Chapter 17

Generalized English Auctions — Secure Verification of Conflicts — Dynamic Distributed CSPs (DyDisCSP)

And I saw that all labor and
all achievement spring from
man's envy of his neighbor.
Solomon, Ecclesiastes 4:4, NIV

We have seen how distributed DisCSPs with openness and privacy can be solved. The legitimate question is: Is this sufficient for solving real distributed problems. This clearly depends on the problem. In this chapter we analyze the case of: *Generalized English Auctions*.

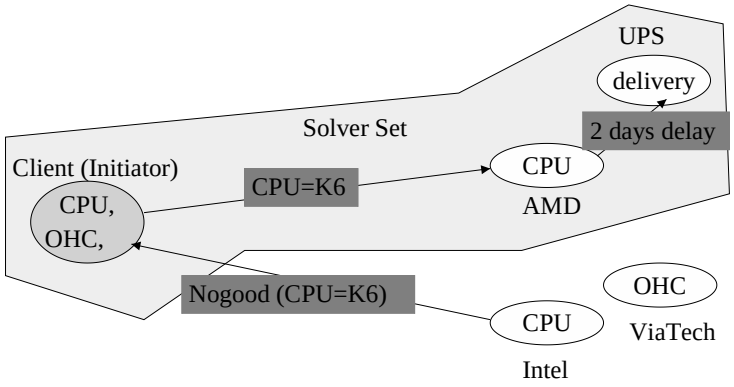


Figure 17.1: In auctions, not everybody has to be satisfied. A new framework is required, to avoid the need to ask Intel^{TM} permission to make a transaction with AMD^{TM}.

Generalized English Auctions are generalizations of English Auctions and combinatorial auctions with multiple participants. It is a generalization first defined in (Silaghi *et al.* 2001c). Some of the properties of GEAs are described in Annex C.

Example 17.22 *In Figure 17.1 I show an illustrative example of the reason why DisCSPs and Valued DisCSPs are not sufficient to model English Auctions. A first intuition is that in the aforementioned frameworks, all the participants have to agree to the solution.*

Imagine that a client wants to build a computer and starts to look on the Internet for finding compatible components at a good price. This process can be seen as a Generalized English Auction

where the client is the auctioneer. Each bidder, once participating in an active partial solution, can dynamically become auctioneer in her turn for satisfying her other eventual requirements.

In Figure 17.1 our client starts to look on the Internet for the components he needs: e.g. a central processing unit (CPU), an open host controller, etc.

From yellow pages, our client finds that Intel and AMD can provide her with CPUs. The client first asks AMD for the transaction of a CPU K6. Before answering, AMD becomes an auctioneer himself, looks up on yellow pages for a mail agent, and checks first with UPS to verify that a delivery is possible in 2 days at a low cost. This may take some time, since UPS personnel may be on strike and a transaction should wait for the end of the strike negotiations.

Imagine now that somebody from Intel finds out somehow about this negotiation (e.g. somebody from AMD speaks too much), and Intel sends a malicious **nogood** *message to the client to announce a false conflict (e.g.* **nogood***(CPU=K6)). In general, with existing asynchronous techniques, there is no known solution to allow the client to detect the fact that a received conflict is not real (we will say "legal" for a real conflict).*

The set of active agents from whom a received message should be taken into account at a certain moment is called the Solver Set*. The Solver Set is changing dynamically and asynchronously (there is no centralizable snapshot of it).*

An auctioneer (client) can send proposals in parallel to several competitors (see (Silaghi & Faltings 2001)*), in distinct computation slots.*

Example 17.23 *As mentioned in (Goldwasser* et al. *1985), the submission of an article to a conference like STOC is a similar auction where techniques for detecting conflict legality and zero-knowledge privacy protection are of important future interest.*

The contributions of this chapter are:

- the idea of Generalized English Auctions (GEAs) (it seems that our mention and formalization of GEAs[1], was the first one).
- definitions of legal conflicts in GEAs.
- a framework and an ABTR-based technique allowing for a secure verification of the legality of received claimed conflicts.
- a (first) asynchronous Branch&Bound optimization technique based on AAS. This is a historical technique and more complex ones have already been described in Chapter 14.
- a technique for the incremental involvement of new participants to a search process.
- a relaxation technique for problems with private constraints, allowing the reuse of some stored conflicts.

In the following we describe the needed GEA elements that are absent in previous protocols for DisCSPs. We also describe some obvious as well as some less obvious solutions.

The techniques of this chapter were first submitted to AA'2001.

17.1 What to satisfy?

In Distributed Constraint Satisfaction, a solution is a valuation that satisfies all the constraints of all agents. The privacy requirements introduced in previous chapters define a solution as a valuation having the agreement of all the agents (Definition 16.4).

This does not extend to Generalized (English) Auction Problems. In auctions, not all participants have to agree with the solution, but only the auctioneer and the winning participants. The use of a DisCSP protocol can then enable a loser to refuse a solution of the auction. I show why this is expected to reduce the Social Welfare. We clearly need a framework for correctly modeling the solution of the new type of problems.

[1] submitted to AA'2001

Two alternative new definitions of solutions have been introduced in 2001 (Silaghi *et al.* 2001c; Felfernig *et al.* 2001). (Felfernig *et al.* 2001) defines distributed problems (DisDyCSPs) as distributions of dynamic CSPs. A variable only has to be instantiated in the solution of a DisDyCSP if its conditional variable is instantiated to true. We propose in (Silaghi *et al.* 2001c) a new formalism where agents (rather than variables) can be excluded from solutions. This framework is called Dynamic Distributed Valued CSPs (DyDisVCSP), in its most detailed form when it is based on Valued CSPs (Schiex *et al.* 1995). The framework could however be defined with basic CSPs, resulting in Dynamic Distributed CSPs (DyDisCSPs).[2]

17.2 Negotiation framework (DyDisCSP)

My intention has never been to define Dynamic Distributed CSPs, but to explain how to make negotiations. It happens that after I have written the content of the following concepts I was suggested to name the new framework DisDyCSP, perhaps due to a related remark made at that time in a presentation of Christian Bessière.

