Chapter 12

Multiply Asynchronous Search

MY FAITH IN DOCTORS
My faith in doctors is immense.
Just one thing spoils it:
their pretense
of authorized omniscience.
FH

In previous chapters we have independently presented several techniques that can be added on top of polynomial space asynchronous backtracking:

- asynchronous definition of value abstractions, namely aggregations (Section 9.2).
- asynchronous maintenance of consistency at all depths in the search tree (Chapter 7).
- asynchronous dynamic reordering (Section 10.3).

In this chapter we show that these techniques can be combined in a quite straightforward way. We also present proofs showing that the obtained general protocol maintains correctness, completeness and termination properties. Due to its high degree of asynchronous interaction, the general protocol is called Multiply Asynchronous Search (MAS).

12.1 Multiply Asynchronous Search (MAS)

We assume that a message does not necessarily need to head directly to its target agent. Its content can travel indirectly to the target via several agents. However, we require the content to arrive at destination in finite time. Parts of the content of a message may become *invalid* due to newer available information. The receiver can discard the invalid incoming information, or can reuse invalid nogoods with alternative semantics (e.g. as redundant constraints). However, when the channel (intermediary agents) holds information that it knows invalid, it is allowed to discard that information.

12.1.1 Maintaining Consistency in AAS (DMAC)

Now we describe how the consistency maintenance technique previously introduced in ABT can be adapted for AAS. The **ok?**, **nogood** and **add-link** messages are as in AAS. In addition, the agents may exchange information about nogoods inferred by DCs. This is done using **propagate** messages.

Definition 12.1 *A* **consistency nogood** *for a level k and a variable x has the form $V \rightarrow (x \in l_x^k)$ or $V \rightarrow \neg(x \in s \setminus l_x^k)$. V is an aggregate and may contain for x an assignment $\langle x, s, h \rangle$, $l_x^k \subset s$. Any assignment in V must have been proposed by predecessors of A^k. l_x^k is a label, $l_x^k \neq \emptyset$.*

The **propagate** messages take as parameters a list of consistency nogoods, the reference k of a level and a counter of the **propagate** messages sent by the sender. They are sent to all interested agents $A^i, i \geq k$. The nogoods are meant to allow the agents A^i to compute DC consistent labels on the problem obtained by integrating the most recent proposals of the agents $A^j, j<k$. A_i may receive valid consistency nogoods of level k with the variables *vars*, *vars* not in vars(A_i). A_i must send **add-link** messages to all agents $A^{k'}, k'<k$ not yet linked to A_i for all *vars*. In order to achieve consistencies asynchronously, besides the structures of AAS, implementations can maintain at any agent A_i^u, for any level k, $k \leq u$:

- The aggregate, V_k^i, of the newest valid assignments proposed by agents $A^j, j<k$, for each interesting variable.
- For each variable x, $x \in \text{vars}(A_i)$, for each agent $A_j^t, t \geq k$, the last consistency nogood sent by A_j for level k, denoted $cn_x^k(i,j)$, if valid. It has the form $V_{j,x}^k \rightarrow (x \in s_{j,x}^k)$.

$cn_x^k(i,i)$ is recomputed by inference (e.g. using local consistency techniques) for each variable x for the problem $P_i(k)$. Let $cn_x^k(i,.)$ be $(\cup_{t,j}^{t \leq k} V_{j,x}^t) \rightarrow (x \in \cap_{t,j}^{t \leq k} s_{j,x}^t)$.

$$P_i(k) := \text{CSP}(A_i) \cup (\cup_x cn_x^k(i,.)) \cup \text{NV}_i(V_k^i) \cup \text{CL}_k^i$$

$(\cup_t V_t) \rightarrow \neg(\cup_t T_t)$, denoted CL_k^i, is obtained from the set of CLs available at A_i, having the form $V_t \rightarrow \neg T_t$, where $V_t \subseteq \cup_j^{j \leq k} V_k^i$. $cn_x^k(i,i)$ is the consistency nogood that is stored and sent to other agents by **propagate** messages iff any constraint of CSP(A_i) was used for its logical inference from $P_i(k)$ and its label shrinks. We now prove the correctness, completeness and termination properties of DMAC. We only use DC techniques that terminate.

Lemma 12.1 *DMAC is correct, complete, 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: By quiescence of a group of agents we mean that none of them will receive or generate any valid nogoods, new valid assignments, or addlink messages. We prove by induction on increasing i the **property**: *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* CSP(A^j)$\cup$NV$_j$(view(A^j)).

