Chapter 13

Replicas-based Distributed CSPs

Who despises the day of small beginnings?
Zacharia 4:10.

In the previous chapter I show that many DisCSP techniques can be easily combined and Multiply Asynchronous Search (MAS) was introduced.

This chapter proposes a technique for modeling DisCSPs in a way that transforms MAS in an even stronger protocol. The modeling technique is called Replica-based DisCSPs. MAS can be optimized for RDisCSPs, obtaining a protocol we name RMAS.

13.1 Granularity of Consistency

As defined above, DMAC maintains consistency during asynchronous search, as DMAC-ABT. There exists, however, an important detail: The consistency is maintained in DMAC not after each variable is instantiated, but after each agent makes a proposal. Depending on the made proposals, the resulting granularity of the consistency that can be achieved can be both: higher or smaller.

Example 13.16 *Given an agent A_1 with a problem:*

$$(x_1, x_2) \in (\{0,1,2,3\} \times \{0,1,2,3\}) \setminus \{(0,0),(0,1),(1,0),(1,1)\}$$

When A_1 proposes $x_1 \in \{2,3\}$, the consistency is maintained with a granularity that is higher than in DMAC-ABT when A_1 proposes $x_1 = \{2\}$, or $x_1 = \{3\}$. The higher granularity leads to efficiency improvements since the values eliminated at this step in DMAC need not be considered twice on the two branches generated otherwise in DMAC-ABT.

On the other side, when A_1 proposes $(x_1, x_2) = (0,3)$, the consistency is maintained with a granularity that is lower than in DMAC-ABT when A_1 proposes $x_1 = 0$ and A_2 proposes $x_2 = 3$. Such a proposal is therefore less efficient and DMAC-ABT can directly eliminate $x_1 = 0$ with consistency inference, while in DMAC one may have to separately infer the infeasibility of $(x_1, x_2) = (0,2)$ and $(x_1, x_2) = (0,3)$.

Remark 13.1 *Due to the fact that DMAC and AAS are based on nogoods, it is possible for the agents to generate intelligent nogoods balancing in a certain measure the lack of granularity. This is the reason why AAS and DMAC are efficient even when proposals have bad granularity.*

In the following we show how the agents can model their problem in such a way that they can use MAS and tune the amount of consistency maintenance granularity that they want to be applied to their proposals.

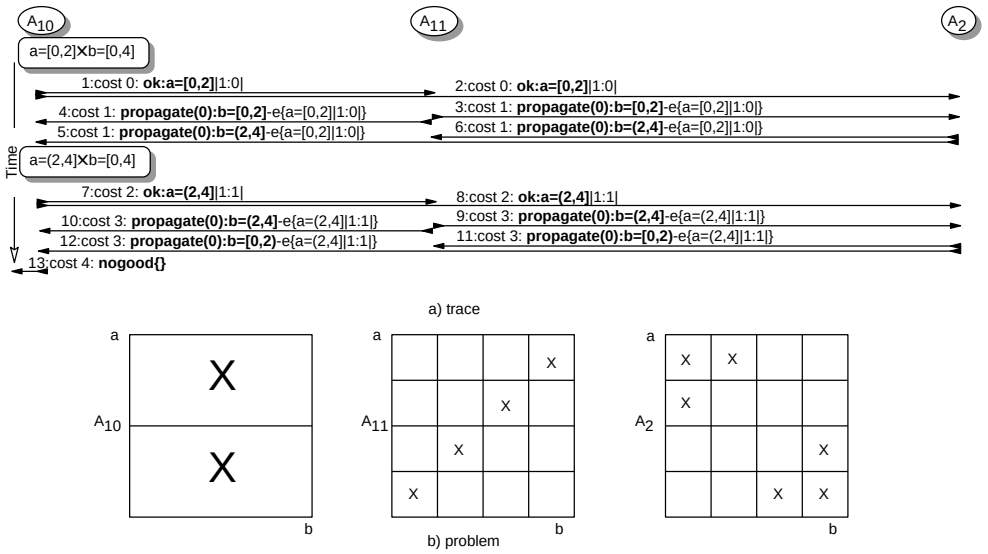

Figure 13.1: *Domain Splitting. Very rough simplified example for A_1: $a==b$; A_2: $|a-b|>2$.*

