ASYNCHRONOUS CONSISTENCY MAINTENANCE

MARIUS-CĂLIN SILAGHI, DJAMILA SAM-HAROUD, AND BOI FALTINGS
EPFL, CH-1015, Switzerland
{*Marius.Silaghi,Djamila.Haroud,Boi.Faltings*}*@epfl.ch*

Maintaining local consistency during backtrack search is one of the most powerful techniques for solving centralized constraint satisfaction problems (CSPs). Yet, no work has been reported on such a combination in asynchronous settings. The difficulty in this case is that, in the usual algorithms, the instantiation and consistency enforcement steps must alternate sequentially. When brought to a distributed setting, a similar approach forces the search algorithm to be synchronous in order to benefit from consistency maintenance. Asynchronism [1,2] is highly desirable since it increases parallelism and makes the solving process robust against timing variations. This paper shows how an *asynchronous* algorithm for maintaining consistency during distributed search can be designed. The proposed algorithm is complete and has polynomial-space complexity. Experimental evaluations show that it brings substantial gains in computational power compared with existing asynchronous algorithms.

1 Introduction

A constraint satisfaction problem (CSP) is defined as a set of variables taking their values in particular domains and subject to constraints that specify consistent value combinations. Distributed constraint satisfaction problems (DisCSPs) arise when the constraints or variables come from a set of independent but communicating agents. The most successful centralized algorithms for solving CSPs combine search with local consistency. The local consistency algorithms prune from the domains of variables the values that are locally inconsistent with the constraints, hence reducing the search effort. When a DisCSP is solved by search using a distributed network of agents, it is desirable that this search exploits asynchronism as much as possible. Asynchronism gives the agents more freedom in the way they can contribute to search. It also increases both parallelism and robustness. In particular, robustness is improved by the fact that the search can still detect unsatisfiability even in the presence of crashed agents. The existing work on asynchronous algorithms for distributed CSPs has focused on one of the following types of asynchronism:

a) *deciding instantiations* of variables by distinct agents. The agents can propose different instantiations asynchronously.

b) *enforcing consistency*. The distributed process of achieving "local" consistency on the global problem is asynchronous (e.g. Distributed Arc Consistency [3]).

We show how these techniques can be combined without losing asynchronism.

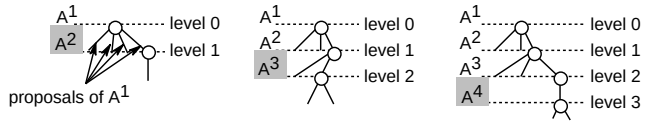

Figure 1. *Distributed search trees: simultaneous views of distributed search seen by A_2, A_4, and A_4, respectively. Each arc corresponds to a proposal from A_{i-1} to A_i.*

2 Preliminaries

Asynchronous search In this paper we target problems with finite domains. We consider that each agent A_i wants to satisfy a local CSP, CSP(A_i). The agents may keep their constraints private but publish their interest on variables. The technique we propose builds on Asynchronous Aggregation Search (AAS), a general complete protocol for solving distributed CSPs with polynomial space requirements [2]. AAS is an extension of Asynchronous Backtracking (ABT) and allows for asynchronism of type a. AAS uses a strict order on agents. We assume that A_j has the position $j, j \geq 1$. If $j > k$, we say that A_j has a lower priority than A_k. A_j is then a successor of A_k, and A_k a predecessor of A_j.

Asynchronous distributed consistency The centralized local-consistency algorithms prune from the domain of variables the values that are locally inconsistent with the constraints. Their distributed counterparts (e.g. [3]) work by exchanging messages on value elimination. The restricted domains resulting from such a pruning are called *labels*. In this paper we will only consider the local consistencies algorithms which work on labels for *individual* variables (e.g. arc-, bound-consistency). Let P be a Distributed CSP with the agents $A_i, i \in \{1..n\}$. We denote by $C(P)$ the CSP defined by $\cup_{i \in \{1..n\}}$CSP(A_i). 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 elimination, the same labels for P as $\mathcal{A}$ for $C(P)$. When DC($\mathcal{A}$) is run on P, we say that P becomes DC($\mathcal{A}$) consistent.