Let this be true for the agents $A^j, j<i$. A^i receives the last valid **ok?** message at time t_o^i. $\exists t_v^i$, $t_v^i \geq t_o^i$ such that after t_v^i, view(A^i) and all $V_k^u, k \leq i$ of any agent A_u are no longer modified. $\text{CL}_k^u, k \leq i$ cannot be invalidated after t_v^i. Since DCs were assumed to terminate, they terminate after each modification of a CL_k^u. Since the number of such modifications after t_v^i is bounded, after a finite time no consistency nogood is received any longer by A^i for levels $k \leq i$. Only one valid explicit nogood can be received for a proposal since the proposal is immediately changed on such an event. Similarly, there is a finite number of valid nogoods that can be received by A^i for any of its proposals made after t_v^i (and after t_o^i).

If one of the proposals is not refused by incoming nogoods, and since the number of such nogoods is finite, the induction step is correct. If all proposals that A^i can make after t_o^i are refused or if it cannot find any proposal, A^i has to send according to rules inherited from AAS a valid explicit nogood $\neg N$ to somebody. If N is empty, failure is detected and the induction step is proved. Otherwise $\neg N$ is sent to a predecessor $A^j, j<i$. From here, as proved for AAS, it results that either empty nogood is detected, or A^i will receive a new proposal. This contradicts the assumption that the last **ok?** message was received by A^i at time t_o^i and the induction step is proved.

The property can be attributed to an empty set of agents and it is therefore proved by induction for all agents.

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 as proved for AAS. □

If the agents using DMAC maintain all the valid consistency nogoods that they have received, then DCs in DMAC converge and compute a local consistent global problem at each level. If not all the valid consistency nogoods that they have received are stored, but some of them are

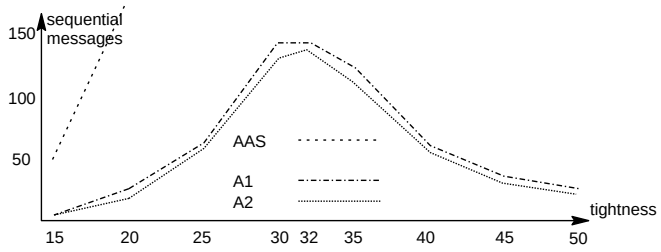

Figure 12.1: *Results averaged over 500 problems per point.*

discarded after inferring the corresponding $cn_x^k(i,i)$, 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 can stop before the global problem is DC consistent at that level.

Among the consistency nogoods that an agent computes itself from its constraints, $cn_x^k(i,i)$, let it store only the last valid one for each variable. Let A_i also store only the last valid consistency nogood, $cn_x^k(i,j)$, sent to it for each variable $x \in \text{CSP}(A_i)$ at each level k from any agent A_j. Then:

Proposition 12.1 *DC($\mathcal{A}$) labels computed at quiescence at any level with the help of* **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.*

Proof. In each sent **propagate** message, the consistency nogood for each variable is the same as the one maintained by the sender. 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 nogoods, are therefore identical and the distributed consistency will not stop at any level before the global problem is local consistent in each agent. □

Proposition 12.2 *The minimum space an agent needs with DMAC for insuring maintenance of the highest degree of consistency achievable with DC is $O(a^2v(v+d))$. With bound consistency, the required space is $O((av)^2)$.*

Proof. The space required for storing all valid assignments is $\text{O}(dav)$ for values and $\text{O}(av)$ for histories (stored separately). The agents need to maintain at most a levels, each of them dealing with v variables, for each of them having at most a 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 $\text{O}(a^2v(v+d))$. The space required by CL and by the algorithm for solving the local problem depends on the corresponding technique (e.g. chronological backtracking requires $\text{O}(v)$). CL also refers at most v assignments in its premise. □

12.1.2 Experimental comparison of asynchronous algorithms

We have run our tests on a local network of SUN stations where agents are placed on distinct computers. We use a technique that enables agents to process with higher priority **propagate** and **ok?** messages for lower levels.