That public remark of Christian Bessière may be the reason of the large number of CSP frameworks with distribution and dynamism that appeared recently.

In order to model practical negotiation problems, we introduce a formalism that enriches the DisCSP framework with dynamism (of participation of involved agents to the Solver Set), preferences and constraint relaxation. The extended framework builds on the notion of Valued CSPs (Schiex *et al.* 1995). First we describe the problem of an agent, A_u, as a Negotiation Valued CSP, (NVCSP$_u$).

17.2.1 Negotiation Valued CSP

Here I describe 2 (out of 4) incrementally simplified definitions of a Negotiation Valued CSP. The first (complex and heavy) definition of NVCSP$_u$ is the one in (Silaghi *et al.* 2001c).

Definition 17.1 (Cost) *The problem of an agent A_u is parametrized by the minimal sum of money that A_u could ever rationally admit to receive*[3] *for agreeing on a certain valuation v of its variables. This sum of money is called cost, and is denoted $cost^v(u)$.*

An intuitive explanation is that the NVCSP of agent A_u consists of a series of VCSPs, that are incremental relaxations. Each relaxation is obtained by either making some infeasible valuations feasible, or by reducing the price that A_u requests for some feasible valuations with at least the minimal increment ε.

Definition 17.2 (Price) *The problem of an agent A_u is parametrized by the sum of money that A_u requests at time i for agreeing on a certain valuation v of its variables. This sum of money is called price, and is denoted $price_i^v(u)$.*

In the next definition, the redundant separate indication of the feasibility (used in (Silaghi *et al.* 2001k)) is replaced by interpreting infinite prices as infeasibility.

Definition 17.3 *NVCSP'$_u$ consists of:*

- *A minimal increment, ε.*
- *A set of public variables, V(u). The domain of each variable contains a value*[4] *F.*
- *An ordered set of global constraints $c_1(u),\ldots,c_{n_u}(u)$.*

[2] Another framework called DyDCSP has been proposed shortly after us by Modi et.al'01, but I find that the choice of the name of their framework does not fit their defined framework which is rather a static distribution of Dynamic CSPs. However, if one would slightly modify their framework such that a variable would move from an agent to another, the obtained framework could be considered a natural and new type of dynamic distributed CSP.

[3] utility 0

[4] When the variable stands for the allocation of a resource, F means *unchanged* and *indifferent*

- *Each pair (valuation v, constraint $c_i(u)$) has associated a price, $price_i^v(u)$.*

$price_i^v(u)$ is such that for each constraint $c_i(u)$, $\exists cost^v(u)$, $price_i^v(u) \geq cost^v(u)$ and if $n_u \geq i > j > 0$ then:

- *for any valuation v, $price_i^v(u) \leq price_j^v(u)$,*
- *there exists a valuation v such that $price_i^v(u) + \varepsilon \leq price_j^v(u)$*

Definition 17.4 (Preference) *The problem of an agent A_u is optionally parametrized by a symbolic value that A_u publicly associates at time i to a valuation v of its variables. This symbolic value is called preference, and is denoted $preference_i^v(u)$.*

As first done in (Silaghi *et al.* 2001c), the framework can be designed with *preferences*, a supplementary evaluation criteria that is less important than the prices. The corresponding definition with preferences is:

Definition 17.5 *NVCSP"$_u$ consists of:*

- *A minimal increment, ε.*
- *A set of public variables, V(u). The domain of each variable contains a value F.*[5]
- *An ordered set of global constraints $c_1(u),\ldots,c_{n_u}(u)$.*
- *Each pair (valuation v, constraint $c_i(u)$) has associated a tuple:*

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

$T_i^v(u)$ is such that for each constraint $c_i(u)$, $\exists cost^v(u)$, $price_i^v(u) \geq cost^v(u)$ and if $n_u \geq i > j > 0$ then:

- *for any valuation v, $price_i^v(u) \leq price_j^v(u)$,*
- *there exists a valuation v such that $price_i^v(u) + \varepsilon \leq price_j^v(u)$*

17.2.2 Negotiation Problem (aka. Dynamic DisCSP)

There exist several types of dynamism in a distributed (valued) CSPs.

- dynamism isolated in local problems of the agents (Modi *et al.* 2001)
- dynamism in moving sub-problems from one agent to another (Jung *et al.* 2000)
- dynamism in common agreement on activation of shared variables (Felfernig *et al.* 2001)
- dynamism in the composition of the set of agents that have to agree on a solution (this chapter) (Silaghi *et al.* 2001c)
- dynamism in the involvement of agents in the process (see Chapter 19) (Silaghi & Faltings 2002a)
- joint dynamism in the involvement of agents and variables (see Section 17.9) (Silaghi *et al.* 2001d; 2001c)
- dynamism through relaxation:
  - relaxation of constraint tuples (see Section 17.10) (Silaghi *et al.* 2000g)
  - joint relaxation of variables and values (see Chapter 18) (Felfernig *et al.* 2002; Zanker *et al.* 2002)
  - relaxation of domains (co-work with Macho-Gonzales and Faltings in Section 17.10.3)

[5] When the variable stands for the allocation of a resource, F means *unchanged* and *indifferent*.

