Chapter 7

Asynchronous Maintaining Consistency

[I] heard two different opinions of this paper:
that it's trivial and that it's brilliant.
Jim Gray about (Lamport 1978)

Consistency maintenance is the most important technique in CSPs. Here I redefine and generalize it such that it can be introduced in asynchronous techniques for DisCSPs.

Shortly after the publication of Asynchronous Backtracking (ABT), described in Chapter 4, people have realized that the computation power of the agents is inefficiently lost. In ABT, many agents (and especially the first ones) are often idle while the remaining agents strive to prove the infeasibility of the received proposal.

Several authors have approached this problem. (Luo *et al.* 1992; 1993; Solotorevsky *et al.* 1996a) describe techniques where agents use their idle time to prepare new proposals in the event that the current proposal is refused. (Hamadi 1999b) proposes a technique where the agents can participate simultaneously in several distributed search processes on (disjoint subspaces of) the same problem. Nogoods can be shared and the order on agents can be different in distinct processes. In this chapter we propose another solution for enhancing the load balance by extending the interaction capabilities of the agents.

7.1 Achieving Distributed "Local" Consistency

The centralized local-consistency algorithms prune from the domain of variables, values that are locally inconsistent with the constraints. Their distributed counterparts (e.g. (Zhang & Mackworth 1991; Baudot & Deville 1997; Nguyen & Deville 1998; Monfroy & Rety 1999)) work by exchanging messages on value elimination. The restricted domains resulting from such a pruning are called *labels*. In this thesis we will only consider the local consistencies algorithms which work on labels for *individual* variables (e.g. arc-, bound-consistency). Such algorithms are denoted by $\mathcal{A}$. To be noted that the local consistency obtained for a problem can be less strong than the one obtained for the same problem represented with *total-constraints* (only one constraint per pair of variables).

Let P be a Distributed CSP with the agents $A_i, i{\in}\{1..n\}$. We denote by $C(P)$ the CSP defined by $\cup_{i\in\{1..n\}}\text{CSP}(A_i)$.[1] Let $\mathcal{A}$ be a centralized local consistency algorithm as just mentioned. We denote by DC($\mathcal{A}$) a distributed consistency algorithm that computes, by exchanging value eliminations, the same labels for P as $\mathcal{A}$ for $C(P)$. When DC($\mathcal{A}$) is run on P, we say that P becomes $\mathcal{A}$ consistent. Generic instances of DC($\mathcal{A}$) are denoted by DC. Typically with DC (Zhang & Mackworth 1991), the maximum number of generated messages is a^2vd and the maximum number of sequential messages is vd (v:number of variables, d:domain size, a:number of agents).

[1] The *union* of two CSPs, P_1 and P_2, is a CSP containing all the constraints and variables of P_1 and P_2.

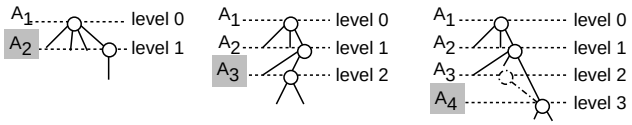


Figure 7.1: Distributed search trees in ABT: simultaneous views of distributed search seen by A_2, A_3, and A_4, respectively. Each arc corresponds to a proposal from A_i to A_j. Circles show the believed state of an agent. Dashed circle and line show known state that may have been changed.

7.2 Distributed search trees

In the centralized setting, the view of the search tree expanded by a consistency maintenance algorithm is unique. Each node n_i^k at depth k, corresponds to assigning to the variable x_k a value v_i from its label. Initially the label of each variable is set to its full domain. After each assignment $x_k = v_i$, a local consistency algorithm is launched which computes for the future variables the labels resulting from the assignment.

7.2.1 Simultaneous views of a system

In distributed search, each agent has its own perception of the distributed search tree. Its perception on this tree is determined by the proposals received from its predecessors. In Figure 7.1 is shown a simultaneous view of three agents. Only A_2 knows the fourth proposal of A_1. A_3 has not yet received the third proposal of A_2 consistent with the third proposal of A_1. However, A_4 knows that proposal of A_2. In Figure 7.1 we suppose that A_4 has not received anything valid from A_3 (e.g. after sending some nogood to A_3 which was not yet received). The term *level* in Figure 7.1 refers to the depth in the (distributed) search tree viewed by an agent.

We mention that (Luo *et al.* 1992; Hamadi 1999b) already mention similar distinctions within the view of an agent. They use it to optimize the amount of local effort deposed by agents for filtering domains.