The DC used in our experimental evaluation maintains bound-consistency. In each agent, computation at lower levels is given priority over computations at higher levels. We generated randomly problems with 15 variables of 8 values and graph density of 20%. Their constraints were randomly distributed in 20 subproblems for 20 agents. Figure 12.1 shows their behavior for variable tightness (percentage of feasible tuples in constraints), averaged over 500 problems per point. We tested two versions of DMAC, A1 and A2. A1 asynchronously maintains bound consistency at all levels. A2 is a relaxation where agents only compute consistency at levels where

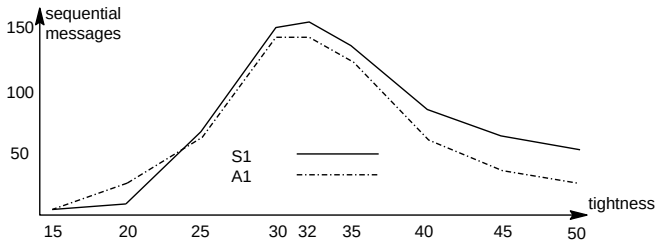

Figure 12.2: *Synchronous vs. Asynchronous. Results averaged over 500 problems per point.*

they receive new labels or assignments, not after reduction inheritance between levels. A2 is obtained in Algorithm 21 by performing the cycle starting at line 21.1 only for $t = k$, where k is the level of the incoming **ok?** or **propagate** message triggering it. In both cases, the performance of DMAC is significantly improved compared to that of AAS. Even for the easy points where AAS requires less than 2000 sequential messages, DMAC proved to be more than 10 times better in average. A2 was slightly better than A1 on average (excepting at tightness 15%). In these experiments we have stored only the minimal number of nogoods. The nogoods are the main gain of parallelism in asynchronous distributed search. Storing additional nogoods was shown for AAS to strongly improve performance of asynchronous search. As future research topic, we foresee the study of new nogood storing heuristics (Havens 1997; Yokoo *et al.* 1998; Turner & Phelps 2000; Silaghi *et al.* 2001h; Freuder *et al.* 2001).

12.1.3 Experimental comparison against synchronous algorithms

When instantiations and DC alternate sequentially (Tel 1999) the obtained protocol is referred to as sMDC. Since MAS can integrate 4 types of asynchronism (backtracking, consistency achievement, consistency maintenance,reordering), we define the notation MAS$\pm\pm\pm\pm$ where each + stands for the support of the corresponding type of asynchronism and − for the lack of support.

Since DCs terminate in finite time, sMDC is an instantiation of MAS, more exactly MAS-+--. However, the asynchronism in MAS-+-- is limited to small slices since the search itself is actually synchronous and any new decision has to be synchronized among all agents. MAS-+-+ does not exist yet. The notion of asynchronous reordering makes little sense with synchronous search. We have designed a version of MAS-+-- that is based on aggregations similarly to AAS. Our version is denoted S1 in Figure 12.2. In S1, termination of DCs is detected using a technique requiring one additional sequential message per search node. Each agent A_i that is being instantiated launches DC on known(A_i) after each instantiation, respectively on its currently allowed domains before changing its instantiation. A_i starts a new DC by sending the current labels for all variables to all uninstantiated agents. All the modified labels are sent by the agents running DC to A_i. After DC ends without domain wipe-out, A_i chooses the agent with the next position in the search to be an agent with the smallest volume (defined in Section 11.2) among all those that are not instantiated. All the labels are then sent to the next agent. The first DC is started by the agent A_0 which then chooses the agent with the first position. S1 was in average slightly worse than asynchronous versions, excepting very over-constrained problems.

12.1.4 Reordering Agents Asynchronously

Reordering in DMAC can be done with the same technique as in ABTR, since the technique reorders agents and does not depend on the problems enforced by these agents.

To use dynamic asynchronous reordering, besides the structures of DMAC, each agent needs to store an order. This is the newest order it has received. The structures that have to be maintained by R^k, as well as the content of the **heuristic** messages depend on the reordering heuristic. In our example (Figure 12.4), we consider the case where agents announce all the interested reordering leaders of all the labels they send. In Figure 12.3, the reordering leader of level k, R^k corresponds to the agent A^k. R^{-1} corresponds to the agent A_1. As previously mentioned, the volume of A_i

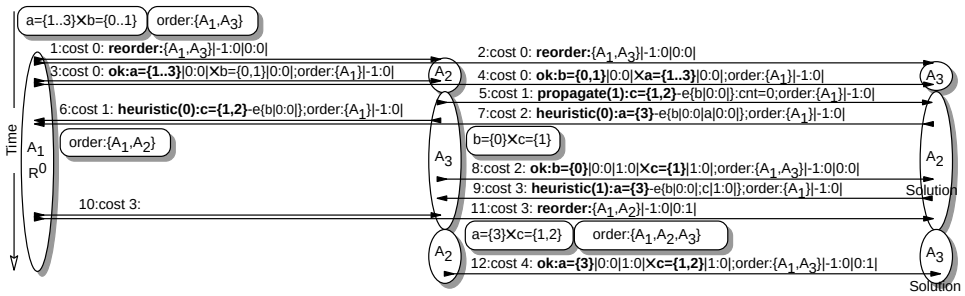

Figure 12.3: *A simple run of consistency with asynchronous reordering.* **add-link** *messages are redundant for this heuristic.*