– homogeneous relaxation performed simultaneously in all agents (Yokoo and Hirayama)

It would be a valuable gain to the field to have an agreement with other researchers on a non-conflicting terminology for the different approaches, as well as for future approaches and hybrids. The negotiation framework that I propose is defined next:

Definition 17.6 (Generalized Auction Problem (GAP) (aka. DyDisCSP)) *A Generalized Auction Problem (GAP) aka. Dynamic DisCSP*[6] *(DyDisCSP) is defined by:*

- *A set of agents* A_1,...,A_n. $A_k, k \in [1,h], n \geq h \geq 1$, *are* h *agents called* initiators.
- *Each agent* A_j *owns a NVCSP,* $NVCSP_j$.
- *Each agent* A_j *is interested in a set of public variables V(j).*
- $cost^v(j)$ *is private to the agent* A_j *for any valuation* v.

The **initiators** are the agents whose gain is maximized. The client in the example at the beginning of the chapter is the initiator of the corresponding auction.

Definition 17.7 ($S(v)$) *Given a valuation v for all the public variables,* $S(v)$ *is the set of agents owning a variable not instantiated in* v *to* F. *By convention, the* initiators *also always belong to* $S(v)$.

Intuitively, $S(v)$ is the set of agents that have to agree to the valuation v, in order for v to be a solution.

Definition 17.8 (Acceptable valuation) *A valuation* v *is acceptable if each agent* A_i *in* $S(v)$ *proposes a feasible tuple for the projection of* v *on V(i).*

I do not want to mix the current auction with other independent negotiations. This is formalized by the next definition.

By $\breve{v}$ we denote a valuation obtained from the valuation v by reassigning a subset of variables to F such that $v \neq \breve{v}$.

Definition 17.9 (Stable valuation) v *is stable if there exists no acceptable* $\breve{v}$.

Intuitively, a *stable valuation* is minimal in the sense that it corresponds to an agreement of the agents in $S(v)$, and by eliminating any subset of agents from $S(v)$, no agreement can be obtained with the *initiators*.

Definition 17.10 (Active) *An agent* A_i *is active either if* A_i *is an initiator, or recursively, if an agent that is active proposes a valid instantiation outside* F *of a public variable of* A_i.

17.2.3 Solution of a Generalized Auction Problem (GAP) (aka. DyDisCSP)

Definition 17.11 (Solution) *A solution of a GAP is a stable acceptable valuation* v *of all the public variables.*

A solution is optimal if it mimimizes the total sum of money paid by initiators (auctioneers) to the rest of the participants, minus their **requested gain**.

The sum of money paid by initiators (auctioneers) to the rest of the participants for a solution a is $\sum_{A_i \in S(a), i>h} price^a_{k_i}(i)$.

The **requested gain** of the initiators in a solution a is $\sum_{i \leq h}(cost^a(i) - price^a_{k_i}(i))$.

[6] We propose to call DyDisCSP a DisCSP where the participation of agents is dynamic. An alternative is to call this framework dynamic distributed valued CSP (DyDisVCSP). A DyDisCSP where the agents own dynamic CSPs can then be called dynamic distributed dynamic CSP (DyDisDyCSP).

Definition 17.12 (Optimal Solution) *Given the set* Γ *of all solutions of a DyDisCSP, and the set*

$$\Gamma^* = \{b|\ b = \operatorname*{argmin}_{a\in\Gamma}((\sum_{A_i\in S(a), i>h} price^a_{k_i}(i)) + (\sum_{i\leq h}(price^a_{k_i}(i) - cost^a_{k_i}(i))))\},$$

a solution v *is optimal when* $v\in\Gamma^*$*, and no agent* A_i*,* $i>0$*, wants to reveal a constraint* c_j*,* $j>k_i$*.*

The feasibility condition is $\sum_{A_i\in S(v)} price^v_{k_i}(i) \leq 0$*.*

The feasibility condition verifies that the solution is acceptable to the *initiators* (they accept to pay the solution).

If v is a solution of a DyDisCSP, then $S(v)$ is the *solver set* for v.

When there are several initiators, their costs may be private from one another. In consequence, the gain of the initiators cannot be estimated and we have to define another type of solutions. I propose to estimate that $price^a_{k_i}(i) = cost^a(i), \forall i\leq h$.

Definition 17.13 (Public Optimal Solution) *Given the set* Γ *of all solutions of a DyDisCSP, and the set*

$$\Gamma^*_p = \{b|\ b = \operatorname*{argmin}_{a\in\Gamma}(\sum_{A_i\in S(a), i>h} price^a_{k_i}(i))\},$$

a solution v *is public optimal when* $v\in\Gamma^*_p$*, and no agent* A_i*,* $i>0$*, wants to reveal a constraint* c_j*,* $j>k_i$*.*

The feasibility condition is $\sum_{A_i\in S(v)} price^v_{k_i}(i) \leq 0$*.*

Especially when the costs are public, it becomes interesting to use at least a symbolic estimation of the preference of a participant for different (public) optimal solution. The preferences define a criteria of pareto-optimality over optimal (public) solutions.

Definition 17.14 (Optimal Preferred Solution) *Given the set* Γ^* *of all (public) optimal solutions of a DyDisCSP,* v *is an optimal prefered (public) solution if* v *is pareto-optimal for initiators (and eventually for* $S(v)$*) over* Γ^**.*

17.2.4 Generalized English Auctions for GAPs

In our framework, the bidding step for solving a DyDisCSP, P, amounts to solving a DisCSP obtained from P where the k_i of any A_i is fixed.