7.2.2 Synchronous vs. Asynchronous Consistency Maintenance

A formal definition for consistency maintenance that encompasses the known variants as well as the most encountered explanations is given next.

Definition 7.1 (centralized consistency maintenance) *Consistency maintenance is a search technique where instantiations are interleaved with propagations of value removals. More exactly, prior to search and after each instantiation, one applies a technique for reducing the domains of the variables by exploiting the existing splits of the search space.*

Let P be a Distributed CSP with the agents $A_i, i{\in}\{1..n\}$, $\mathcal{A}$ be a centralized local consistency algorithm and DC($\mathcal{A}$) one of its distributed counterparts. Suppose that the instantiation order of the variables in $C(P)$ is determined by the order of the agents in P. In order to guarantee that with DC($\mathcal{A}$) one maintains for the variables of agents A_i of P the same labels, $\mathcal{L}$, than with $\mathcal{A}$ in $C(P)$, one simply imposes that:

1. A_i must have received the proposals of *all* its predecessors before launching DC($\mathcal{A}$),
2. A_i cannot make any proposal with values outside $\mathcal{L}$, computed by DC($\mathcal{A}$).

The resulting instantiation process is by construction synchronous. This is why DisCSP researchers have so far either synchronized their algorithms to introduce consistency maintenance, or they have used consistency only as a preprocessing step. Alternatively, I propose to handle consistency maintenance as a hierarchical task. I introduce now a new definition of consistency maintenance. First let us define consistency level.

Definition 7.2 (consistency level) *Consistency level k is a process that performs value eliminations by propagating domain splits at the depth k in a search tree, and eventually domain splits at lower depths, but no split from deeper depths.*

One can say that a consistency level k is active when the depth k in the search tree has defined a known split which has not yet been fully propagated according to the used notion of local consistency.

Definition 7.3 (consistency maintenance) *Consistency maintenance is a search process involving backtracking and consistency levels. It gives priority to consistency levels over backtracking and to lower consistency levels over deeper consistency levels.*

Remark 7.1 *It can be easily shown that the new definition introduced here for consistency maintenance generalizes Definition 7.1 in the sense that it correctly describes the behavior of centralized consistency maintenance. Initially only the consistency level* 0 *is active and has priority over backtracking, so that it is launched first. When it is stabilized, the backtracking can be launched, but after its first instantiation, consistency level* 1 *is activated and is given the control, and so on.*

Remark 7.2 *Definition 7.3 is more powerful than Definition 7.1, as it applies to asynchronous techniques.*

Based on this newly proposed definition, I can start introducing consistency maintenance in asynchronous backtracking. We see that A_i can then benefit from the value eliminations resulting from the proposals of *subsets* of its predecessors, as soon as available. More precisely, if A_i has received proposals from some of its k first predecessors, we say that it can benefit from value elimination (nogoods) of level k. Such nogoods are determined by instantiations of x_t, $t \leq k$ (known proposals), DC process at level k or inherited from DCs at previous levels along the same branch. A DC process of level k, DC_k, is a process which only takes into account the known proposals of the k first agents. The resulting labels are said to be of level k. When the labels are coherently assorted with their corresponding levels, and when they are coherently managed by agents as shown here, the instantiation decisions and DCs of levels k can then be performed asynchronously for different k without loosing the inference power of $\mathrm{DC}(\mathcal{A})$.

7.3 Maintaining Asynchronously Consistencies in ABT (DMAC-ABT)

Parts of the content of a message may become *invalid* due to newer available information. We require that messages arrive at destination in finite time after they are sent. The receiver can discard the invalid incoming information, or can reuse invalid nogoods with alternative semantics (e.g. as redundant constraints).

An important step used for enabling the consistency maintenance in asynchronous search is the achievement of the consistency with a technique based on nogoods. Some related work in the centralized CSP community is described in (Bessière 1991; Jussien *et al.* 1997; Jussien & Lhomme 1997; 1998; Jussien *et al.* 2000).

7.3.1 The DMAC-ABT Protocol

In addition to the messages of ABT, the agents in DMAC-ABT may exchange information about nogoods inferred by DCs. This is done using **propagate** messages as shown in Algorithm 19. Before making their first proposal as in ABT, the agents start with a call to **maintain_consistency**(0).