3 Asynchronous consistency maintenance

In distributed search, each agent has its own perception of the distributed search tree. It is determined by the proposals received from its predecessors. In Figure 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. Suppose that A_4 has not received anything valid from A_3, A_4 will assume that A_3 agrees with A_2. The term *level* in Figure 1

refers to the depth in the (distributed) search tree viewed by an agent. We show that A_i can then benefit from the value eliminations resulting by local consistency from the proposals of *subsets* of its predecessors, as soon as available.

4 The DMAC protocol

This section presents DMAC (Distributed Maintaining Asynchronous Consistency), a complete protocol for maintaining asynchronous consistency, built on AAS.

Definition 1 (Aggregate) *An aggregate is a triplet (x_j, s_j, h_j) where x_j is a variable, s_j a set of values for x_j, $s_j \neq \emptyset$, and h_j a history of the pair (x_j, s_j).*

The history guarantees a correct message ordering. Let $a_1 = (x_j, s_j, h_j)$ and $a_2 = (x_j, s'_j, h'_j)$ be two aggregates for the variable x_j. a_1 is *newer* than a_2 if h_j is more recent than h'_j. The ordering of histories is described in full detail in [4]. The newest aggregates received by an agent A_i define its *view*, view(A_i). An *aggregate-set* is a set of aggregates. Let V be an aggregate-set and vars(A_i) the variables of CSP(A_i). $T_i(V)$ will denote the set of tuples directly disabled from CSP(A_i) by V.

Definition 2 (Nogood entailed by the view) *$V' \rightarrow \neg T_i(V)$ is a nogood entailed for A_i by its view V, denoted $NV_i(V)$, iff $V' \subseteq V$ and $T(V') = T(V)$.*

Definition 3 (Explicit nogood) *An explicit nogood has the form $\neg V$, or $V \rightarrow$fail, where V is an aggregate-set.*

The information in the received nogoods that is necessary for completeness can be stored compactly in a polynomial space structure called conflict list nogood.

Definition 4 (Conflict list nogood) *A conflict list nogood, denoted by CL, for $\mathbf{A_i}$ has the form $V \rightarrow \neg T$, where $V \subseteq$view(A_i) and T is a set of tuples: $T = \{t \ | t = (x_{t^1} = v_t^1, ..., x_{t^{n_t}} = v_t^{n_t}), \forall k, x_{t^k} \in$vars($A_i$)$\}$, such that T can be represented by the structures (stack) of a complete centralized backtracking algorithm.*

An aggregate with history h_x^j built by A_j for a variable x is *valid* for an agent A_m, $m \geq j$, if no other history known by A_m and built by agents A_k, $k \leq j$, in some aggregate for x, is more recent than h_x^j. A nogood containing only valid aggregates is valid. The AAS protocol is defined by the `ok`, `nogood` and `addlink` messages. The `ok` messages have as parameter an aggregate-set, V. They announce proposals of domains for a set of variables and are sent from agents with lower priorities to agents with higher priorities. The proposal is sent to all successor agents interested in it. Let the set of valid aggregates known to the sender A_i be denoted known(A_i). $V \subseteq$known(A_i). Any tuple not in T_i(known(A_i)) must satisfy the local constraints of the sender A_i and its valid nogoods[a]. An agent maintains its view and a valid CL and always enforces its CL and its nogood entailed by the view. `nogood` messages

[a] Except for constraints about which A_i knows that a successor enforces them (as in ABT).

annouce explicit nogoods. Any received valid explicit nogood is merged into the maintained CL using an inference technique.

4.1 DMAC

In addition to the messages of AAS, the agents in DMAC may exchange information about nogoods inferred by DCs. This is done using `propagate` messages.

