Chapter 19

Openness in Complete Asynchronous Search

Once fully enslaved, no nation, state, city of this earth
ever afterwards resumes its liberty.
Walt Whitman

RESEARCHERS often see representative distributed applications as over-constrained problems that should undergo homogeneous relaxation (Yokoo 1993a; Hirayama & Yokoo 2000). This is a natural view of many negotiation problem. Our perception is that in the negotiations that humans like to perform, homogeneity of the relaxation cannot be ensured or checked. Due to this personal understanding of the matter, we had to develop an appropriate relaxation concept.

After describing the mentioned relaxation technique, we present a set of algorithms for dealing with openness in distributed computations: agent joining or leaving the computation, respectively agents that crash and recover. The relaxation technique that we describe is useful for modeling agents leaving a computation as a relaxation of their constraints.

19.1 Introduction

A major characteristic of distributed systems is their *openness*: the combination of agents making up the system is not built into the algorithms, and may not even be known when the algorithm is running, and agents may even be joining and leaving the system while an algorithm is active. Algorithms for DisCSP have focussed on asynchronous execution, but have not considered openness to a great extent.

In general, constraint satisfaction algorithms can be easily modified to allow addition of constraints during search. Similarly, it proves straightforward to allow new agents to introduce their variables and constraints in an existing DisCSP without need to restart an ongoing search process. This is similar to what has been noticed in general for distributed applications (Boichat 2001).

It is however considerably more complex to allow an agent to leave and remove its variables and constraints, since this may render feasible earlier options which have already been discarded. We show how a relaxation technique coupled with reordering allows us to reactive the right parts of the search space and thus makes it possible for agents to also leave a DisCSP while search for a solution is ongoing.

The next section enumerates some work on openness in complete algorithms. In the remaining sections we analyze the cases when agents can join the search, suffer a crash, and can leave with or without announcing.

19.2 Relaxation in Asynchronous Search

Agents that leave a process are typically involved in many structures maintained by other agents: outgoing-links, assignments, explicit and consistency nogoods. To cleanly remove agents, the structures maintained by protocol have to be updated. An elegant way of isolating an agent A_i is to reorder it to the position with the lowest priority. Then, no other agent owns assignments issued by A_i, and no agent maintains structures storing consistency nogoods at corresponding search levels. Additional mechanisms are used for eliminating outgoing-links, nogoods inferred from internal constraints of A_i, and consistency nogoods generated by A_i.

Several researchers have studied constraint removal in the framework of centralized consistency achievement and maintenance. (Prosser *et al.* 1992) proposes a natural and simple solution for AC3. (Bessière 1991) maintains more information with AC4, namely a separate justification for each eliminated value. This enables to recover some more information when a constraint predicate is removed. At the other extreme, (Neveu & Berlandier 1994) gives a technique which intelligently reuses much of the existing work without storing any additional data during computations of consistency. Open systems for problem solving are discussed in (Raja *et al.* 2000) and (Sycara 2001).

The Distributed Constraint Satisfaction Paradigm (DisCSPs) is defined in (Yokoo *et al.* 1998). Some well known extensions are Partial Distributed Constraint Satisfaction (Yokoo 1995) for approaching over-constrained problems, and Distributed Dynamic Constraint Satisfaction (Felfernig *et al.* 2001; Modi *et al.* 2001; Jung 2001) for modeling dynamic local problems.

19.3 Open System

For simplification, here we consider that all sent messages arrive in finite time to their destination, when the destination does not leave or suffer a crash. E.g. using TCP connections – when on the failure of such a connection, one should make sure that the destination agent eventually leaves or starts crash recovery procedures. Something similar to this is implemented in RETSINA (Sycara 2001).

19.3.1 Joining Asynchronous Search

It is very easy to allow new agents to join a search process since all existing work remains valid. When new agents want/accept to get involved in the computation, all the existing agents should receive, via an **involved** message, information on

- the initial priority, and
- the external variables of the new agents.

The agents that receives an **involved** message have to send their valid proposals, order, as well as their last generated valid labels, to the new agents. The simplest solution is to place new agents on positions following existing agents. Protocols supporting agent reordering (such as AWC, ABTR, MAS) allow then for reordering agents toward any other wished configuration.

