Chapter 4

Asynchronous Search

We are all born equal. But some are more equal than others.
Folklore

Now we introduce the initial formulation of the Distributed Constraint Satisfaction Problem and the first well known complete asynchronous algorithms for solving it. In the rest of the thesis we will study several algorithms by analyzing the transitions that lead from one to another. Each time we will discuss the different advantages and drawbacks.

After reading this chapter you are expected to know the basic notions and definitions that are used in the remaining part of the thesis.

4.1 Distributed CSP (DisCSP)

Distributed Constraint Satisfaction (DisCSP) is a powerful framework for modeling distributed combinatorial problems.

Definition 4.1 ((Yokoo *et al.* 1992)) *A* **DisCSP** *is given by a set of agents $A_1, ..., A_n$ where each agent A_i controls exactly one distinct variable x_i, and each agent knows all constraint predicates relevant to its variable. The agents want to agree on instantiations such that all the predicates are satisfied.*

Example 4.7 *Consider the problem discussed in the first chapter of this thesis. It can be modeled by:*

- *Two agents: A_1, A_2. A_1 stands for "Seller" and A_2 stands for "Buyer".*
- *A_1 enforces a CSP defined by:*
  - *a variable x_1 standing for "offer" and having as domain the set {(128 MB, SIPP)(256GB, DIMM)}.*
- *A_2 enforces a CSP defined by:*
  - *a variable x_2 standing for "needs" and having as domain the set {(128 MB, SIPP)(256GB, DIMM),...}.*
  - *a unary constraint: $\exists a, b$, $a \in D_a$, $b \in D_b$, where a and b are two technical parameters such that:*
    - *if(size(x_2) > 128 GB) $a + b = 14$*
    - *if(technology(x_2) is DIMM) $a + b < 20$*
- *a global constraint: $x_1 = x_2$.*

Remark 4.1 *The case with more variables in an agent can be easily obtained from it. Whatever protocol developed for the (Yokoo* et al. *1992) formulation of a DisCSP can be used in this new case by letting the same agent to act under different names for the different variables that it owns.*

4.2 Asynchronous Backtracking (ABT)

In asynchronous backtracking, the agents run concurrently and asynchronously. Each agent A_i instantiates its variable x_i and communicates the variable value to the relevant agents. (Yokoo *et al.* 1992) assumes FIFO channels. The channels are reliable, delivering messages within a finite delay, τ.

Definition 4.2 (Assignment) *An assignment for a variable x_i is a tuple $\langle x_i, v\rangle$ where v is a value from the domain of x_i.*

Since the channels deliver messages in FIFO order, the assignments are received in the order in which they are generated. A static (global) total order is imposed on agents and we assume that A_i has the i-th position in this order. If $i>j$ then A_i has a *lower priority* than A_j and A_j has a *higher priority* then A_i. A_j can impose first eventual preferences it has on its values

Rule 1 (Constraint-Evaluating-Agent) *Each constraint C is evaluated by the lowest priority agent whose variable is involved in C. It is denoted CEA(C).*

Each agent holds a list of *outgoing links* represented by a set of agent names. Links are associated with constraints. ABT assumes that every link is directed from the value sending agent to the constraint-evaluating-agent.

Definition 4.3 (Agent_View) *The* agent view *of an agent, A_i, is a set containing the* newest *assignments received by A_i for distinct variables.*

Based on their constraints, the agents perform inferences concerning the assignments in their *agent view*. By inference the agents generate new constraints called *nogoods*.

Definition 4.4 (Nogood) *A* nogood *has the form $\neg N$ where N is a set of assignments for distinct variables.*

The following types of messages are exchanged in ABT:

- **ok?** message transporting an assignment is sent to a constraint-evaluating-agent to ask whether a chosen value is acceptable.
- **nogood** message transporting a *nogood*. It is sent from the agent that infers a *nogood* $\neg N$, to the constraint-evaluating-agent for $\neg N$.
- **add-link** message announcing A_i that the sender A_j owns constraints involving x_i. A_i inserts A_j in its *outgoing links* and answers with an **ok?**.