13.2 Domain splitting

One of the ways of approaching a large problem is by successively solving increasingly detailed abstractions of it, until the original problem is solved. In the following sections we show how distributed search can be performed asynchronously and in parallel over all abstractions. The approach is appropriate for numerical CSPs.

The approaches that enumerate values in domains, feasible tuples in constraints or aggregates of such feasible tuples are called here *enumeration techniques*. The inherent disadvantage of enumeration techniques is the dependence of the efficiency of the search for finding a solution on the order of the elements. Their efficiency for proving unsatisfiability is also dependent on the size of the problems. The heuristics for value reordering theoretically can help, but are seldom sufficiently informed. For hard problems with large (e.g. numerical) domains and weak value reordering heuristics, the enumeration techniques are therefore hopeless. However, when the domains are "naturally" ordered, which means that the constraints have a high *density of the current value order* (Silaghi *et al.* 2000c), dichotomous techniques that recursively split domains into halves often improve efficiency on difficult problems. The best known examples consist of the numerical constraint satisfaction problems that can be solved when dichotomous techniques are combined with some kind of bound consistency (Silaghi *et al.* 2001j; Van Hentenryck 1998).

13.3 Replica-based DisCSPs (RDisCSPs)

Implementing asynchronous dichotomous behavior in the distributed context of DMAC requires special treatment. The main impediment comes from the fact that the number of consistency levels in DMAC equals the number of agents. In DMAC the constraints are private to agents and simple dichotomous behavior would hinder the agents with high priority from requiring the satisfaction of their constraints. To overcome this problem:

Definition 13.1 (Replicas) *Each agent A_i owning private constraints can be represented in the search by a set of k_i+1 replicas (abstract agents, see Definition 8.3) denoted $A_{i_0}, ..., A_{i_{k_i}}$.*

Assume that $A_{i_{k_i}}$ is ordered after all other replicas of A_i.

Definition 13.2 (checking replica) *The replica $A_{i_{k_i}}$ is called checking replica of A_i and has the goal usually taken by A_i in DMAC.*

A checking replica of A_i makes sure that the subproblem proposed to its followers is consistent with all the private constraints of A_i.

Definition 13.3 (artificial replica) *The artificial replicas are agents that are not checking replica of any agent.*

All the *artificial replicas* have to behave according to conservative splitting and reduction operators, and send messages consistent with the DMAC protocol.

We call this framework Replicas-based DisCSP.

13.3.1 Distributed Dichotomous Splitting

We now present an algorithm for Asynchronous Dichotomous search Maintaining Bound consistency (ADMB). In Figure 13.1 is given a small example where domain splitting is used to solve a problem with two agents. We only show two replicas A_{1_0} and A_{1_1} standing for A_1. A_{1_0} proposes the halves of the domain of variable a. The search space is exhausted with 4 sequential messages. Without domain splitting the minimal number of sequential messages is 6. In the given trace, we consider that consistency nogoods are sent to all agents (one of the heuristics that can be used in DMAC (Silaghi *et al.* 2001g)). Therefore, the consistency-based domain wipe-out can be detected by the agent generating the faulty level. Otherwise, nogood messages would be sent by the agents A_{1_1} and A_2 to A_{1_0}.