Definition 7.4 (Consistency nogood) *A consistency* nogood *for a level k and a variable x has the form $V \rightarrow (x \in l_x^k)$ or $V \rightarrow \neg(x \in s \backslash l_x^k)$. V is a set of assignments. Any assignment in V must have been proposed by A_k or its predecessors. l_x^k is a label, $l_x^k \neq \emptyset$.*

```
when received(propagate,A_j,k,c^k_{x_v}(j),V→(x_v∉l)) do
19.1  when have higher tag c^k_{x_v}(j,i)≥c^k_{x_v}(j) then return;
      c^k_{x_v}(j,i) ← c^k_{x_v}(j);
      when any ⟨x,d,c⟩ in V is invalid (old c) then return;
      when ⟨x_u,d_u,c_u⟩, where x_u is not connected, is contained in V
          send add-link to A_u;
          add ⟨x_u,d_u,c_u⟩ to agent view;
19.2  add other new assignments in V to agent view;
      eliminate invalidated nogoods;
      cn^k_{x_v}(i,j) ← {V→(x_v∉l)};
      maintain_consistency(minimal level that is modified);
      check_agent_view; //only satisfies consistency nogoods of levels t, t≤cL_i;
procedure maintain_consistency(minT) do
      if (minT > cL_i) then
          return; //cL_i is the current inconsistent level (initially i+1);
19.3  for (t←minT; t≤i; t++) do
19.4      new-cn ← consistency nogood for x_i after local consistency on P_i(t);
          when (domain wipe out by computing explicit nogoods nogoods)
              for every V ∈ nogoods do
                  select ⟨x_j,d_j,c_{x_j}⟩ where x_j has the lowest priority in V;
                  send (nogood,A_i,V) to A_j;
                  eliminate invalidated explicit nogoods;
                  remove ⟨x_j,d_j,c_{x_j}⟩ from agent view;
              cL_i ←t;
              break;
          when new-cn shrinks label of x_i (obtained from ∪_{k≤t} cn^k_{x_i}(i,i))
19.5          cn^t_{x_i}(i,i) ← new-cn;
              C^t_{x_i}++;
              send (propagate,A_i,k,C^t_{x_i},new-cn) to interested agents A_j, j≥t;
```

Algorithm 19: *Procedure of A_i for receiving* **propagate** *messages in DMAC-ABT.*

Example 7.8 **propagate***(A_5,3,312,($\langle x_1,2,124\rangle \wedge \langle x_2,4,756\rangle$) → ($x_5 \notin \{2,5,6,23\}$)) denotes a message transporting a consistency nogood generated by A_5. This is a consistency nogood for the level 3 and for the variable x_5. It is the 312th* **propagate** *message sent by A_5 for x_5 at level 3.*

The semantic is that x_5 cannot be instantiated to values from the set $\{2,5,6,23\}$, as long as both assignments: $\langle x_1,2,124\rangle$, and $\langle x_2,4,756\rangle$, are valid.

To be noticed that the nogood ($\langle x_1,2,124\rangle \wedge \langle x_2,4,756\rangle$) → ($x_5 \notin \{2,5,6,23\}$) can be a consistency nogood of level 2, and its power is stronger if it is considered as such. Also, if invalidated, this nogood can be transformed in the constraint $\neg((x_1{=}2) \wedge (x_2{=}4) \wedge (x_5 \in \{2,5,6,23\}))$.

Definition 7.5 *An agent is interested in consistency nogoods for a variable x if it enforces some constraints on x.*

Remark 7.3 *In the previous definition one can decide to also consider backtracking nogoods as constraints. In our implementations we have restricted ourselves to consider that an agent is interested only in variables for which it enforces initial constraints.*

The **propagate** messages for a level k are sent to all interested agents $A_i, i \geq k$. They take as parameters the reference k of a level and a consistency nogood. Each consistency nogood for a variable x_i and a level k is **tagged** with the value of a counter $C^k_{x_i}$ maintained by the sender. The agents A_i use the most recent proposals of the agents $A_j, j \leq k$ when they compute DC consistent

when received *(***ok?**,$\langle x_j, d_j, , c_{x_j}\rangle$*)* **do**
  if(old c_{x_j}) **then** return;
20.1 add(x_j,d_j,c_{x_j}) to *agent view*; eliminate invalidated nogoods;
  maintain_consistency(j);
  check_agent_view; //only satisfies consistency nogoods of levels t, t<cL$_i$;
procedure check_agent_view do
  when agent view *and* current_value *are not consistent //cf. nogoods of levels t, t<cL$_i$*
    if *no value in D_i is consistent with* agent view **then**
      backtrack;
    else
      select $d \in D_i$ where *agent view* and d are consistent;
      current_value ← d;
      $C^i_{x_i}$++;
      maintain_consistency(i);
      send (**ok?**,$\langle x_i, d, C^i_{x_i}\rangle$) to lower priority agents in *outgoing links*;