The agents start by instantiating their variables concurrently and send **ok?** messages to announce their assignment to all agents with lower priority in their *outgoing links*. The agents answer to received messages according to the Algorithm 11 (Yokoo 2001).

4.2.1 Problems opened by ABT

ABT is the first asynchronous complete algorithm for solving DisCSPs. Besides his important qualities, it has also opened several research topics:

- Space Complexity. For its correctness (avoiding deadlocks), ABT requires that the agents store all the received nogoods. This cannot scale in the worst case since the number of possible nogoods is exponential in the number of agents/variables.
- Dynamic Reordering. For its termination, to coherently break ties and avoid infinite loops (Collin *et al.* 1991a), ABT requires a static order on its variables/agents.

```
when received (ok?,⟨x_j, d_j⟩) do
    add ⟨x_j, d_j⟩ to agent view;
    check_agent_view;
when received (nogood,A_j,¬N) do
    when ⟨x_k, d_k⟩, where x_k is not connected, is contained in ¬N
        send add-link to A_k;
        add ⟨x_k, d_k⟩ to agent view;
    put ¬N in nogood-list;
    old_value ← current_value;
    check_agent_view;
    when old_value = current_value
11.1    send (ok?,⟨x_i, current_value⟩) to A_j;
procedure check_agent_view do
    when agent view and current_value are not consistent
        if no value in D_i is consistent with agent view then
            backtrack;
        else
            select d ∈ D_i where agent view and d are consistent;
            current_value ← d;
            send (ok?,⟨x_i, d⟩) to lower priority agents in outgoing links;
procedure backtrack do
    nogoods ← {V | V = inconsistent subset of agent view};
    when an empty set is an element of nogoods
        broadcast to other agents that there is no solution;
        terminate this algorithm;
    for every V ∈ nogoods do
        select ⟨x_j, d_j⟩ where x_j has the lowest priority in V;
11.2    send (nogood,A_i,V) to A_j;
        remove ⟨x_j, d_j⟩ from agent view;
    check_agent_view;
```

Algorithm 11: *Procedures of A_i for receiving messages in ABT.*

- Shared Control. As mentioned in (Yokoo *et al.* 1992), it is true that DisCSPs solved with ABT can model problems where agents can control more than one variable. However, as we have noted in 1999, ABT cannot be used to allow a variable to be controlled by more than one agent. If the modeled problem allows several agents to control a variable, only one of them receives the control with ABT and it becomes a bottleneck if the other agents want to interfere.

- Opportunities for Interaction. As long as a proposal of an agent is not refused, that agent is idle. Much of the available computing power is lost in this way.

- Abstractions. ABT assumes that any agent is fully satisfied by its proposals. This impedes agents from making proposals based on abstractions of their local problem, as their real problem won't be satisfied.

- Security from Malicious Lies. In ABT, an agent that is announced about a conflict has no mean to detect whether that conflict is real.

- Openness. The configuration of the agents in ABT is static.

- Ranking of alternatives. With ABT there is no way to enforce preferences.

4.2.2 Add-link messages (DIBT and DDB)

In (Hamadi & Bessière 1998), the authors start from the idea that the **add-link** messages of ABT can be a source of problems in certain applications where dynamic management of communication is expensive (e.g. hardware). The topic has been further analyzed in (Bessière *et al.* 2001) that describes a complete algorithm solving this problem. The main technical idea is to discard on backtracking all the nogoods that in ABT require **add-link** messages upon their reception.