The auctions enabled by our approach to GEA (DyDisCSP) are a kind of multi-unit combinatorial exchanges where the final solution has to get the agreement of a predefined (sub)set of agents (the *initiators*). We therefore call such auction problems *Multi-Unit Supervised Combinatorial Exchanges (MUSCEWDP).*

Definition 17.15 (MUSCEWDP) *MUSCEWDP are Multi-Unit Combinatorial Exchanges winner determination problems where the solution needs the agreement of a predefined set of agents.*

17.3 Modeling Auctions

The Generalized Auction Problems can be used to model Auctions. We draw now a parallel between the introduced framework and typical (English) Auctions. The equivalence of notions is:

- public variable ⇔ transaction for the allocation of a good to an agent, other than the current owner.

  Each variable is in the NVCSPs of the current owner and of the target owner of the good.

  values $\in \{T, F\}$, showing if the transaction is chosen.

- initiator ⇔ auctioneer.

- price $\Leftrightarrow$ minus of the bids for a combination of allocations

  The cost in DyDisCSPs is the minus of the worth of the outcome in English Auctions.

- price $-$ cost $\Leftrightarrow$ utility or worth.

The initiator launches the search in the space of allocations. At each step, an agent A_u imposes the constraint $c_{k_u}(u), k_u \leq n_u$. A relaxation of the imposed constraints corresponds to increasing k_u.

17.4 Classic Framework for Generalized (English) Auctions

While by now it should be clear that the GAP framework introduced so far generalizes the classic Auctions Problems. However, for inertial specialists of English Auctions, the new notions should be difficult to learn. I therefore rephrase the framework with the standard notions for English Auctions.

It is worthy to notice that the only modification performed in the following definition is to replace *cost* with $-$*worth_outcome*, price with $-$*bid*, and to explicit the domains. A classic negotiation valued CSP of an agent A_u in a GAP is:

Definition 17.16 *NVCSP'$_u$ with binary domains consists of:*

- *A minimal increment, ε.*
- *A set of public variables representing tranzactions, V(u). The domain of each variable contains a value T for performing the tranzaction and a value F for not performing the tranzaction.*
- *An ordered set of global constraints $c_1(u),\ldots,c_{n_u}(u)$.*
- *Each pair (valuation v, constraint $c_i(u)$) has associated a bid, $bid_i^v(u)$.*

$bid_i^v(u)$ is such that for each constraint $c_i(u)$, $\exists worth\_outcome^v(u)$, $bid_i^v(u) \leq worth\_outcome^v(u)$ and if $n_u \geq i > j > 0$ then:

- *for any valuation v, $bid_i^v(u) \geq bid_j^v(u)$,*
- *there exists a valuation v such that $bid_i^v(u) - \varepsilon \geq bid_j^v(u)$*

A classic framework for GAPs is obtained directly by simply replacing *cost* with *worth_outcome* and initiators with auctioneers.

Definition 17.17 (Classic Generalized Auction Problem) *A Classic Generalized Auction Problem (CGAP) is defined by:*

- *A set of agents $A_1,\ldots,A_n$. $A_k, k \in [1,h], n \geq h \geq 1$, are h agents called* auctioneers.
- *Each agent A_j owns a NVCSP, $NVCSP_j$.*
- *Each agent A_j is interested in a set of public variables V(j).*
- *$worth\_outcome^v(j)$ is private to the agent A_j for any valuation v.*

The remaining important definition is the one for Optimal Solution. $worth^a(i)$ is the reservation price of a for auctioneer A_i.

Definition 17.18 (Optimal Solution) *Given the set Γ of all solutions of a CGAP, and the set*

$$\Gamma^* = \{b |\ b = \operatorname*{argmax}_{a \in \Gamma}((\sum_{A_i \in S(a), i>0} bid_{k_i}^a(i)) - (\sum_{i \leq h}(worth^a(i))))\},$$

a solution v is optimal when $v \in \Gamma^*$, and no agent A_i, $i>0$, wants to reveal a constraint c_j, $j > k_i$.

The feasibility condition is $\sum_{A_i \in S(v)} price_{k_i}^v(i) \leq 0$.

17.5 Illegal Nogoods

..., and ye shall be kept in prison,
that your words may be proved, whether
there be any *truth in you: or else*
by the life of Pharaoh surely ye are *spies.*
Josef, Genesis 42:16, KJV

Once GAPs are modeled as DyDisCSPs, we have the problem of adapting DisCSP protocols for GEAs. The candidate we discuss here is AASR with the convention $R^k \equiv A^k$. When used in contentional situations (e.g. within negotiation), the AASR technique is no longer appropriate. The reason is that a solution of such problems does not need to be an acceptable solution for all the agents, as long as some of them are not **active** in the solution and do not gain anything[7].

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 assignment. We want to prevent the agents from disturbing the search by generating illegal messages. A message (containing a nogood $\neg N$) is illegal if it is generated by an agent that can be inactive in some valuation extending a partial valuation in the Cartesian-product defined by N. The new protocol is called Secure Asynchronous Search (SAS). SAS requests agents to build messages in such a way that their lawfulness can be proved.

Definition 17.19 (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* assignment *of a variable v from $V(i)$ such that v does not contain F.*

17.6 Illegal Proposals

Definition 17.20 (Justification) *Each assignment I_i generated by an agent A_i that is not* initiator *needs a* justification*. The* justification *of the assignment I_i consists of a pair (v,h) built from an assignment $\langle v, s, h\rangle$ that activates A_i.*

The *justification* of an *assignment*, a, corresponds to a relaxation of the nogood entailed by the view given by a and is stored in the signature of the *assignment*, attached to the pair corresponding to the agent that has generated a. A signature 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.