Algorithm 20: *Procedures of A_i for receiving* **ok?** *messages in DMAC-ABT.*

labels of level k. A_i may receive valid consistency nogoods of level k with assignments for the variables *vars*, *vars* not in evars(A_i). A_i must then send **add-link** messages to all agents $A_{k'}, k' \leq k$ not yet linked to A_i and owning variables in *vars*. In order to achieve consistencies asynchronously, besides the structures of ABT, implementations can maintain at any agent A_i, for any level k, $k \leq i$:

- The set, V^i_k, of the newest valid assignments proposed by agents $A_j, j \leq k$, for each interesting variable.
- For each variable x, $x \in$vars(A_i), for each agent $A_j, j \geq k$, the last consistency nogood (with highest tag) sent by A_j for level k, denoted $cn^k_x(i,j)$. $cn^k_x(i,j)$ is stored only as long as it is valid. It has the form $V^k_{j,x} \rightarrow (x \in s^k_{j,x})$.

By union of two CSPs (constraints) I denote the CSP defined by the union of the variables and constraints of the parameters. $\mathbf{NV_i(V^i_k)}$ is the constraint of coherence of A_i with the view V^i_k. Let $cn^k_x(i,.)$ be $(\cup^{t\leq k}_{t,j} V^t_{j,x}) \rightarrow (x \in \cap^{t\leq k}_{t,j} s^t_{j,x})$. $P_i(k) := \text{CSP}(A_i) \cup (\cup_x cn^k_x(i,.)) \cup \text{NV}_i(V^i_k) \cup \text{CL}^i_k$. $C^k_{x_i}$ is incremented on each modification of $cn^k_{x_i}(i,i)$ (line 19.5).

On each modification of $P_i(k)$, $cn^k_{x_i}(i,i)$ is recomputed by inference (e.g. using local consistency techniques at line 19.4) for the problem $P_i(k)$. $cn^k_{x_i}(i,i)$ is initialized as an empty constraint set. CL^i_k is the set of all nogoods known by A_i and having the form $V \rightarrow C$ where $V \subseteq V^i_k$ and C is a constraint over variables in vars(A_i). $cn^k_{x_i}(i,i)$ is stored and sent to other agents by **propagate** messages iff its label shrinks and either CSP(A_i) or CL^i_k was used for its logical inference from $P_i(k)$. This is also the moment when $C^k_{x_i}$ is incremented. The procedure for receiving **propagate** messages is given in Algorithm 19.

We now prove the correctness, completeness and termination properties of DMAC-ABT. We only use DC techniques that terminate (e.g. (Zhang & Mackworth 1991; Baudot & Deville 1997)).

Definition 7.6 (Quiescence in DMAC-ABT) *By quiescence of a group of agents we mean that none of them will receive or generate any valid nogoods, new valid assignments,* **propagate** *or* **add-link** *messages.*

Property 7.1 *In finite time t^i either a solution or failure is detected, or all the agents $A_j, 0 \leq j \leq i$ reach quiescence in a state where they are not refused a proposal satisfying* ECSP(A_j)$\cup$NV$_j$(view(A_j)).

Proposition 7.2 *DMAC-ABT is correct, complete and terminates.*

The proof is given in Section 7.7. It remains to show the properties of the labels computed by DMAC-ABT at each level of the distributed search tree. If the agents, using DMAC-ABT, store all the valid consistency nogoods they receive, then DCs in DMAC-ABT converge and compute a local consistent global problem at each level (each pair initial_constraint-variable_label is checked by some agent). If on the contrary, the agents do not store all the valid consistency nogoods they receive but discard some of them after inferring the corresponding $cn_x^k(i,i)$, then some valid bounds or value eliminations can be lost when a $cn_x^k(i,i)$ is invalidated. Different labels are then obtained in different agents for the same variable. These differences have as result that the DC at the given level of DMAC-ABT can stop before the global problem is DC consistent at that level.