Proposition 19.1 *If the first message sent by newly joining agents is sent after all previous agents have received the corresponding* **involved** *message, any running protocol that is an instantiation of MAS remains correct, complete and terminates.*

Proof. The protocol obtained by this combination behaves as an instance of MAS where t_h is higher than the time up to the involvement of the new agents, some messages are delayed for this period of time, and channels discard invalidated messages. □

When any agent answers **involved** messages with an acknowledgment toward the new agents, the last ones can straightforwardly detect the condition in the assumptions of Proposition 19.1. Alternatively, the algorithms could be adapted to ensure correct treatment of messages from unexpected agents. It has to be noted that termination and solution detection algorithms (Mattern 1987;

$$\begin{array}{lll} 9\text{: } \mathbf{A_2}/A^2 & \text{—}\mathbf{leaving}(A_2,||,A_1)\text{—}\rightarrow & \mathbf{A_1}/A^1/R^0/R^1 \\ 10\text{: } \mathbf{A_2}/A^2 & \text{—}\mathbf{leaving}(A_2,||,A_1)\text{—}\rightarrow & A_3/A^3 \\ 11\text{: } \mathbf{R^0}/A_1/A^1 & \text{–}\mathbf{reorder}(A_1,A_3)|0:1|\rightarrow & A_3/A^2 \end{array}$$

Figure 19.1: Example of ABTR when A_2 leaves the search. The default order is (A_1, A_2, A_3), $R^k \equiv A^k$, and initially $R^0 \equiv A_1$.

$$\begin{array}{lll} 9\text{: } \mathbf{A_2}/A^2/R^0 & \text{—}\mathbf{leaving}(A_2,||,A_1)\text{—}\rightarrow & \mathbf{A_1}/A^1/R^1 \\ 10\text{: } \mathbf{A_2}/A^2/R^0 & \text{—}\mathbf{leaving}(A_2,||,A_1)\text{—}\rightarrow & A_3/A^3 \\ 11\text{: } \mathbf{R^0}/A_1/A^1 & \text{–}\mathbf{reorder}(A_1,A_3)|0:1|\rightarrow & A_3/A^2 \end{array}$$

Figure 19.2: Example of ABTR when A_2 leaves the search. The default order is (A_1, A_2, A_3), $R^k \equiv A^k$, and initially $R^0 \equiv A_2$.

Silaghi *et al.* 2001i) that continuously monitor the system have to be adapted for taking into account the insertion of the new agents.

19.3.2 Leaving Asynchronous Search

To remove an agent cleanly, both its position and the effect of its constraints have to be removed.

19.3.2.1 Isolating an agent

Our first step towards removing an agent consists in isolating it from search. Given the reordering capabilities of MAS, it becomes easily possible to place any leaving agent, A_i, on the last position before removing outgoing-links and triggering the relaxation of the nogoods that A_i has generated.

We describe here the case where the agents agree on the convention: $A^j \equiv R^j$. Each reordering leader R^i stores the set L of agents that have left. Let us consider the case when an agent A^i leaves and N agents remain in the search process. Let $q = \min(i-1, N-2)$. When R^q knows that A_j^{q+1} leaves, (or also A_j^N when $q=N-2$), then R^q has to reorder A_j. If the *known order of* R^q specifies the sequence of agents: $A_{p_1}^1, ..., A_{p_{i-1}}^{i-1}, A_j^i, A_{p_{i+1}}^{i+1}, ..., A_{p_{N+1}}^{N+1}, ...$, then the agent R^q has to broadcast the message $\mathbf{reorder}(A_{p_1}^1, ..., A_{p_{i-1}}^{i-1}, A_{p_{i+1}}, ..., A_{p_{N+1}}, L)$ to all the agents $A_j, A_{p_{i+1}}, ..., A_{p_{N+1}}$. This order is tagged with a signature, as previously discussed. Any new proposed order should put the set L at the end of the sequence of agents.

19.3.2.2 Nogood management

I recall that a technique for saving nogoods in case of constraint relaxation consists of explaining inferences with references to constraints (CR). Details appear in Section 17.10.1.

19.3.2.3 Agents Announcing their Retreat

We see the departure of an agent A_i as a relaxation, namely the removal of the constraints of A_i. Obviously, the remaining agents need additionally to remove the links that they have towards A_i and also eliminate the corresponding data structures (assignments and set of labels generated by A_i for different variables). The removal of A^p or of R^p, can also be realized easily in MAS if the reordering leader R^{p-1} generates a **reorder** message which places that agent at the end of the search. However, when the agent delegated to act for R^0 withdraws, the remaining agents have to reach a consensus on a new delegation of R^0. Such a consensus can be easily obtained using the convention that the agent on the first position given the last order among the remaining agents, (e.g. A^1 if it did not leave), will act for R^0.

