in wait of messages. Also, if there is no solution, they point out that the algorithm reveals this situation and closes.

4. THE COMPLETENESS OF THE AWCS ALGORITHM - ASYNCHRONOUS WEAK–COMMITMENT SEARCH

The next algorithm to be analyzed is the asynchronous search algorithm, in its improved variant. We have seen in the previous paragraph that the completeness problem was solved for the basic algorithm (the asynchronous algorithm), which this new algorithm is derived from.

The AWCS algorithm is a hybrid one, obtained by combining the ABT algorithm with the WCS one, existent for CSP, within centralized architectures. It can be considered as an improved variant of ABT, due to the priority change. It purposely aims at stocking all the nogood values in order to ensure the completeness of the algorithm, but also to avoid instable situations.

In [6] the authors show that this new algorithm can be built by a dynamic change of the priority order.

The AWCS algorithm uses, as ABT, the two types of messages, OK? and nogood, with the same signification. There is a major difference in dealing with the OK? message. In case of receiving it, if the agent cannot find a value for its variable to be consistent with the values of higher priority variables, it no longer generates and sends the nogood message, but increases the priority, in order to maximize it with respect to the neighbors.

By means of the rules mentioned above, when a backtracking arises, the priority order will be changed in such a way that the agent having higher priority before the backtracking should meet the constraints of the agent generating the backtracking and having now a higher priority value. Moreover, in the asynchronous backtracking algorithm, the agents are trying to avoid situations labeled as nogood. Yet, because of the delays that might arise on message transmittal, the view agent set of an agent can occasionally be a superset of the values previously found as nogood. In order to avoid the effects of unstable situations and of those where useless changes of priority value were operated, each agent keeps a record of the nogood situations reported. When the *agent view* list is identical to the *nogood* one, the agent will not change the priority value, but will wait for the next message.

As to the problem of its completeness, we have noticed that it was raised and solved for the basic algorithm (the asynchronous algorithm) out of which this one is derived. When can blocking situations arise? The priority of values is changed if and only if a non-solution is found. But the number of nogood value combinations is finite (even if large), the value priority cannot be changed at infinitum. Therefore, according to the authors of the algorithm, after a certain, surely finite, period of time, the priority of values is bound to be stable. In [6], the authors show that the situation mentioned above cannot arise in the situation in which value priority is stable.

We have to point out that the AWCS algorithm itself needs record keeping of each nogood list, in order to ensure the completeness of the algorithm, giving the impression of an inefficient exponential space in real media. Yet, the experiments show its greater efficiency, as compared to the ABT algorithm.

Therefore, AWCS is efficient and complete by record keeping of all nogoods (whose number is smaller), but suffers if there is an outburst in the appearance of nogood values. Thus, the costs of constraint checking can be rather high, as in AWCS, an agent can generate nogoods for all its neighbors. One last idea, related to practice, is to limit the number of nogood recordings to a fixed value, sacrificing the completeness, but leading to more rapid results.

5. COMPLETENESS IN THE CASE OF DISTRIBUTED BACKTRACKING.

Another algorithm for asynchronous search is the **Distributed Backtracking**, published in [2, 3]. This is a variant of algorithm that does not imply add-link-type messages and eliminates completely the record keeping of nogood values.

The variant of distributed backtracking (DIBT- Distributed Backtracking) is based on the classical backtracking algorithm, the centralized case. If the ABT asynchronous algorithm uses learning schemata, this one eliminates them.

This is a synchronous algorithm, but it needs a certain order for the agents applying the backtracking schema, in order to ensure the completeness of the algorithm. Hence, a partial order among the agents is to be sought, in order to be used at initiating the variables and which will be further extended to a total order with the aim of guiding the backtracking steps. As there are no restrictions regarding the order to be used, we will be able to decide upon the use of a certain order, which best fits the topology of the constraint graph. This fact results in the restriction of the search space and of the number of exchanged messages. The author defines in [3] a generic method, named DAO, in order to determine the order of the distributed variables. This order, as well as the lexicographic order in ABT, is vital for ensuring completeness.