Theorem 17.1 *The space needed by an agent to store all the assignments is O(nv), where n is the number of agents and v is the number of variables.*

Proof. The number of possible simultaneous valid assignments is nv since each agent can generate at most v valid assignments at a time (one per variable). All assignments and justifications can be represented as a directed graph having the valid assignments as nodes. The maximum number of arcs in this graph is $2nv$ since there cannot be more than 2 arcs getting out of a node. □

Corollary 17.1.1 *The size of an assignment is O(nv).*

Property 17.2 *SAS has polynomial space complexity in each agent.*

Proof. AASR requires polynomial space and the only additional structures required by SAS consist of the new assignments in justifications. For all the references to assignments in the structures of AAS, Corolary 17.1.1 shows that a polynomial mapping exists to the new form of the assignments. □

Besides generating illegal nogoods, the agents can also generate illegal assignments against their competitors.

[7] And have to reduce their preferences.

Definition 17.21 (Legal assignment) *An assignment is legal if its justification is valid and the variable in the justification does not contain F in its instantiation. By convention, any assignment generated by an initiator is legal.*

No aggregate $\langle v, s, h\rangle$ generated in SAS may aggregate in s both the value F and some other values.

17.7 Secure Search

In order to enable agents to make proposals, they must be given the opportunity to know when they are activated. The active/inactive state of an agent is know when either one of its public variables is instantiated outside F (active), or when all its public variables are instantiated with F (inactive). For the security of the search, we want to involve on low positions in search only agents that are known to be active.

Rule 9 (Initiator first) *The agent A^1 has to be an* initiator*.*

Rule 10 (Active first) *Whenever possible, each agent proposes orderings to make sure that the agent on the next position is known to be active.*

In order to let agents know which of the next agents are active, a brute force solution is that active agents must announce all their instantiations for public variables to all their successors.

Since 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.

Rule 11 (Nogood generation) *Whenever an agent A_i computes an explicit nogood N that is not legal, and the set in the newest assignment it has received for some variable v_j from $V(i)$ does not contain F, it should add the newest assignment of v_j to N.*[8] *If this is not possible, it means that A_i is inactive and it should refrain from sending N to other agents. This rule does not apply to initiators.*

Coalitions can still be created in SAS. In fact any agent that does not check if a receiving message is valid makes a (temporal) coalition with the sender.

Rule 12 (Checking) *The receiver of an explicit nogood N should check that N is legal. Also the receiver of any* assignment*, (when an* **ok?** *message is received), should check that the new assignment is legal and the assignment is not empty (the search cannot be voluntarily blocked).*

If one of these conditions is not respected, the messages must be discarded.

In order to ensure completeness and termination of SAS, the management of justifications has to be coherent. The *justifications* trigger **add-link** messages in the same conditions as the assignments received in an explicit nogood in AAS. Moreover, justified nogoods should not be delivered to the receiving agent and integrated in the other structures inherited from AASR before the answer to eventual **add-link** messages is received.

Rule 13 (Justification change) *Whenever the* justification *of an agent A_i is modified, A_i has to send again all its assignments.*

Only one assignment is used as justification by an agent at a time.

Rule 14 (Stored agent-view) *Each agent stores all the valid assignments it receives.*

[8] 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 assignment was generated by an agent with the lowest priority.)

Rule 15 (Justification invalidation) *Whenever the* justification J *of a stored* assignment a_1 *in* A_i *is invalidated by some incoming new* assignment a_2, A_i *has to invalidate* a_1 *and has to apply again this* invalidation rule *as if a new* assignment *of the variable in* a_1 *would have been received.*

Lemma 17.1 *The information maintained by agents in SAS is consistent with their predecessors.*

Proof. The rules for nogoods generation and nogood checking are allowed by the AAS framework and the proof in (Silaghi *et al.* 2000a) remains valid. Sending any (finite) number of **add-link** messages when justifications are received fits AAS as well. It remains to prove that the properties are maintained when the invalidation of a *justification*, j, triggers the invalidation of an *assignment*, x, that it justifies.

The role of a stored *assignment* in AAS is to define a valid *nogood entailed by the view* of the agent. Therefore x defines a *nogood entailed by the view.* In the SAS, the agent activated by j has to invalidate and recompute its instantiation (due to the justification change rule). Therefore x, becomes indeed invalid. Consequently, in all cases the information maintained by the agents will eventually be consistent with their predecessors. □

Definition 17.22 (Quiescence) *By quiescence of a group of agents we mean that none of them will receive or generate any legal valid nogoods, new legal valid assignments,* **reorder** *messages or* **add-link** *messages.*

Lemma 17.2 *At quiescence for the agents* $A^1, ..., A^i$*, an agent* A^i*, knows whether there exist any active agent placed after it or not.*

Proof. If any initiator is placed after A^i, this is known to A^i since the last ordering was received.
Otherwise, A^i knows all assignments outside F done by itself or predecessors for public variables. A^i also knows the variables that can activate its successors since they are announced as initial data. A^i therefore knows when any successor is activated by the agents $A^1, ..., A^i$.
If no successor is activated by $A^1, ..., A^i$ and no successor is initiator then, since an inactive agent cannot activate another agent, no successor of A^i is active. □

Lemma 17.3 *In SAS, let a set of* k *agents reach quiescence in finite time* t^k *and at quiescence they take positions* $1, ..., k$ *where each of them agrees with its predecessors. Then, if some of the other agents are active, either failure is detected or an active agent will reach quiescence on position* $k+1$ *in finite time after* t^k *having an instantiation that agrees with its predecessors.*

Proof. The number of possible **add-link** messages is bounded so we do not need to consider them.