Another aspect is that the number of consistency levels depends on the pruning computed by bound consistency during search and cannot be safely estimated in advance. A straightforward way to solve this problem is by using a number of *artificial replicas* equal with the maximal number of consistency levels. However, this can be a rather expensive solution for large problems (numerical problems where very fine resolution is required). Two alternative solutions can be imagined. The first alternative, **static agent configuration**, consists of allowing existing *artificial replicas* to abandon their current proposal and generate new splits. Some nogoods are lost due to this procedure. The second one, **dynamic replica spawning**, consists of dynamically spawning new replicas whenever the current search space of the *checking replicas* is not contained in the solution space. *replica spawning methods* and *static agent configuration methods* are described in (Silaghi *et al.* 2001i).

The volume (see Section 11.2) of a view V of an agent A_i, volume(V), is given by the product of the size of the allowed domains for the variables in CSP(A_i) as allowed by the view V. The role of the artificial replicas is to make sure that the volume of the search space that they propose is (with a certain tolerance) half of the volume of the search space that they were proposed. Namely, checking replicas must generate aggregates such that:

$$|\text{volume}(\text{view}(A_i))/2\text{-volume}(\text{known}(A_i))| < k \text{ volume}(\text{known}(A_i)).$$

Whenever this condition is not respected due to modifications brought to the domains by distributed consistency, the artificial replicas will generate a new proposal adjusting known(A_i). When a proposal of an artificial replicas is refused by some *nogood* message, half of the other half (or the other half, when the maximum resolution ε — e.g. required resolution or an element of $\mathcal{F}$ for numerical constraints– is reached) of the search space is proposed. When its whole search space is similarly refused, an artificial replica sends an explicit nogood generated by inference from the received nogoods, the nogood entailed by the view and those entailed by consistency.

Whenever the whole search space of an *artificial replica* A_{i_k} is feasible for A_i, A_{i_k} does not need to split its search space (by generating new proposal). The feasibility test can be done for "continuous" domains in Numerical CSPs using the technique of Benhamou&Goualard employed in (Silaghi *et al.* 2001j). The variable that will be split first can be chosen to be the one with biggest impact on the future search (see (Silaghi *et al.* 2000c; 2001j)). The heuristic in (Silaghi *et al.* 2000c) is available since the agents declare their interest on variables. Alternatively to the dichotomous one, splitting techniques can be based on constraint analysis. Namely, each artificial replica A_{i_k} can choose a constraint of A_i and devise a split according to heuristics based on the

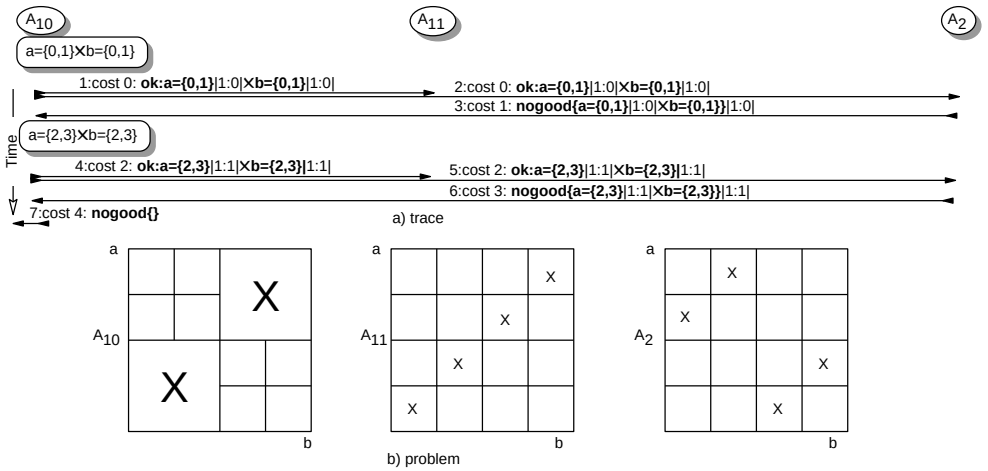

Figure 13.2: *Tuples Clustering Method.*

constraint structure as described in (Silaghi *et al.* 2001j). The artificial replicas need not generate consistency nogoods since their work would be redundant to the one of the checking replicas which enforce all the corresponding constraints.