Among the consistency nogoods that an agent computes itself at level k from its constraints, $cn_x^k(i,i)$, let it store only the last one for each variable and only as long as it is valid. Let A_i also store only the last (with highest tag) consistency nogood, $cn_x^k(i,j)$, sent to it for each variable $x \in \text{vars}(A_i)$ at each level k from any agent A_j. $cn_x^k(i,j)$ is also stored only as long as it is valid. Each agent stores the highest tag $c_x^k(j)$ for each variable x, level k and agent A_j that sends labels for x. Then:

Proposition 7.3 *DC($\mathcal{A}$) labels computed at quiescence at any level using* **propagate** *messages are equivalent to $\mathcal{A}$ labels when computed in a centralized manner on a processor. This is true whenever all the agents reveal consistency nogoods for all minimal labels, l_x^k, which they can compute and when CL_k^i are not used.*

Proof. In each sent **propagate** message, the consistency nogood for each variable is the same as the one maintained by the sender. By checking $c_{x_v}^k(j)$ at line 19.1, the stored consistency nogoods are coherent and are invalidated only when newer assignments are received (event that is coherent) at lines 17.1,19.2,20.1. Any assignment invalid in one agent will eventually become invalid for any agent. Therefore, any such nogood is discarded at any agent, iff it is also discarded at its sender. The labels known at different agents, being computed from the same consistency nogoods, are therefore identical and the distributed consistency will not stop at any level before the global problem is local consistent in each agent. □

Since consistency nogoods are not discarded when nogoods are sent to agents generating their assignments, asynchronism is ensured by temporarily disregarding those consistency nogoods. In Algorithm 20 we only satisfy consistency nogoods at levels lower than the current inconsistent level, cL_i. Alternatively, such consistency nogoods could be discarded but then, to ensure coherence of labels, agents receiving any nogood should always broadcast assignments with new tags and many nogoods would be unnecessarily invalidated.

ABT may deal with problems that require privacy of domains. For such problems, agents may refuse to reveal labels for some variables. The initial labels at level 0 are given by the initial domains. The strength of the maintained consistency is function of how many such private domains are involved in the problem. The versions of ABT dealing with privacy on constraints do not have this problem.

Proposition 7.4 *The space an agent requires with DMAC-ABT for ensuring maintenance of the highest degree of consistency achievable with DC is $O(v^2(v+d))$. With bound consistency, the required space is $O(v^3)$.*

The proof is given in Section 7.7.

Remark 7.4 *For efficiency reasons, agents may decide to store much more nogoods than the required ones. Nogoods can be stored as redundant constraints.*

Remark 7.5 (DMAC_1-ABT) *Agents may store one nogood per value instead of one nogood per label (for each level of consistency, variable and source). This technique leads to the same degree of consistency but saves more work on changes and is important since it resembles some existing techniques in centralized settings.*

The technique conforming with the previous remark will be referred to as DMAC_1.

```
    procedure maintain_consistency(minT) do
        if (minT > cL_i) then return;
21.1    for (t←minT; t≤i; t++) do
            new-cns ← consistency nogoods for x_k∈vars(A_i)
                    after local consistency on P_i(t);
            when (domain wipe out by computing explicit nogoods nogoods)
                for every V ∈ nogoods do
                    select ⟨x_j, d_j, c_{x_j}⟩ where x_j has the lowest priority in V;
21.2                send (nogood,A_i,V) to A_j;
                    eliminate invalidated explicit nogoods;
                    cL_i ←t;
                    remove ⟨x_j, d_j, c_{x_j}⟩ from agent view;
                break;
            for every new-c_{x_u} (consistency nogood for x_u) ∈ new-cns do
                when new-c_{x_u} shrinks label of x_u (obtained from ∪_{w,k≤t} cn^k_{x_u}(i,w))
                    cn^t_{x_u}(i,i) ← new-c_{x_u};
                    c^t_{x_u}(i)++;
                    send (propagate,A_i,t,c^t_{x_u},new-cn) to interested agents A_j, j≥t;
```

Algorithm 21: *Procedure of A_i for receiving* **propagate** *messages in DMAC-ABT1.*

Remark 7.6 (DMAC^k-ABT) *Another possibility for ensuring coherence and maximal strength of consistency with more nogoods is to let each agent store the last k valid consistency-nogoods generated by each agent for each variable, at each level, for some k.*

When in DMAC^k-ABT the mentioned consistency-nogoods are stored separately for each value, the obtained protocol can be denoted DMAC$^k_1$-ABT.

7.3.2 Using all available valid nogoods for consistency inferences (DMAC-ABT1)

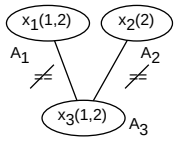
In Algorithm 19, an agent A_i only sends consistency nogoods for the variable x_i. However, when the local consistency is computed for $P_i(k)$, new labels are also computed for other variables known by A_i.