Let τ be the maximum delay for delivering a message. After $t^k+\tau$, all the agents know the instantiations of $A^i, ..., A^k$. Within time $t_h^k = t^k + \tau + t_h + \tau$, no **heuristic** message is received any longer by the agents $R^0, ..., R^k$. The final order that decides the agent taking the position A^{k+1} is therefore issued before $t_h^k + t_r$ and if any successor is active, an active agent becomes A^{k+1}. After $t_h^k + t_r + \tau$, the identity of A^{k+1} no longer changes and its view remains stable after $t_o, t_o < t_h^k + t_r + \tau$. Since the domains are finite, in finite time after t_o either some proposal of A^{k+1} is no longer refused, or all its proposals are refused and A^{k+1} infers at time $t_n, t_n > t_o$ a legal valid nogood $\neg N$.
Since $\neg N$ is valid and legal, either:
I. N it is empty and the search fails, or
II. when $t_o < t_k$, its receiver changes its instantiation for a variable, v, and either it or its predecessors announce A^{k+1} of the new assignment for v. But such an announcement arrives at A^{k+1} after $t_n + 2\tau > t_o$. This contradicts the hypothesis that the view is stable after t_o and that $t_o < t_n$.
III. The predecessor receiving $\neg N$ detects failure and the search ends.

Therefore in all possible cases the lema holds. □

Lemma 17.4 *In SAS, the active agents reach quiescence on consecutive positions* 1 *to* k*, and all inactive agents are placed after* A^k*.*

Proof. Due to the lemma 17.1, each agent knows when some successor is active and according to the rule 10 (Active first), the active agents are ordered consecutively. □

The solutions computed by SAS correspond to an agreement among the tuples revealed by the active agents. This agreement consists in the instantiation of the shared public variables to the same value. The optimality of the prices has to be checked separately by the broker.

Proposition 17.3 *The SAS is complete, correct, and terminates. The solutions it computes correspond to an agreement on the tuples revealed by the active agents.*

Proof. **Quiescence of the active agents:** is a result of lemma 17.3. In bounded time, the active agents find a solution and reach quiescence letting the broker to detect it, or a failure is detected.

Completeness: All the nogoods are generated by inference and therefore no empty nogood is generated if a solution exists. An erroneous inference made by an agent A_i cannot eliminate a solution that does not instantiate outside F a variable activating the agent A_i. Therefore those solutions needed the acceptance of A_i.

Correctness: The assignments that activate the active agents at quiescence and the other variables set by these agents to values that do not contain F is a Cartesian-product C. Each element in C defines a **stable valuation** v when it is expanded by instantiating any other public variables to F. The active agents correspond to $S(v)$. They all know the instantiations of their predecessors and they can generate legal valid nogoods since they are active. Since at quiescence the active agents agree with their predecessors (Lemma 17.3), then all the agents in $S(v)$ agree on v. □

For the SAS, the termination detection algorithms from AAS can no longer be used since the agents are not trusted. The messages for announcing the solution may be sent via inactive agents and therefore they can be corrupted. For the general case, the *solution* can be detected by the *system agent*. Each agent has to send its solution to the *system agent*. The *system agent* has to find the *solver-sets*, namely all the subsets of agents agreeing with the initiators that are consistent and together define a *solution*. This is a problem with polynomial complexity. For some real problems, alternative techniques for *solution* detection can be used.

17.8 Branch and Bound

As described in Annexes, Branch and Bound is an optimization technique that consists in continuously computing a better bound on the quality of a partial valuation. Current partial valuations that are worse than the current bound are trimmed.

Branch and Bound can also be used dynamically during the search. In Section 7 of (Silaghi *et al.* 2001d) we have proposed a distributed Branch and Bound algorithm based on AAS. We further refer to it as AASBB.

First, whenever a new proposal is made by an agent, it has to generate aggregates for all its external variables. The values (prices and preferences) for all the tuples in the sent proposal have to be attached to the values in the Cartesian product of the proposal.

Example 17.24 *A proposal*

$$\mathbf{ok}?(\langle x_i, \{1,2\}, |1,3|\rangle, \langle x_j, \{3,5\}, |1,3|\rangle),$$

in AAS, will become

$$\mathbf{ok}?(\langle x_i, \{1,2\}, |1,3|\rangle, \langle x_j, \{3,5\}, |1,3|\rangle, \{5,5,5,5\}, \{3,3,3,3\}),$$

in AASBB. The last parameters of the message are the set of prices, {5,5,5,5}, and the set of preferences, {3,3,3,3}, attached to each element of the Cartesian product of the sets of values of the aggregates in the proposal.

If aggregations are done in such a way that all values in an assignment have equal prices and preferences, then this value needs to be sent only once.

Example 17.25 *The proposal in the previous example becomes*

$$\mathbf{ok}?(\langle x_i, \{1,2\}, |1,3|\rangle, \langle x_j, \{3,5\}, |1,3|\rangle, \{5\}, \{3\}),$$

Whenever a solution is found, its quality (price and preferences) is computed and broadcasted to all agents as a new bound (e.g. using a system agent).

The trimming of Branch and Bound is more informed if each agent sends its proposed aggregates to all the agents with lower priority[9].

Since each agent receives the proposals of all its predecessors, each agent can compute the quality of a partial valuation including its proposal and can renounce to its proposal if it leads to expensive solutions.

Theorem 17.4 *AASBB is complete, correct and terminates.*

Proof. AASBB maintains the properties of AAS as all its messages are in agreement with the AAS protocol (a restriction), with the introduction of new nogoods for the branches cut by the bounds. This nogoods are correct so that the completeness is maintained.