To avoid generating new physical agents, a physical agent A'_i acts for all replicas of A_i. Messages are sent in only one transmission to all the replicas for which the receiver agent acts. In terms of internal structures, letting an existing agent act for a new replica amounts to storing new consistency levels and an additional CL. The final number of levels is upper-bounded by $\sum_{i \in X} \log_{2/(1+2k)} |D_i|$. The *checking replicas* are ordered after all the *artificial replicas*.

A related work for centralized CSPs is proposed in (Jussien & Lhomme 1998). Their algorithm generates dynamically new constraints that can be added and retracted using explanations. A difference is that (Jussien & Lhomme 1998) maintains only one label per variable and are restricted to the reordering in (Ginsberg 1993a).

13.3.2 Constraint Splitting

The private problem of an agent can often be represented as a set of constraints. Let us consider that CSP(A_i) can be split in k_i+1 problems CSP(A_{i_0}),...,CSP($A_{i_{k_i}}$) (e.g. corresponding to variables or constraints of CSP(A_i)). A_i can therefore have k_i+1 replicas, each of them dealing with a subproblem of CSP(A_i) in such a way that $\forall j$, CSP(A_{i_j})$\subseteq$CSP(A_i) and $\cup_{j \in \overline{0,k_i}}$CSP($A_{i_j}$)=CSP($A_i$). We mention that such a distribution makes sure that any solution space accepted by all the agents A_{i_j} is acceptable to A_i. MAS solves the problem modeled this way, and the additional consistency levels introduced by replicas are a means for reusing nogoods at backtracking.

13.3.3 Tuples Clustering

Let us assume that each agent owns exactly one constraint (either obtained by the composition of several constraints, or by problem splitting with replicas as in section 13.3.2). Let us now assume that for each agent A_i owning exactly one constraint we use k_i+1 replicas, ordered A_{i_0},...,$A_{i_{k_i}}$, such that CSP($A_{i_{k_i}}$)=CSP(A_i) while any CSP($A_{i_{j,j<k_i}}$) is a relaxation of CSP(A_i). For a constraint with infinite domains, such relaxations can be built by merging elements of some rough outer covering. For any constraint with finite domains, such a relaxation can be obtained by means of *clustering methods* (Binary splitting, WARD, etc.) with *seeds* usually employed in automated classification techniques (e.g. speech recognition) (Boite *et al.* 2000). One such technique consists of distributing a predefined number of points (seeds) uniformly in the search space defined by the variables in the constraint. Iteratively, each feasible tuple is associated with the closest seed

(Euclidean distance) and then each seed is moved to the center of mass for the partition that the seed i defines on feasible tuples. When this iteration converges, for each seed i we compute the minimal box B_i that encloses the whole partition it defines. B_i is given by the minimal and maximal values of each variable over the feasible tuples associated to the seed i. Therefore we get a relaxation composed of a number of potentially overlapping boxes at most equal with the number of initial seeds.

Proposition 13.1 *A constraint obtained by clustering can be decomposed in maximum ks^2 disjoint boxes (covers) where k is the arity of the constraint and s is the number of seeds.*

Proof. Each box needs at least one *cover*. For each already covered box, a not yet covered box may need $2k-1$ additional disjoint boxes to be covered. The total number is therefore $s+(2k-1)(s-1)s/2 = ks^2+s(1-k)+s(1-s)/2 \leq ks^2, \forall s,k \geq 1$. □

Each CSP($A_{i_{j,j<k_i}}$) can be built with a number of seeds that is a strictly monotonically ascending function on j. This heuristically yields an increasing detail of the abstraction. The covering boxes can be computed either statically or dynamically. Statically computed covering boxes can either be used without modification or they can be re-aggregated if that is possible when given proposals are made. A second problem is that when many seeds fall out during the clustering, it is possible to obtain the same abstraction in different replicas. Figure 13.2 shows an example where **tuples clustering** is used with one replica for A_1.