If in $P_i(k)$ we only use consistency nogoods and initial constraints, the final result of the consistency maintenance is coherent in the sense that at quiescence at any given level, each agent ends knowing the same label for each variable. Namely the new label obtained by A_i for some variable x_u will be computed and sent by A_u after receiving the other labels in consistency nogoods and instantiations that A_i knows and are related to x_u.

Agents can use in their $P_i(k)$ valid nogoods that they have received by **nogood** messages or old and invalidated consistency nogoods stored as redundant constraints. In this last case the labels obtained with Algorithm 19 are no longer minimal since an agent A_u does not know all constraints that can be used by A_i locally for computing its version of the label of x_u at level k.

In Algorithm 21 we present a version of DMAC-ABT that we call DMAC-ABT1. In DMAC-ABT1, A_i can send consistency nogoods for all variables found in CSP(A_i). The space complexity for storing the last tags for the consistency nogoods at all levels and coming from all other agents is now $O(v^3)$ and for DMAC-ABT1 the space complexity is $O(v^3(v+d))$. However, the power of DCs is increased since it accommodates any available nogood. The number of sequential messages is also reduced since there is no need to wait for A_u to receive the label of x_i before reducing the label of x_u. Rather A_i propagates itself the label of x_u.

Proposition 7.5 *The minimum space an agent needs with DMAC-ABT1 for ensuring maintenance of the highest degree of consistency achievable at each level with DC is $O(v^3(v+d))$. With*



1: A_1 —— **ok?**$\langle x_1, 1, 1\rangle$ ——→ A_3
2: A_2 — **propagate**$(A_2,0,1,x_3 \notin \{2\})$ → A_1
3: A_2 — **propagate**$(A_2,0,1,x_3 \notin \{2\})$ → A_3
4: A_2 —— **ok?**$\langle x_2, 2, 1\rangle$ ——→ A_3
5: A_1 — **propagate**$(A_1,0,1,x_1 \notin \{1\})$ → A_3
6: A_1 —— **ok?**$\langle x_1, 2, 2\rangle$ ——→ A_3
7: A_3 — **propagate**$(A_3,0,1,x_1 \notin \{1\})$ → A_1
8: A_3 —— **nogood**$\neg(\langle x_1, 1, 1\rangle \rightarrow$ A_1

Figure 7.2: Simplified example for DMAC-ABT1. Function of the exact timing of the network, some of these messages are no longer generated. Only 2 messages are sequential (half round-trips). ABT needs 4 sequential messages (half round-trips) for the same example (see (Yokoo 2001)).

bound consistency, the required space is $O(v^4)$.

The proof is given in Section 7.7.

7.4 Example

In Figure 7.2 we show a trace of DMAC-ABT1 for the example described in (Yokoo 2001). Before making its proposal, A_2 sends **propagate** messages to announce the consistency nogood $x_3 \notin \{2\}$ of level 0, tagged with $c^0_{x_3}(2) = 1$. These **propagate** messages are sent both to A_1 and A_3. A_1 answers by an **ok?** proposing a new instantiation.

A_3 (and A_1 when the domain of x_3 is public) compute both the consistency nogood $x_1 \notin \{1\}$ at level 0. A_3 computes an explicit nogood from consistency at level 1 and sends it to A_1. This nogood is invalid since A_1 has already changed its instantiation (and a small modification of DMAC-ABT1, for simplicity not given here, can avoid sending it). Then solution and quiescence are reached. The longest sequence of messages valid at their receivers (length 2) consists in messages 2,6. The worst case timing (slow communication channel from A_2 to A_1 or privacy for the domain of x_3) gives the longest sequence 3,7,6 (5 would not be generated). The fact that ABT (as well as any synchronous algorithm) would require at least 4 sequential messages illustrates the parallelism offered by asynchronous consistency maintenance.

7.5 Optimizations for nogoods detected by consistency

The domain wipe-outs generated due to consistency nogoods may be detected by several agents simultaneously. This happens when inference from consistency nogoods generated concurrently by two agents leads to a domain wipe-out represented as an explicit nogood. We call the result of this inference **consistency explicit nogood**.

Definition 7.7 *A consistency explicit nogood is an explicit nogood detected via a domain wipe-out generated by combining consistency nogood.*

Versions can be designed where

1. **Rule 3** *All consistency nogoods are sent to all agents generating the corresponding consistency level.*
2. **Rule 4** *The algorithm design can be such that any consistency nogoods of level i has in its premise an aggregate generated by the agent* A_i.