Definition 5 (Consistency nogood) *A consistency nogood for a level k and a variable x has the form $V \rightarrow (x \in l_x^k)$ or $V \rightarrow \neg(x \in s \setminus l_x^k)$. V is an aggregate-set and may contain for x an aggregate (x,s,h), $l_x^k \subset s$. Any aggregate in V must have been proposed by predecessors of A_{k+1}. l_x^k is a label, $l_x^k \neq \emptyset$.*

Each consistency nogood for a variable x and a level k is tagged with the value of a counter C_x^k at sender and is sent via `propagate` messages to all interested agents $A_i, i \geq k$. The agents A_i use the most recent proposals of the agents $A_j, j \leq k$ when they compute DC consistent labels. A_i may receive valid consistency nogoods of level k with aggregate-sets for the variables *vars*, *vars* not in vars(A_i). A_i must then send `addlink` messages to all agents $A_{k'}, k' \leq k$ 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, for any level k, $k \leq i$:

- The aggregate-set, V_k^i, of the newest valid aggregates 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 $cn_x^k(i,j)$, if it is valid. It has the form $V_{j,x}^k \rightarrow (x \in s_{j,x}^k)$.

Let $cn_x^k(i,.)$ be $(\cup_{t,j}^{t \leq k} V_{j,x}^t) \rightarrow (x \in \cap_{t,j}^{t \leq k} s_{j,x}^t)$. $P_i(k) := \mathrm{CSP}(A_i) \cup (\cup_x cn_x^k(i,.)) \cup \mathrm{NV}_i(V_k^i) \cup \mathrm{CL}_k^i$. C_x^k is incremented each time a new $cn_x^k(i,i)$ is stored.

On each modification of $P_i(k)$, $cn_x^k(i,i)$ is recomputed by inference (e.g. using local consistency techniques) for each variable x for the problem $P_i(k)$. $cn_x^k(i,i)$ is initialized as an empty constraint set. CL_k^i is the set of all nogoods known by A_i and having the form $V \rightarrow \neg T$ where $V \subseteq V_k^i$ and T is a set of tuples in $\mathrm{CSP}(A_i)$. CL_k^i may contain the CL of A_i. An agent can manage to maintain one CL for each instantiation level and the space requirements do no change. $cn_x^k(i,i)$ is stored and sent to other agents by `propagate` messages if and only if any constraint of $\mathrm{CSP}(A_i)$ or CL_k^i was used for its logical inference from $P_i(k)$ and its label shrinks.

We only use DC techniques that terminate (e.g. [3]). By quiescence of a group of agents we mean that none of them will receive or generate any valid nogoods, new valid aggregates, or addlink messages. The proofs are given in [5].

Property 1 *$\forall i$ 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)).

Proposition 1 *DMAC is correct, complete, and terminates.*

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 valid one for each variable. Let A_i also store only the last (with highest tag) valid consistency nogood, $cn_x^k(i,j)$, sent to it for each variable $x \in$CSP(A_i) at each level k from any agent A_j. Then:

Proposition 2 *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.*

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

Additional nogoods can be stored as redundant constraints.

5 Conclusion

Consistency maintenance is one of the most powerful techniques for solving centralized CSPs. Bringing similar techniques to an asynchronous setting poses the problem of how search can be asynchronous when instantiation and consistency enforcement steps are combined. We present a solution to this problem. A new distributed search protocol which allows for *asynchronously maintaining* distributed consistency with polynomial space complexity is then proposed.

References

1. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The Distributed CSP: Formalization and algorithms. *IEEE Trans. on KDE, 10(5):673–685, 98.*
2. *M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In* Proc. of AAAI2000, *pages 917–922, 2000.*
3. *Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In* Proc. of Third IEEE Symposium on Parallel and Distributed Processing, *pages 394–397, 91.*
4. *M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. ABT with asynchronous reordering. In* IAT, *01.*
5. *M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous consistency maintenance with reordering. Technical Report #01/360, EPFL, March 2001.*