Both the **constraint splitting** and the **tuples clustering method** are complete and correct since there exist equivalent problems for which they are identical to MAS. The overhead consists of the communication with the new agents and can be reduced when the replicas are explicitly collocated. The **tuples clustering** is a special case of domain splitting.

13.4 Dynamic Replica Spawning?

Here we shortly describe a method for dynamically spawning replicas. The next new messages are introduced:

- **spawn?** messages are sent from broker at quiescence to ask checking replicas if they are satisfied with the current solutions.
- **replicate** messages are sent from checking replicas to broker as answer to **spawn?** messages. They signal the existence of infeasible tuples in the newest proposals known by the sender and have as parameter the address of a new replica.
- **satisfied** messages are sent from checking replicas to broker in answer to **spawn?** messages. They signal that the space defined by the newest proposals known by the sender is feasible.
- **spawn** messages are sent by the broker to all agents and announce the insertion of the new replicas in search.

Dynamic replica spawning (DRS) is achieved by running MAS in parallel with a solution detection and a termination/quiescence detection process. For this approach, the *checking replicas* need not create corresponding consistency levels by enumerating proposals, but only generate **accepted** messages when the whole space that was proposed to them is feasible for their constraints. For, the solution detection algorithm, the same technique can be used as for AAS. The termination/quiescence detection uses counters similarly with sMDC (Silaghi *et al.* 2000g; 2001g) as presented later. When the termination detection algorithm signals silence on the network, the broker sends **spawn?** messages to the *checking replicas*. The *checking replicas* that are not satisfied by the current domains send a **replicate** message to the broker, while the others send a **satisfied** message. Each **replicate** message contains the address of a new replica of the sender. The broker will then send a **spawn** message to all the agents, establishing a priority for the new replicas between the old *artificial replicas* and the *checking replicas*. The *artificial replicas* treat **spawn** messages as **add-link** for the common variables. The *checking replicas* initialize the new

corresponding consistency levels. To maintain coherence of the views and consistency levels of the agents, they must not restart the search before the broker acknowledges them that everybody has received the **spawn** message.[1]

The termination is ensured by the fact that whenever all agents answer with satisfied, it is guaranteed that a solution will be detected in finite time.

An algorithm with **static agent configuration** can also be obtained from DRS when a neutral agent A_r acts as a replica and is positioned between artificial and checking replicas. Whenever quiescence is detected, no agent is added, rather A_r is requested to split a domain for a variable of some agent that has sent a **replicate** message. The splitting is conservative, so that the eliminated space is tried on backtracking. It is no longer the broker that detects quiescence but A_r. The disadvantage consists in the confidence in the fairness of A_r and the loss of nogoods.

13.5 Experiments

The dichotomous domain splitting and the tuples clustering without replica spawning were preliminarily simulated with an improvised system using my implementation for MAS.

Both algorithms worked correctly according to the theory described here. The system I improvised did not allow me to take usual measures of the efficiency (Chapter A) and the running time was also very slow. However, the running time was so bad due to the inadequacy of the improvisation used for the preliminary simulations. I am currently working for a better implementation.

13.6 Versions of R-MAS

Replica-based Multiply Asynchronous Search (R-MAS) is the most general generic family of search algorithms proposed in this thesis.

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

13.7 Summary

In the end of this chapter we have seen how appropriate modeling can allow the agents to control the granularity of their proposal, and how this can be used to maintain consistency with a high granularity. The new framework is called Replica-based DisCSPs, and the obtained protocol is called Replica-based Multiply Asynchronous Search (RMAS).

When I proposed to make my thesis on this topic almost three years ago (Silaghi & Faltings 1999), wise researchers smiled with indulgence. And indeed, even if I already had all the required components in mind, their development was as complex as expected.

[1] Alternatively, an ID of the last **spawn** message can be attached to any message. No message is then processed before the corresponding **spawn** is received.