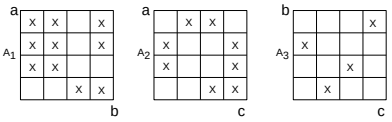

Figure 12.4: *A problem with 3 agents.*

is estimated as the product of the allowed size of domains for all variables in vars(A_i) that the corresponding agent has announced he wants to be allowed to change. R^i maintains for all variables and for the levels k, $k \leq i+1$ similar structures to those of DMAC at agent A^u_j for vars(A^u_j) and levels t, $t \leq u$. The space complexity remains the same as with DMAC.

Proposition 12.3 *MAS 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: We prove by induction on increasing i the same property as in the proof of lemma 12.1 where quiescence now also refers to **reorder** messages and to any **propagate** message of level $k \leq i+1$. Let all agents $A^j, j<i$ reach quiescence before time t^{i-1}. Reasoning as for lemma 12.1, $\exists t^i_h$ after which no propagate of level k, $k \leq i$ and therefore no heuristic message toward any R^u, $u<i$ is exchanged. Then, R^u becomes fixed, receives no message, and announces its last order before a time t^i_r. After t^i_r the identity of $A^k, k \leq i$ is fixed. A^i receives the last new assignment or order at time t^i_o. Reasoning as in lemma 12.1 for A^i and R^i, it follows that after t^i_o, within finite time, the agent A^i either finds a solution and quiescence or exhausts its possibilities and sends a valid explicit nogood to somebody. From here, the induction step is proven as in lemma 12.1. R^{-1} is always fixed and the property is true for the empty set. The termination results by induction on i.

Correctness: Results at quiescence for the newest order since all the agents know the interesting decisions of their predecessors. After any given isolated reordering decision, all the proposals of the reordered agents and the whole structures depending on them are reset, as if those agents would have delayed the start of the search up to that moment. Instead, the proposals of agents that do not change their position (fixed agents) are maintained. Valid nogoods transformed in redundant constraints reduce the choice of the agents, but this has no negative effect on required properties, as repeated in lemma 12.1. The same applies to eventual nogoods and labels that have been sent and integrated in fixed agents. The obtained run of MAS is therefore equivalent to an instance of a run of DMAC where agents got some additional redundant constraints. After a bounded time from each reorder decision the search behaves as DMAC, when no other reorder is issued, and all the properties are therefore maintained. We call *stable search tree branch* that branch of the distributed search tree that will no longer be invalidated by any reordering decision before being proved unsatisfiable. It remains to show that no set of agents are reordered in an infinite loop such that they can never behave as DMAC. Let the lower priority agent in this set be A^k. This results from the fact that the number of **ok?** and **propagate** messages of level lower than k is finite and was proved (in (Silaghi *et al.* 2000a)) to terminate. Therefore, for each stable search tree branch of the search, there is a finite time after which no **heuristic** message are issued, therefore no

reordering is done any longer for the agent at the next level. By inference on all the branches of the search, it results that the search terminates. Correctness and completeness are immediate. □

12.2 The MAS family

MAS is a mixture of compatible asynchronous techniques: abstraction, consistency, reordering. Here we review the names of all the known members of the MAS family, for the permutations of the above techniques. They are structured in the Figure 12.5.

Protocol	Description	Abstractions	Consistency	Reordering
ABT/DDB	(Yokoo *et al.* 1992; Bessière *et al.* 2001)			
AAS	(Silaghi *et al.* 2000a; Meseguer & Jiménez 2000)	X		
DMAC-ABT	(Silaghi *et al.* 2001l)		X	
DMAC	(Silaghi *et al.* 2001f)	X	X	
AWC/ABTR	(Yokoo 1995; Silaghi *et al.* 2001e)			X
AASR	(Silaghi *et al.* 2001k)	X		X
DMAC-ABTR	-		X	X
MAS	(Silaghi *et al.* 2001i)	X	X	X

Figure 12.5: The MAS family of asynchronous backtracking protocols

More than this, it can be noted that synchronous backtracking can be defined as an instance of a version of ABT, for certain timing policies. Introduction of synchronous consistency (Tel 1999), or synchronous reordering (Meisels & Razgon 2001), in synchronous backtracking define new elements of MAS (Figure 12.6). As shown in the Figure 12.6, synchronous versions of MAS are less described in literature, partly because they can be obtained in an obvious manner from centralized algorithms.