A new message, **leaving**, has to be used for signaling departure. When A_i leaves the search, it broadcasts **leaving**(A_i,order,R^0) to all other agents. The message takes as parameter the strongest order and the identity of R^0 known by the sender.

9: $\mathbf{A_1}/A^1/R^0/R^1$ —**leaving**$(A_1, ||, A_2)$→ $\mathbf{A_2}/A^2$
10: $\mathbf{A_1}/A^1/R^0/R^1$ —**leaving**$(A_1, ||, A_2)$→ A_3/A^3
11: $\mathbf{R^0}/A_2/A^1$ –**reorder**$(A_2, A_3)|0:1|$→ A_3/A^2

Figure 19.3: Example of ABTR when A_1 leaves the search. The default order is (A_1, A_2, A_3), $R^k \equiv A^k$, and initially $R^0 \equiv A_1$.

To enable the detection of the nogoods that depend on any leaving agent, any generated nogood has to be marked with the corresponding CRs. Whenever an agent, A_i, leaves the process, this information can be broadcasted to all the agents and the nogoods marked with CRs of A_i must be removed by everybody.

Figure 19.1 shows an example where A_2 leaves during the example in Figure 17.3. Normally, R^1 should undertake the task of reordering the agents, but since R^1 disappears when the number of agents reduces to 2 (Silaghi *et al.* 2001i), the task is undertaken by R^0. The Figures 19.2 and 19.3 show the case where the agent delegated to act for R^0 leaves. In both cases, the first remaining agent is delegated to act for R^0 and generates the new order. In case $R^0 \equiv A_3$, no new order needs to be generated, but A_1 is delegated to act for R^0.

While all the links, nogoods and assignments for leaving agents are removed when the corresponding **leaving** message is received, those agents are still stored in the sets L of each agent. Whenever the size of L grows over an agreed disturbing threshold, the owner agent can broadcast it to the other agents involved in the search, using a **left**(L) message. An agent stores the set L_i received via a **left** message from A_i. The agents contained in the intersection of all L_i received from all known agents absent from L can be safely removed from all stored L_i and from L. Knowing the total number of agents, the removed agents can be recognized in any received valid message.

19.3.2.4 Leaving without Announcing

Agents may leave without announcing. As long as nobody detects this departure, the search continues to use the nogoods inferred from the internal constraints of the disappeared agents. The dependencies continue to propagate and may lead to the replacement of valid nogoods with nogoods that depend on withdrawn agents. It is therefore important to detect such withdrawal as soon as possible.

When no time-out is established, one cannot ensure the achievement of any solution. If a time-out t_t is agreed-on, any agent that recovers after this time-out elapses will have to join as a new agent, and much information can be lost.

ABTR allows the agents to redelegate R^0 during a predefined delay $t_r + t_h$ from the beginning of the search. Moreover, the withdrawal of the agent acting for R^0 also leads to the redelegation of R^0. When the timeout is detected for an agent A_i, the system detecting it cannot be sure in asynchronous search whether A_i is or is not acting for R^0. This problem can be solved cleanly with a two rounds protocol, assuming that no other agent crashes during them. The first round consists in sending **leaving**$(A_i, \emptyset, \emptyset)$ messages to all remaining agents. When an agent A_j receives **leaving**$(A_i, \emptyset, \emptyset)$ without an order from an agent $A_k, k \neq i$, A_j will trigger the elimination of any message coming from A_i, but will not start acting for R^0, even if $A_j \equiv A^1$. Instead A_j sends a message **leaving-data**(A_i) to A_k attaching to it the strongest order known at A_j, and the estimated identity of R^0.