This is the case for the version described here when A_u stores consistency nogoods for $k{=}u$ and for all variables.

Remark 7.7 (DMAC') *In such versions, where both aforementioned rules are enforced, a consistency explicit nogood for level i does not have to be sent by* **nogood** *messages since it is guaranteed that the target agent,* A_i *detects the nogood itself.*

Remark 7.8 (DMAC") *If only the condition of having premises generated by A_i in consistency nogoods at level i (Rule 4) is not respected, the target may not be A_i, but an improvement is still possible by assigning only to A_i the task of sending corresponding consistency explicit nogoods by messages.*

Remark 7.9 (DMAC"') *However, even if none of the conditions of algorithm design just mentioned, DMAC', are fulfilled, a convention can be established for sending by a* **nogood** *message from an agent A_k a consistency explicit nogood detected at level i for a domain wipe-out on the variable v and having as target an agent A_j. The convention requires A_k to send such a message only if none of the agents $A_l, l \in [i,k)$ is interested in v.*

After computing a consistency explicit nogood from consistency nogoods at level i, an agent A_k does not have to wait for this nogood to be invalidated. A_k is allowed to use the otherwise idle time for a search where the consistency nogoods of level i and upper are not used.

7.6 Hybridizing DMAC-ABT and SyncDFSTBranching

DMAC, can be transferred to a schema built on the SyncDFSTBranching algorithm of (Collin *et al.* 1991a). It is sufficient to initially order the agent according to DFS. Several centralized and distributed algorithms are available for this task (Makki & Havas 1996; Tel 1999; Tsin 2002). The levels on distinct branches of the DFS hierarchy of agents need not be interleaved. Theoretically, there should be no essential modification in terms of messages since the problems on the distinct branches are disjoint. The only difference is in the structures that agents maintain, where levels from other branches need not be checked.

7.7 Proof

Property 7.1 *In finite time t^i either a solution or failure is detected, or all the agents $\mathbf{A}^j, 0 \leq j \leq i$ reach quiescence in a state where they are not refused a proposal satisfying* $\mathrm{ECSP}(\mathbf{A}^j) \cup \mathrm{NV_j}(\mathrm{view}(\mathbf{A}^j))$.

Proof. The proof is by induction on i. Let this be true for the agents $\mathbf{A}^j, j<i$. Let τ be the maximum time taken by a message. After $t^{i-1}+\tau$, $\mathbf{A}^i$ no longer receives **ok?** messages. $\mathbf{A}^i$ receives the last valid **ok?** message at time $t^i_o \leq t^{i-1}+\tau$. $\exists t^i_v$, $t^{i-1}+\tau \geq t^i_v$ such that after t^i_v, view($\mathbf{A}^i$) and all $V^u_k, k<i$ of any agent A_u are no longer modified.
The set of disabled tuples in $\mathrm{CL}^u_k, k<i$ can contain only a bounded number of elements for each agent A_u and they cannot be invalidated after t^i_o. $\mathrm{CL}^u_k, k<i$ cannot be invalidated after t^i_v. Since DCs were assumed to terminate, they terminate after each modification of a CL^u_k. Since the number of such modifications that can generate a new consistency nogood after t^i_v is bounded, after a finite time no consistency nogood is received any longer by $\mathbf{A}^i$ for levels $k<i$.
Since the domains are finite, $\mathbf{A}^i$ can make only a finite number of different proposals satisfying view($\mathbf{A}^i$). Once any of them is sent, the total number of consistency nogoods that can be received before the proposal is modified is finite (this results by induction to levels $k \leq i$ of the reasoning for $k<i$ in the previous paragraph since after $v\tau$, $\mathbf{A}^i$ can receive only valid nogoods: valid explicit nogoods trigger the modification of the instantiation of $\mathbf{A}^i$ so that they can arrive only in finite time; if valid explicit nogoods are not received and no instantiation modification is done in finite time, no **ok?** is sent any longer by $\mathbf{A}^i$, and the number of valid consistency nogoods at level i is limited as in the previous paragraph).
Only one valid explicit nogood can be received for a proposal since the proposal is immediately changed on such an event. Invalid nogoods can be received only within $v\tau$ time delay after a proposal is made. Therefore, there is a finite number of nogoods that can be received by $\mathbf{A}^i$ for any of its proposals made after t^i_v (and after t^i_o).

1. If one of the proposals is not refused by incoming nogoods, and since the number of received nogoods is finite, the induction step is correct.