The termination is maintained since the new nogoods (introduced by bounds) are not removed until their cause is removed, so that the enumeration remains the one from AAS. the proof of correctness of AAS does not change. □

Once Branch and Bound has been defined on AAS, it extends straightforwardly to SAS, yielding an algorithm what we will denote SASBB. Note that due to the fact that only messages from active agents are accepted, the trimming in SASBB takes into account only prices and preferences announced by active agents.

17.9 Practical Considerations

Some details have to be addressed for making this approach usable in practice.

17.9.1 Limiting Commitment

An agent proposing a deal to another agent offers a timeout of the offer, beyond which the offer is withdrawn. This timeout should normally be communicated together with a proposal. The agent receiving a proposal can use this deadlines to compute delays it can offer as sub-contractor. Algorithms based on such techniques are no longer complete.

17.9.2 Managing Problem Size

In order to enhance the scalability of SAS, special care must be devoted to managing the size of the problem.

Remark 17.1 *Since the total number of agents in the world, ready to be involved in a negotiation can be huge, we want to maintain the set of involved agents to a minimum, without losing completeness. The current techniques can be adapted to this new requirement.*

Remark 17.2 *In order to reduce computation effort, the agents have the interest to use acceptable value ordering heuristics for guiding the search (see DVR-MAS in Chapter 14).*

Definition 17.23 (MIS) *An agent belongs to the* minimal interesting set *if, during search, at least one partial solution, better than the lowest bound has been found, which requires the participation of this agent.*

Definition 17.24 (Involved agent) *An agent is involved in a current computation if it has been sent at least one proposal for that computation.*

An agent is involved only if it belongs to the *minimum interesting set*. Usually, agents can be added in a natural manner to a computation. In several existing protocols, it is less easy to eliminate agents. We have introduced the notions of *minimum interesting set* and *involved agent* to guarantee that an agent is involved only when this is needed for completeness. For example, in

[9]In the initial version of AAS the priority was defined to have the same value as the position: higher and lower priorities were inversed.

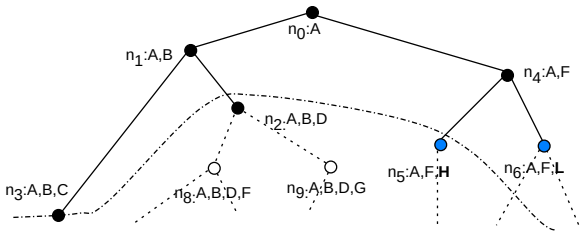


Figure 17.2: The solid circles correspond to visited nodes, up to node n_4. The list of agents active at each of them is shown under the node. The curved dashed line show the lower bound for ESW.

Figure 17.2 the active agents at node n_4 are A and F. The involved agents are A, B, C, D, F, the MIS also contains H and L. G has not been in MIS.

Let us consider that the agents with right to control each external variable can be found in some *yellow pages*. The *first initiator* maintains a set of involved agents, empty in the beginning. Whenever a set of *initiator* agents pose a new problem, they are inserted in the set of involved agents.

Each time that the involved agents reach quiescence, but instantiated variables in the partial solution correspond to the activation of some agents not yet involved in the search, the *first initiator agent* checks that the partial solution built so far has a lower value than the best solution found so far. If this condition is not respected, a nogood is generated for the current valuation. Otherwise, the *first initiator agent* sends them a request to get involved (an **involve** message). The **involve** takes as parameter the current priority of the receiving agent in the search, the set of *initiators*, and the priorities (and information on external variables) for all the other agents involved in the search.

When new agents accept to get involved in the computation, all the existing agents should receive information in a synchronized manner on the priority and the external variables of the new agents via an **involved** message. They will have to send their proposals to the new agents.

If some agents cannot be contacted or refuse to get involved in the search, the *first initiator agent* announces it to the already involved agents by an **update-yellow-pages** message mentioning the name of the removed agents and the variables they control. The alternatives on variables controlled by removed agents have to be disabled from all agents.

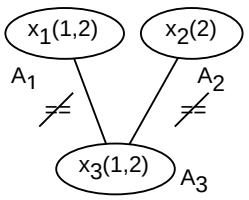
The nogoods can also be reused between rounds by using the technique proposed in the next section (see also (Silaghi *et al.* 2000g)). That technique consists of explaining inferences with references to constraints (CR). The CRs do not necessarily stand for a given constraint, but provides a way to signal when due to relaxations, a nogood is invalidated. The algorithms remain polynomial in space only if the number of CRs in use is bounded. The reuse of CRs can be enabled by attaching to them counters.

17.10 Constraint relaxation for DisCSPs

Constraint relaxation is an important step in negotiations, but it is also an important element in improving efficiency of problem solving. The following subsections address these issues separately. First, I introduce a mechanism for saving nogoods when a relaxation occurs.

17.10.1 Nogood management

We now recall the constraint relaxation schema for private constraints used in (Silaghi *et al.* 2000g). That technique consists of explaining inferences with references to constraints (CR). A related technique has been used for dynamic domain splitting for numeric CSPs (Jussien & Lhomme 1998). To enhance privacy support, the CRs do not necessarily stand one to one for a given constraint, but provide a way to signal when due to relaxations, a nogood is invalidated. Therefore, some constraints and sets of constraints can be associated with a CR.