when received *(***ok?***,*($\langle x_j, d_j \rangle$, priority)*)* **do**
  add(($\langle x_j, d_j \rangle$, *priority*)) to *agent_view*;
  check_agent_view;
when received *(***nogood***,*A_j*,*$\neg N$*)* **do**
  when ($\langle x_k, d_k \rangle$, priority)*, where* x_k *is not connected, is contained in* $\neg N$
    add ($\langle x_k, d_k \rangle$, *priority*) to *agent_view*; add A_k to *outgoing links*;
  put $\neg N$ in *nogood-list*;
  old_value ← *current_value*; *old_priority* ← *current_priority*;
  check_agent_view;
  when (old_value = current_value) ∧ (old_priority = current_priority)
    for every *new agent* A_k *added to* outgoing links **do**
12.1 send (**ok?**,($\langle x_i$, *current_value*$\rangle$, *current_priority*)) to A_k;
procedure check_agent_view do
  when *high priority* agent_view*,* hpav*, and* current_value *are not consistent*
    if *no value in* D_i *is consistent with* hpav **then**
      backtrack;
    else
      select $d \in D_i$ where *hpav* and d are consistent and d minimizes
        the number of constraints violations with lower priority agents;
      current_value ← d;
      send (**ok?**,($\langle x_i, d \rangle$, *current_priority*)) to agents in *outgoing links*;
procedure backtrack do
  nogoods ← {V | V = inconsistent subset of *agent_view*};
  when *an empty set is an element of* nogoods
    broadcast to other agents that there is no solution;
    terminate this algorithm;
  when nogoods ∩ nogood_sent = ∅
    for every $V \in$ nogoods **do**
      add V to *nogood_sent*;
      for every ($\langle x_j, d_j \rangle, p_j$) *in* V **do**
        send (**nogood**,A_i,V) to A_j;
    $p_{max} \leftarrow max_{(\langle x_j, d_j \rangle, p_j) \in agent\_view}(p_j)$;
    current_priority$\leftarrow 1 + p_{max}$;
    select $d \in D_i$ where d minimizes the number of constraints violations
      with lower priority agents;
    current_value ← d;
    send (**ok?**,($\langle x_i, d \rangle$, *current_priority*)) to agents in *outgoing links*;

Algorithm 12: Procedures of A_i for receiving messages in AWC'.

4.3 Dynamic variable and value reordering in AWC

In AWC, each agent A_i has a priority defined by a local value $k(A_i)$ in conjunction with the initial position i. $k(A_i)$ is increased when a **new** nogood $\neg N$ is computed, such that A_i will precede all the agents generating assignments in N. The new priority is broadcasted to the neighboring agents. Excepting this priority, AWC behaves like the ABT which stores all nogoods. AWC is guaranteed to terminate since the number of possible nogoods is bounded. However the number of possible nogoods is exponential in the size of the problem. This algorithm has been published first time in (Yokoo 1995).

It is interesting to mention that a version of AWC that has appeared in (Yokoo 1993b) proposes that each agent A_i should stop from increasing $k(A_i)$ when $k(A_i)$ reaches a predefined threshold, rather than checking that the newly found nogood is new. While such a technique does not scale dynamically with the complexity of the problem at hand, it seems it has been abandoned. However, we show later that it is important from a theoretical point of view.

The procedures for receiving messages in AWC and in AWC' (AWC' is a new version I propose for AWC) are shown in Algorithm 12 (Yokoo 2001).

Remark 4.2 *In AWC', the line 12.1 is added here to ensure that agent views are correctly updated and remain consistent.*

AWC is one of the most efficient algorithms available. Even if its theoretical space requirements are combinatorial, the size of the DisCSPs that can be solved in our days is relatively small and AWC can often be used (Yokoo 2001).

An efficient local search algorithm for distributed settings is the Distributed Breakout. It was recently shown that for particular cases, e.g. trees, Distributed Breakout is complete (the lack of solution can be detected by measuring the number of cycles). Distributed Breakout is an algorithm with a particular (limited) type of asynchronism.

4.4 Summary

In this section we have learned the first framework described for Distributed Constraint Satisfaction. We have seen that its only deficiency of generality consists in not allowing shared control of variables.

ABT is the first asynchronous complete algorithm for solving DisCSPs. While this chapter enumerates several problems open by ABT, the subsequent chapters discuss ways of solving them.

Among the problems open by ABT, the enabling of dynamic ordering is solved for the first time by AWC. AWC remains one of the most efficient algorithms for solving DisCSPs. The problem opened by AWC is its theoretically combinatorial storage requirements. Later we describe some efforts to alleviate this problem.