2. If all proposals that $\mathbf{A}^i$ can make after t^i_o are refused or if it cannot find any proposal, $\mathbf{A}^i$ has to send according to rules inherited from ABT a valid explicit nogood $\neg N$ to somebody. $\neg N$ is valid since all the assignments of $\mathbf{A}^k, k < i$ were received at $\mathbf{A}^i$ before t^i_o.

2.a) If N is empty, failure is detected and the induction step is proved.

2.b) Otherwise $\neg N$ is sent to a predecessor $\mathbf{A}^j, j<i$. Since $\neg N$ is valid, the proposal of $\mathbf{A}^j$ is refused, but due to the premise of the inference step, $\mathbf{A}^j$ either

2.b.i) finds an assignment and sends **ok?** messages or

2.b.ii) announces failure by computing an empty nogood (induction proven).

In the case (i), since $\neg N$ was generated by $\mathbf{A}^i$, $\mathbf{A}^i$ is interested in all its variables, and it will be announced by $\mathbf{A}^j$ of the modification by an **ok?** messages.

Case 2.b.i contradicts the assumption that the last **ok?** message was received by $\mathbf{A}^i$ at time t_o^i and the induction step is therefore proved for all alternative cases. The property can be attributed to an empty set of agents and it is therefore proved by induction for all agents. □

Proposition 7.2 *DMAC-ABT is correct, complete and terminates.*

Proof. **Completeness:** All the nogoods are generated by logical inference from existing constraints. Therefore, if a solution exists, no empty nogood can be generated.
No infinite loop: The result follows from Property 7.1.
Correctness: All valid proposals are sent to all interested agents and stored there. At quiescence all the agents know the valid interesting assignments of all predecessors. If quiescence is reached without detecting an empty nogood, then all the agents agree with their predecessors and their intersection is nonempty and correct. □

Proposition 7.4 *The minimum space an agent needs with DMAC-ABT for ensuring maintenance of the highest degree of consistency achievable with DC is $O(v^2(v+d))$. With bound consistency, the required space is $O(v^3)$.*

Proof. d-maximal domain size;v-number of variables. The space required for storing all valid assignments is $\mathrm{O}(v)$ for values and $\mathrm{O}(v)$ for the corresponding counters. The agents need to maintain at most v levels, each of them dealing with maximum v variables, for each of them having at most 1 last consistency nogood. Each consistency nogood refers at most v assignments in premise and stores at most d values in label. The stack of labels requires therefore $\mathrm{O}(v^2(v+d))$. The space required by the algorithm for solving the local problem depends on the corresponding technique (e.g. chronological backtracking requires $\mathrm{O}(v)$). The stored explicit nogoods require $\mathrm{O}(dv)$ as mentioned in Property 6.1. In DMAC-ABT are also stored $\mathrm{O}(v^2)$ tags for consistency nogoods. □

Proposition 7.5 *The minimum space an agent needs with DMAC-ABT1 for ensuring maintenance of the highest degree of consistency achievable with DC is $O(v^3(v+d))$. With bound consistency, the required space is $O(v^4)$.*

Proof. The agents need to maintain at most v levels, each of them dealing with maximum v variables, for each of them having at most v last consistency nogoods. Each consistency nogood refers at most v assignments in premise and stores at most d values in label. The stack of labels requires therefore $\mathrm{O}(v^3(v+d))$. DMAC-ABT1 also stores $\mathrm{O}(v^3)$ tags for consistency nogoods. The other structures are identical as for DMAC-ABT. □

7.8 Summary

In this chapter we have learned how agents can be offered the oportunity to cooperate with inferences for the reduction of the search space. This process takes place in parallel and concurrently with the distributed search.

The main introduced ideas are:

- A new definition for consistency maintenance that generalizes existing approaches and applies to asynchronous search.
- A consistency maintenance technique based on nogoods.

- An integrated framework for nogood-based consistency maintenance and nogood-based backtracking where nogoods can be exchanged among techniques leading to an algorithm that is strictly more powerful than existing centralized and distributed techniques.
- A technique for guaranteeing wished consistency strength using a polynomially bounded number of nogoods.
- A set of possible optimizations, DMAC', DMAC", DMAC"', in announcing nogoods detected by domain wipe-outs.
- Another optimization, DMAC_1-ABT, is based on storing consistency nogoods for individual values.
- DMAC^k-ABT, consists in ensuring the strongest achievable consistency notion by storing the last k valid consistency nogoods generated by each agent for each variable and at each level.
- The different optimizations can be combined.