Proposition 19.2 *If A_k receives the answer* **leaving-data***(A_i) from all remaining agents, and if A_i is R^0 in the strongest received ordering, then the leaving agent is R^0.*

Proof. After any agent A_j receives a **leaving**(A_i), A_j will discard any new information generated by A_i. If A_i is R^0, it can no longer change it at A_j since any such change is discarded by A_j. □

After A_k receives **leaving-data**(A_i) from all remaining agents, if A_i is R^0 in the strongest received order, then A_k broadcasts **leaving**(A_i) with the strongest received order and identity for R^0.

procedure *elimination*($A : \{A_i, ...\}$) **do**
  broadcast **leaving**($A, \emptyset, \emptyset$);
  wait until receives all **leaving-data**(A,order,R^0);
    or timeout(t_t);
  if *no timeout* **then**
    broadcast **leaving**(A,strongest-order,R^0)
  else
    restart elimination procedure for agents that did not answer and for A

when A_k *detects time-out for* A_i **do**
  elimination($\{A_i\}$);

when A_j *receives* **leaving**$(A, \emptyset, \emptyset)$ *from* A_k **do**
  block A;
  answer with **leaving-data**(A,strongest-order,R^0);
  discard data from A_i or tagged C^{A_i}, $A_i \in \{A_i\}$;
  launch timer $2t_t$ for $\{A_k\} \cup A$;

when A_j *receives* **leaving**$(A$,*order*,$R^0)$ *from* A_k **do**
  block A;
  discard data from $\{A_i\}$ or tagged C^{A_i}, $A_i \in \{A_i\}$;
  if $A_j \equiv A^u$, *all* $A^{<u}$ *have left, and* $A_i \equiv R^0$ **then**
    $A_j \leftarrow R^0$;
  stop $R^u, u \geq N-1$, N-the nr. of remaining agents;
  while $A_i \in A, A_i \equiv A^v$ *and* $((v{=}N \wedge A_j \equiv R^{v-2})$ *or*
      $(v{<}N \wedge A_j \equiv R^{v-1}))$ **do**
    reorder A_i as A^{N+1};
  stop timer $2t_t$ for $\{A_k\} \cup A$;

when *timer* $2t_t$ *for* A **do**
  elimination(A);

Algorithm 32: Procedures for eliminating a set of agents A after time-out is detected by A_k.

If some other agent, A_u, does not answer to **leaving** messages, the removal procedure is interrupted after the corresponding time-out, A_k launches the protocol for announcing that both A_u and A_i have left (Algorithm 32). If A_k abandons himself without notice, any agent that has received from A_k an **leaving**(A_i) without an order and did not receive a **leaving**(A_i) with order, timeouts A_k after a delay $2t_t$ and starts the protocol for announcing the departure of both A_i and A_k. It can be easily proven that this technique leads to a system where:

Property 19.3 *Eventually, any departure suspected by one remaining agent will be suspected by all remaining agents.*

Proof. Messages broadcasted by correct agents are delivered by all remaining correct agents. If an agent fails after sending at least one announcement of a failure, the correct agent delivering such a message continues the elimination procedure which eventually succeeds. □

Unfortunately this technique leads to important losses when an agent cannot be reached for long time only due to network congestion. To reduce these problems, a longer timeout, T, can be established. Systems that are unreachable in acceptable time t, $t{<}T$, and that may have lost some messages, can be updated with a recovery mechanism similar to the one given in (Silaghi *et al.* 2001i).

19.4 Recovery From Hard Failures in MAS

Hard failures are failures where the agents have acted correctly until a certain moment when they disappear. The failure of agents can be usually detected with failure detectors (Schiper 1997). The failure detectors are not always safe. However, in this section we consider only the case when safe failure detectors are available. We present now an algorithm for recovering of agents in MAS from hard failures. The new messages used by this algorithm are **recovery** messages telling the receiver that the sender needs **recovery-data**. **recovery-data** messages are sent in answer to **recovery** messages and transport information that the receiver should know.

In the case of hard failures, the agents that do not crash can continue the inference work on all consistency levels. On recovery, a crashed agent A_i can easily reintegrate itself in the distributed computation. It has to request (by **recovery** messages) from any other agent A_k: the last aggregates, order and labels that A_i and A_k have built respectively sent at all levels $l \leq u$. u is the position known by A_k for A_i in the moment when it sends the recovery data. All this data is sent to A_i via **recovery-data** messages. A_k can clear all the links towards A_i and has to request again all add-links from A_i. During recovery, the recovering agent A_i must process only the incoming **recovery** and **recovery-data** messages. All other messages have to be stored and processed after the recovery stage.

After recovery data is obtained from all correct agents, the labels, order, aggregates and add-link messages generated by A_i are sent back to all interested agents that did not reported them. Counters that are not restored can be safely reinitialized to 0. Any message m received by A_k from A_i after the n-th recovery request of A_i is answered, is discarded if m was sent before the n-th recovery request of A_i. If we don't have FIFO channels, this condition has to be ensured (e.g. by attaching to all messages a counter of the crashes of the sender).

