SECURE ASYNCHRONOUS SEARCH

MARIUS-CĂLIN SILAGHI, DJAMILA SAM-HAROUD, AND BOI FALTINGS
Swiss Federal Institute of Technology Lausanne
1015 Ecublens,Switzerland
{*silaghi,haroud,faltings*}*@lia.di.epfl.ch*

Distributed Constraint Satisfaction (DisCSP) is a general framework for modeling distributed combinatorial problems. Practical distributed problems can involve competitional situations among agents. In such cases not all agents are needed for building a final solution and agents may try to hamper their competitors from reaching a solution. This peculiarity cannot be modeled by current DisCSP formalisms. In particular, there is no mechanism for thwarting agents from disseminating fake nogoods in order to impede their competitors from proposing or reaching a good solution. In this paper we extend a DisCSP framework in order to model competition. We adapt one of the most recent search algorithms to offer agents means to check that received messages are legal.

1 Introduction

A wide variety of problems such as negotiation, resource allocation, design or scheduling are inherently distributed. Importing techniques from a problem to another is easier when general frameworks are used. Distributed Constraint Satisfaction (DisCSP) provides such a framework for static distributed combinatorial problems. A DisCSP is composed of a set of agents $A = \{A_1, A_2, ...A_n\}$ and a set of k variables $V = \{v_1, v_2, ...v_k\}$, each of them under the control of the agents interested in it. The variables in V are called external variables. With each agent A_i is associated a set of external variables $V_i = \{v_{i1}, v_{i2}, ..., v_{im_i}\}$, $V_i \subseteq V$, and a set of constraints $C_i = \{c_{i1}, c_{i2}, ...c_{ik_i}\}$ such that any external variable constrained by a constraint in C_i is also contained in V_i. The domain of a variable v_i is D_i. All the variables x_j constrained by constraints in C_i, and such that $x_j \notin V_i$ are said to be internal.

In problems with self-interested agents, the agents can actually be competing for a resource or state and some agents can reach a solution without the agreement of some others (e.g. with several clients or several providers). The competitors are interested in concealing solutions they dislike. Often they can do it by illegally generating nogood messages for solutions that normally do not need their agreement. The existing distributed protocols for DisCSPs do not offer the possibility to check these byzantine failures.

In this paper, we present an extended DisCSP framework that can model these missing features. Since in practice competition between agents often

occurs in conjunction with negotiation problems, the extended framework also enables the agents to attach *preferences* to their alternatives and to *relax* their constraints. We assume that the sum of preferences in solutions has to be minimized. The relaxation consists in either reducing values or, as proposed in [1], in accepting new tuples of valuations. The new algorithms can prevent agents neither from making coalitions, nor from byzantine failures that act against themselves. However, the new technique helps agents to avoid being cheated with the help of the distributed search protocol.

2 Dynamic DisCSPs

By dynamism we understand that the participation of an agent to the solution/search process is dynamically conditioned by certain value assignments. The extended framework builds on the notion of Valued CSPs [2]. First we describe the problem of an agent, A_u, as a Negotiation Valued CSP, (NVCSP$_u$). NVCSP$_u$ consists of a minimal increment, ε, a set of external variables, V(u), and an ordered set of global constraints, c_1(u),...,c_{n_u}(u). The domain of each external variable contains a value, F, meaning *unchanged* and *indifferent*. Each pair (valuation v, constraint c_i(u)) has associated a tuple:

$$T_i^v(u) = (feasible_i^v(u), preference_i^v(u)).$$

$T_i^v(u)$ is such that if $n_u \geq$i>j>0 then for any valuation v, $feasible_j^v(u) \rightarrow feasible_i^v(u)$ and $preference_i^v(u) \leq preference_j^v(u)$. There exists a valuation, v, such that either $feasible_i^v(u) \neq feasible_j^v(u)$, or otherwise $feasible_i^v(u) = feasible_j^v(u) = T$ and $preference_i^v(u) + \varepsilon \leq preference_j^v(u)$.

A Dynamic DisCSP (DyDisCSP) is defined by a set of agents A_0,...,A_n. $A_{k,k=[0,h),n\geq h>0}$ are h agents called *initiators*. Each agent A_j owns a NVCSP, NVCSP$_j$. Given a valuation, v, for a set of external variables, $S(v)$ is the set of agents owning a variable not instantiated in v to F. By convention, the *initiators* always belong to $S(v)$. An agent is *active* if it belongs to the minimal subset, $A(v)$, of $S(v)$ such that $S(\Pi_{vars(A(v))}v) \cap (S(v) \setminus A(v)) = \emptyset$.

Definition 1 (Acceptable valuation) *A valuation v is acceptable if each agent in $S(v)$ proposes for v a feasible associated tuple ($feasible_{k_i}^v(i) = T$).*

Definition 2 (Solution) *A solution of a DyDisCSP is an acceptable valuation v of all the external variables such that if each agent A_i in $S(v)$ is active and proposes for v an associated tuple ($T, preference_{k_i}^v(i)$), where $k_i \leq n_i$, then*

$$v \in \{b|\ b = \underset{a}{\text{argmin}}(\sum_{A_i \in S(v), i \geq h} preference_{k_i}^a(i))\}$$

and no agent A_i, $i>0$, wants to reveal a constraint c_j, $j>k_i$. The feasibility condition is $\sum_{A_i \in S(v)} preference^a_{k_i}(i) \leq 0$.

The feasibility condition verifies that the solution is acceptable to the *initiators*. If v is a solution of a DyDisCSP, then $S(v)$ is the *solver set* for v.

3 Extending AASR

In this section we introduce Secure Asynchronous Search (SAS) which is an adaptation of Asynchronous Aggregation Search with Reordering (AASR) to the DyDisCSP framework. First we recall the basic elements of AASR [4].