In [3] is introduced a demonstration schema for the correctness of the algorithm. The author points out to a few elements meant to ensure the demonstration of the correctness of the algorithm, but they are unfortunately not enough. We are going to analyze them further on. The author says that irrespective of the heuristic used for the order of variables, the oriented graph induced by set Γ in algorithm DAO, has no circuit. Thus, the backtrack step (respectively the instantiating one) from one agent cannot lead to the receiving of *bt* - type messages (respectively *infoVal*) from one of the children (respectively parents). During the backtrack step, it is only the remaining values that are tested for consistency, while during *infoVal* message receiving, the agent reconsiders its entire initial domain. Based on this observation, associated to the fact that message transmittal is done in finite time and also on the correctness of the detection of algorithm ending from **[LC85],** the author concludes that it is correct.

Unfortunately, while this algorithm is efficient from the point of view of memory, it fails some valid situations, being unable to find the solution. For instance, for the network with 5 agents given in figure 1, the algorithm is incomplete.

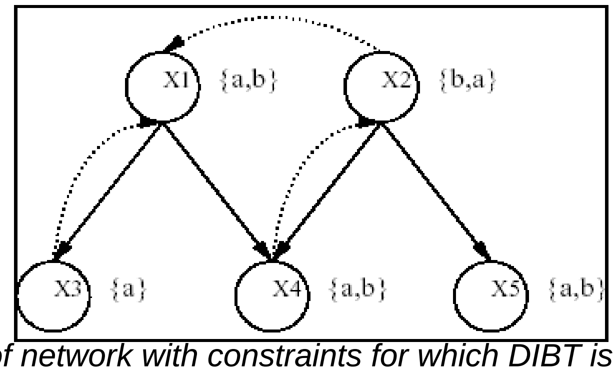

Fig.1. *Example of network with constraints for which DIBT is not complete*.

The links among variables represent constraints of inequality type. The domains are formed of values a, b, as it can be seen in figure 1. DIBT can run in the following way:

- x_1 and x_2 select a, respectively b and inform their children (see fig. 1.).
- x_3 has no other choice than value a so that a backtrack is generated to x_1 while x_4 performs a backtrack to x_2 because of the value conflicts.
- x_1 and x_2 select b, respectively a and inform their children.
- x_4 performs a backtrack to x_2 whose domain is empty, then a backtrack to x_1 follows, leading to inconsistency.

This way of running of algorithm DIBT leads to the impossibility of finding a solution, although it exists (a,a,b,b,b). One cause of the fail is that x_2 ignores the behavior of x_1 and does not reset its domain when x_1 changes its value from a into b.

In order to eliminate these difficulties and ensure the completeness of the algorithm, in [2] the author extends sets Γ^+ and Γ^-, named the sets of children and parents, in order to be sure that the *InfoVal()* message will reset all the relevant domains, and the *btSet()* message will beable to identify, for all backtracks, the guilty ones, adding in advance a few ABT-type links.

Unfortunately, in [7], Yokoo shows that this algorithm is incomplete. This is due to the fact that when it checks if a nogood value is old, algorithm DIBT takes into consideration its own values only, neglecting the values in the parent sets. This can lead to false deductions, such as the unification of several nogood values from different contexts. One example of such behavior is given in figure 2.

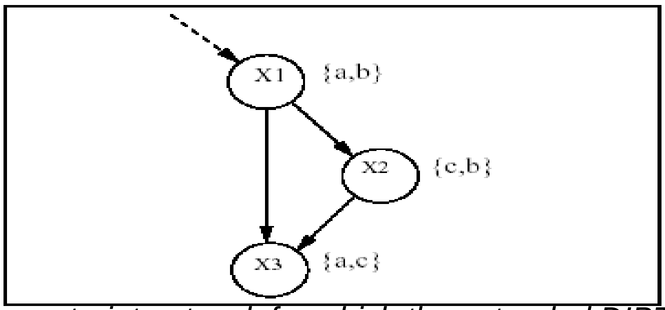

Fig.2. *Example of constraint network for which the extended DIBT is not complete.*