This protocol is a variant of a typical class of crash recovery protocols where all messages ever exchanged are sent to recovered agents (Boichat 2001). The peculiarity of this case is the fact that not all messages have to be exchanged, but only the strongest aggregates and orders, and the last labels.

Proposition 19.4 *After the recovery data is received, the recovered agents are fully integrated in search and their instantiation, view and counters are coherent with the other agents. The consistency labels they get are also coherent with the other correct agents, ensuring that the maximum degree of consistency allowed by DC is reached at quiescence if all crashed agents recover.*

Proof. It is only the last labels, the valid aggregates and the last add-links known by other agents that need to be known. All these are received in the recovery data. If an agent knows something newer than the others, the only case when it will not send that information is if it crashes before. But in this case the crashed agent forgets that information. All the received information is then broadcasted back so that everybody that is interested knows it.

Another agent A_j may be simultaneously in the recovery stage. A_j does not yet know all its own generated data but will broadcast it at the end of its recovery. If the last aggregates and labels sent by A_i before the crash were sent successfully to all other alive agents and were not meanwhile invalidated by other agents, then that data is received back for recovery and the situation is coherently reestablished.

However, data may have been successfully sent to only a subset of the interested agents before the crash. If any agent has received it correctly, that information will be received back on recovery if it was not invalidated. The last sent data is then resent, if valid, to all the other agents so that the views of the correct agents become coherent. □

Even if the search continues in a clean manner after crashed agents recover, the explicit and conflict-list nogoods that they lose due to the crash cannot be automatically recovered from others. The lost is important especially if the agent had a high priority. Therefore the agents need to make backups of their explicit and conflict-list nogoods so that they are enabled to recover as much as possible of the already spanned search space. If nogoods in backups contain newer or older aggregates than the received recovery data, the corresponding nogoods have to be invalidated. The validity of an aggregate in the nogoods in backups can be checked by testing whether their set of values is the same with the set in some valid aggregate received at recovery for the same variable.

If this holds, then the signature of the valid aggregate substitutes the one of the recovered one. If no valid aggregate can be found for some aggregate a, then a and the nogood that contains a are invalidated. The set of messages required in order to reconnect to broker, to get the addresses of the other agents and to request recovery data are obvious and are not presented here.

As argued in (Boichat 2001), the access to local permanent storage can be an expensive operation. Here, such access is not required for correctness, but only for efficiency. Trade-offs can be analyzed by each agent independently.

19.5 Main problems and research directions for enhancing openness

For existing complete protocols with polynomial space requirements, the losses that can incur when agents withdraw can vary from very little to almost everything. The worst cases are expected to occur either when a high priority agent withdraws, or when an agent involved in most stored nogoods withdraws. The last case is very likely towards the end of a search process for difficult problems.

The main advance that can be foreseen towards an improved response to openness in complete search protocols is the definition of some data structures to reduce the loss of information when certain patterns of events take place. Such a pattern is the withdrawal of only one agent. A possible strategy of A_j can consist of storing for each agent A_i, all the last valid labels where it was not involved in the inference process (labels and nogoods not tagged with $\mathrm{CR}(A_i)$). The space complexity would increase with a factor n (the number of agents). Also the local worst case computation cost of the agents can increase by the same factor. Namely, each time a new valid label arrives, it has to be combined with all existing labels for agents not involved (as shown by tags) in its inference.

19.6 Summary

We have analyzed the openness that can be expected currently from existing complete distributed asynchronous search protocols for DisCSPs. The discussion concentrates around instances of MAS. Few additional messages are sufficient to maintain the properties of the instances of MAS when new agents join the process. However, most messages and data structures have to be modified in order to allow agents to leave the search cleanly. The additional information is designed in a way that minimizes the loss of privacy without worsening the space complexity.

Following our personal understanding of the way people like to interact, we have developed a new and natural relaxation technique for DisCSPs. We have also presented a set of algorithms for dealing with openness in distributed computations: agent joining or leaving the computation, agents that crash and recover. The relaxation technique that we describe proves useful for modeling agents leaving a computation as a relaxation of their constraints. The reordering technique presented in previous chapters helps in the reinsertions of new agents with preferred priorities.