Definition 3 (Aggregate) *An aggregate is a triplet (v, s, h) where v is a variable, s a set of values for v and h a history of the pair (v, s).*

A history h for an aggregate $a = (v, s, h)$ proposed by an agent A_k takes the form of a list of pairs $|i : l|$ where i is the index of an ancestor of h that has made a proposal on v and l is the value of a counter. An aggregate requests higher priority agents to comply with a proposal, therefore it defines by itself a nogood. Such nogoods are called **nogoods entailed by the view**.

Definition 4 *An explicit nogood has the form $\neg V$. V is a list of aggregates.*

The agents communicate via: **ok**, **nogood**, **add-link** and **reorder** messages. **ok** messages are sent from agent A_j to agent A_i, and have as parameter a list of aggregates for variables in which A_i is interested. **nogood** messages have as parameter an explicit nogood. **add-link** messages are sent from agent A_j to agent A_i, informing A_i that A_j is interested in a set of variables, *vars*. The agent on position i is denoted A^i. R^i is the agent that can reorder A^{i+1} by sending **reorder** messages.[4]

3.1 Legal messages

In AASR, both **ok** and **nogood** messages transport some kind of nogoods. These are the nogoods entailed by the view, respectively the explicit nogoods. In order to allow the agents detect messages that are potentially harmful for the quality of the computed solution, we introduce the notions of legal nogood and legal aggregate. We want to prevent the agents from disturbing the search by generating illegal messages. A message is illegal if it is generated by an *inactive* agent. SAS requests agents to build messages in such a way that their lawfulness can be proved.

Definition 5 (Legal explicit nogood) *Any legal explicit nogood generated by an agent A_i, where A_i is not an* initiator*, must contain at least one* aggregate *(v_j, s, h), $v_j \in V(i)$ such that s does not contain F.*

Definition 6 (Justification) *Each aggregate I_i generated by an agent A_i that is not* initiator *needs a* justification. *The* justification *of the aggregate I_i consists of a pair (v,h) built from an aggregate (v,s,h) that activates A_i.*

The *justification* of an *aggregate*, a, corresponds to a relaxation of the nogood entailed by the view given by a and is stored in the history of the *aggregate*, attached to the pair corresponding to the agent that has generated a. A history has now the form $|i_1,l_1,j_1|i_2,l_2,j_2|...$ where i_k is the index of an agent, l_k is the value of an instantiation counter and j_k is the justification of the corresponding instantiation.

Property 1 *The space needed by an agent to store all the aggregates is O(nv), where n is the number of agents and v is the number of variables.*

Corolary 1 *The size of an aggregate is O(nv).*

Property 2 *SAS has polynomial space complexity in each agent.*

The proofs are given in [3]. Besides generating illegal nogoods, the agents can also generate illegal aggregates against their competitors.

Definition 7 (Legal aggregate) *An aggregate is legal if its justification is valid and the variable in the justification does not contain F in its instantiation. By convention, any aggregate generated by an initiator is legal.*

3.2 The SAS protocol

In SAS the messages must prove that their sender is active. Agents must generate only legal nogoods. Any other nogood would be discarded. The next rule shows how legal nogoods can be obtained.[a]

Rule 1 (Nogood generation) *An agent A_i may compute an explicit nogood N that is not legal, but the set in the newest aggregate it has received for some variable v_j from $V(i)$ does not contain F. A_i should add the newest aggregate of v_j to N. If this is not possible, it should refrain from sending N to other agents. This rule does not apply to initiators.*

Rule 2 (Checking) *The receiver of an explicit nogood $\neg N$ should check that $\neg N$ is legal. Also the receiver of any* aggregate*, should check that the new aggregate is legal. Illegal information is discarded.*

The *justifications* trigger **add-link** messages in the same conditions as the aggregates received in an explicit nogood in AASR. Justified nogoods should not be delivered to the agent and integrated in the other structures inherited from AASR before the answer to eventual **add-link** messages is received.

[a] When illegal nogoods are made legal, they are in fact relaxed. Agents that must relax nogoods can use heuristics for choosing the variable v_j from $V(i)$. (e.g. choosing the variable for which the known aggregate was generated by an agent with the lowest position.)

Rule 3 (Justification change) *Whenever the* justification *of an agent A_i is modified, A_i has to send again all its aggregates.*

Rule 4 (Justification invalidation) *Whenever the* justification J *of a stored* aggregate a_1 *in A_i is invalidated by some incoming new* aggregate a_2, *A_i has to invalidate a_1 and has to apply again this* invalidation rule *as if a new* aggregate *of the variable in a_1 would have been received.*

Each proposal that activates or inactivates agents is broadcast to all agents with higher positions.

Rule 5 (Next active) *If possible, acting for R^i, each A^i, proposes new orders to ensure that A^{i+1} is active. R^0 is an initiator.*

Proposition 1 *The Secure Asynchronous Search maintains the characteristics of completeness, correctness, and termination of AASR.*

SAS is an asynchronous protocol. A corresponding synchronous protocol (SSS) can be obtained in an obvious way.

4 Conclusions

We present an approach to distributed problems with competition and byzantine-failures. The concept of Dynamic Distributed Constraint Satisfaction is proposed and we show how it allows for modeling complex characteristics of such problems. As shown in [3], DyDisCSPs can easily model and solve (Generalized) English Auctions. The presented algorithms and framework inherit from Constraint Reasoning generality and flexibility in modeling.

References

1. K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *CP,LNCS 1330*, pages 222–236, 97.
2. T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *IJCAI*, pages 631–637, 95.
3. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Generalized English Auctions by relaxations in DyDisCSPs with private constraints. TR #01/365, EPFL, 2001.
4. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. ABT wih asynchronous reordering. IAT, 2001.