Protocol	Description	Abstractions	Consistency	Reordering
sBT/SyncDFSTBranching	(Yokoo *et al.* 1992; Collin *et al.* 2000)			
-	-	X		
sMC	(Tel 1999)		X	
sMDC	(Silaghi *et al.* 2000g)	X	X	
-	-			X
-	-	X		X
DFCR	(Meisels & Razgon 2001)		X	X
-	-	X	X	X

Figure 12.6: The MAS family of synchronous backtracking protocols

As new compatible asynchronous techniques will be detected, the number of distinct protocols will grow exponentially. An example can be seen in the technique for removing **add-link** messages (Bessière *et al.* 2001), which can be added to all members of MAS. A naming technique as the one used in (Silaghi *et al.* 2001i), $\text{MAS}_{(\mp\mp\mp\mp)}$, can help to refer some standard members of MAS:

The first $\mp$ signals the presence of asynchronism of instantiation. The second $\mp$ signals the presence of asynchronism of consistency maintenance. The third $\mp$ signals the presence of asynchronism of consistency maintenance. The fourth $\mp$ signals the presence of reordering.

12.3 Versions of MAS

Multiply Asynchronous Search (MAS) is the most general generic family of search algorithms proposed in this thesis. Some more improvements are shown for MAS in Replica-based MAS. However they are mainly modeling and implementation tricks. The main ingredients of MAS are:

x The search strategy of AAS. One can identify the next polynomial space versions:

p AAS with tagged proposals (AASp) (Silaghi *et al.* 2000a).

p' MAS with one modifier per variable (history only consists of a counter) (Silaghi *et al.* 2001e)

$p_1^{[']}$ MAS with conflict lists - AAS0_1 (MAS-p_1) (Silaghi *et al.* 2000a)

$p_2^{[']}$ MAS with conflict lists - AAS0_2 (MAS-p_2) (Silaghi *et al.* 2000a)

r AAS with proposal retransmission (AASr).

r' AASr with conflict list

y The view update strategy in AAS can be:

* AAS with ABT-like dynamically added links (Yokoo *et al.* 2002).

l AAS without add links (AAS**l), as DDB (Bessière *et al.* 2001).

s AAS with one-shot add links (AAS**s) (Silaghi *et al.* 2001i).

d AAS with pre-added links (Hamadi 1999b).

z The nogood validity strategy in AAS can be:

e AAS with extended nogood storage (AASe). This can be obtained from the version of AAS described in (Silaghi *et al.* 2000a), namely integrating the nogood storage philosophy of GOB.

t The consistency maintenance of DMAC. There are three extensively described versions:

g Consistency maintenance with guaranteed strength given by the last label/variable/level/agent (Silaghi *et al.* 2001l).

v Consistency maintenance with guaranteed strength given by the last value/variable/level/agent (as in DMAC_1, see Remark 7.5).

h Consistency maintenance with greedy combination of labels (as in MHDC) (Silaghi *et al.* 2000g).

Many other versions are known: guaranteed strength with several labels, broadcast of labels, different ways of announcing explicit nogoods from consistency nogoods. As shown in Section 7.5, several versions exist for DMAC, namely $\text{DMAC}^{[']|['']|[''']][k]}$. The corresponding notation is $\text{t}^{[']|['']|[''']][k]}$.

w There exist several types of consistency that can be enforced:

A arc consistency

B bound consistency

u Reordering technique. Currently, there exists a reordering technique encompassing all known polynomial versions.

o ABTR-based reordering (Silaghi *et al.* 2001e).

b bounded number of reorderings (Yokoo 1993b).

i increasing delay reordering (Silaghi *et al.* 2001h).

n reordering after increasing number of discovered nogoods (the version n appeared in a discussion with Makoto Yokoo in 2002).

Give the main versions of the components of MAS described above, a sufficiently comprehensive notation is $\text{MAS}_{(\mp\mp\mp\mp)}$-xyz-twu. Compactly the set of existing variants is described by the regular expression: $\text{MAS}_{(\mp\mp\mp\mp)} - (p_{(1|2)}^{[']}|r^{[']})[(*|l|s|d)[e]] - [(g|h|v)^{[']|['']|[''']][k}[A|C][o|b|i|n]]$.

Example 12.15 *Given the previously mentioned notation, MAS-rs-h stands for MAS based on AASr, without conflict lists and with one-shot links, with greedy consistency maintenance like MHDC.*

12.4 Summary

In this chapter we have seen that the techniques presented in previous chapters of this thesis can be combined in a straightforward manner. A general notation has been introduced for naming unambiguously existing algorithms.