We consider, as in the previous example, the existence of inequality constraints among the link knots. DIBT can run in the following way :

- x_1 selects value a and informs its children (notice x_2 and x_3), x_2 selects value c and informs its child (x_3). In exchange, x_3 has an empty domain, having nothing left to choose. Therefore, it performs a backtrack to x_2 with value nogood (x_1=a and x_2 =c) .(**P1**)
- in the meantime, x_1 is bound to change its value because of the messages coming from the parents, eliminating value a. x_1 selects value b and informs its children, making x_2 eliminate value b from its domain. (**P2**)
- the message sent by x_3 to step 1 now reaches x_2. Unfortunately, this nogood is old in the context of the agents (x_1 no longer has value a but b), but because local values are not checked again, it is processed. Therefore, is make to backtrack with value nogood (x_1=b). (**P3**)
- now, the domain of x_1 is empty, x_1 also performing backtrack. And thus, not finding any solution (x_1=a, x_2 =b şi x_3=c). (**P4**)

In [1] a patch is suggested for fixing this bug. It consists in checking once again the incoming nogood messages, within the new context: each agent must be sure that the incoming and outgoing item of information is consistent with its local view. As a conclusion, we notice that the complete wipe out of the ABT-type additional links and of the nogood values is not 100% possible. Yet, the algorithm DIBT a needed to add a few ABT-type links and some nogood values, in order to ensure completeness (according to the elements mentioned above).

6. CONCLUSIONS

Each of the three techniques presented above is based on certain elements meant to ensure the completeness of the algorithm.

The ABT algorithm ensures completeness by means of a technique based on a unique identifier, technique used to avoid the dead ends in systems with distributed databases. This technique consists in using a relationship of total order among the knots. If a knot has a unique identifier, a priority order can be defined among the agents by using an alphabetical order of those identifiers (the alphabetically preceding agent has a higher priority with respect to the other agent).

The AWCS algorithm needs the record keeping of nogood lists in order to ensure the completeness of the algorithm. The main problem consists in changing the priority of agents. But value priority is changed if and only if one nogood is found. But, the number of nogood value combinations is finite (even if large), value priority cannot be changed at infinitum. Therefore, according to the authors of the algorithm, after a certain time, which is surely finite, value priority will be stable.

The DIBT algorithm needs no add-link-type messages, eliminating completely the record keeping of nogood values and uses a method of ensuring order among the agents applying the backtracking schema, meant to ensure the completeness of the algorithm. Unfortunately, in [7], Yokoo shows that this algorithm is incomplete. This is due to the fact that when verifying if a nogood value is old, algorithm DIBT takes into calculation only its values, ignoring the values in the parents' set.

As a conclusion, we notice that the complete wipe out of ABT-type additional links and nogood values is not 100% possible. Nevertheless, algorithm DIBT needed some extra ABT-type add-links and some nogood values in order to be able to ensure completeness in the situation given in [7].

7. BIBLIOGRPHY

[1] Christian Bessière, Arnold Maestre, Pedro Meseguer. *Distributed Dynamic Backtracking*.In Proceedings of the CP'00 Workshop on DCSP, Singapore,2000.

[2] Hamadi Y. – Traitement des problemes de satisfaction de constraintes distributes. PhD theisis, Universitatea Motpellier II, 1999.

[3] Hamadi Y., Bessiere C., Quinqueton J. *Backtracking in Distributed Constraint Networks*. In Proceedings of the 13th, ECAI , Brighton, UK, 1998, pag. 219-223.

[4] L. Lamport K. M. Chandy. *'Distributed snapshots: Determining global states of distributed systems'*, TOCS, 3(1), 63–75, 1985.

[5] Yokoo M., E.H. Durfee, T. Ishida, K. Kuwabara - *Distributed constraint satisfaction for formalizing distributed problem solving.* In ICDCS, 614{621). June 1992.

[6] Yokoo M., E.H. Durfee, T. Ishida , K. Kuwabara - *The distributed constraint satisfaction problem : formalization and algorithms* . IEEE 10 (5), 1998 .

[7] Yokoo Makoto . Private communication, 2000.