1: A_1 —**ok?**$\langle x_1, 1, 1\rangle$→ A_3
2: A_2 —**ok?**$\langle x_2, 2, 1\rangle$→ A_3
3: A_3 —**nogood**$\neg(\langle x_1, 1, 1\rangle, \langle x_2, 2, 1\rangle), \{C^{A_3}\}$→ A_2
4: A_2 —**nogood**$\neg(\langle x_1, 1, 1\rangle, \langle x_2, 2, 1\rangle), \{C^{A_2}, C^{A_3}\}$→ A_1
5: A_2 —**add-link**→ A_1
6: A_2 —**ok?**$\langle x_2, 2, 2\rangle$→ A_3
7: A_1 —**ok?**$\langle x_1, 2, 2\rangle$→ A_2
8: A_1 —**ok?**$\langle x_1, 2, 2\rangle$→ A_3

Figure 17.3: Example of ABT with maintenance of CRs.

Example 17.26 *The CSP of A_i, defined by the constraints c_1, c_2, c_3, c_4, c_5, can be associated with CRs from the set (a, b, c, d, e, f, g, h) in the next way:*

c_1, c_2, c_3, c_4, c_5	$\{a\}$
c_1	$\{b, c\}$
c_1, c_3	$\{d\}$
c_2	$\{e\}$
c_5	$\{f, g, h\}$

Each agent is associated with a predefined CR, CR(A_i), which in the previous example is a. At any inference, the nogood resulting from the inference is tagged with the union between:

- CR(A_i), where A_i is the agent making the inference,
- CRs of the constraints used for inference, and
- the union of CRs tagging the nogoods used for inferences.

The algorithms remain polynomial in space only if the number of CRs in use is bounded. The reuse of CRs can be enabled by attaching to them counters. When CRs are obtained from A_i, the versions with lower value of the counters cannot be used any longer, being discarded.

In Figure 17.3 is given an example of ABT with maintenance of CRs. CR(A_i) is denoted by C^{A_i}. The example uses the simple problem exploited in (Yokoo *et al.* 1992) to illustrate ABT. The agent A_3 enforces the constraints $x_1 \neq x_3, x_2 \neq x_3, x_3 \in \{1, 2\}$. As in (Silaghi *et al.* 2000g) each of these constraints can be represented with distinct CRs: $\{C_1^{A_3}, C_2^{A_3}, C_3^{A_3}\}$. For privacy reasons, A_3 can use more than three CRs, having several CRs for the same constraint. The separate relaxation of some of these constraints can be announced by broadcasting a set of CRs representing them. Therefore, each agent discards the nogoods tagged by CRs of relaxed constraints. Currently, a CR with a new counter cannot be generated after constraints previously tagged with it have been relaxed.

For simplicity, each agent in the example of Figure 17.3 uses only one CR. The agent A_3 infers the nogood $\neg(\langle x_1, 1, 1\rangle, \langle x_2, 2, 1\rangle)$ and tags it with C^{A_3}, (CR(A_3)). When the agent A_2 infers $\neg(\langle x_1, 1, 1\rangle, \langle x_2, 2, 1\rangle)$, it tags this nogood with $\{C^{A_2}, C^{A_3}\}$ by adding C^{A_2} due to its constraint: $x_2 \in \{2\}$.

17.10.2 Relaxation for negotiation with private constraints

DisCSPs with private constraints fit well problems involved in negotiations. However, it may happen that these problems are over-constrained or that the agents have different preferences for some solutions. Max-CSPs and Valued-CSPs (Yokoo 1993a) have already been used to tackle this problem in the context of public constraints and honest agents. We now briefly discuss how constraint relaxation can be incorporated into MAS. Since the constraints of each agent are private, an agent can be tempted to create fake or modified constraints in order to mislead the others. In order to give equitable chances to each agent in the case of over-constrained DisCSPs, we must then consider that each agent owns exactly one global constraint.

The relaxation can be achieved following a negotiation schema (Silaghi *et al.* 2000g):

Rule 16 (Relax failure point) *To enhance efficiency, it is logically advisable to add new possibilities for partial valuations where the previous reasoning has failed. Such cases are signaled in asynchronous search by nogoods.*

Whenever the search fails to find a solution, a deadline is given to the agents for the submission of a message containing tuples that the sending agent has rejected as nogood in the previous search and it would accept in a new one. The messages have to be opened simultaneously. If no agent relaxes its constraints, the search fails.

We want to avoid exploring twice the same search space, and this can be easily done in the following way.

Rule 17 (Avoid explored space) *If some agents have submitted nonempty messages, the agents are reordered by giving higher priority to the yielding agents. Search is then performed by cutting the branches containing no new tuple.*

The previous elements describe an algorithm that we will denote lated by MAS-PrivRelax (MAS with Relaxation of Private Constraints). The strategies of MAS-PrivRelax can be followed until failure or a solution is found.

Theorem 17.5 *MAS-PrivRelax is correct, complete, and terminates*

Proof sketch. There cannot be an infinite number of failure points, therefore either a solution is found, or the agents stop from relaxing their constraints. The search space whose exploration is avoided due to the Rule 17 is eliminated by previous inference that remains valid. □

Some problems are posed by this approach. The first one is related to the submission of messages with new tuples. A cryptographic technique may be a solution. The second problem consists in the detection of lying agents, namely agents that submit tuples that were not included in their local search space or not in conflict with their local CSP in the previous search. A trace of all the generated nogoods may be required and a space problem can appear for storage in hard problems. It may be relaxed by checking that the previous agents could have accepted that combination.

17.10.3 Open Constraint Satisfaction

Following an idea of Professor Faltings and with Santiago Macho, we defined open constraint satisfaction, the equivalent framework for relaxation in problems with privacy of domains.

The idea in open constraint satisfaction is to start with tight unary constraints (few values), and to relax (add values) as required by the observed failures. This is a simpler (primal) version of starting with tight constraints and adding tuples.

A previous work where we added values on failure in a CSP, is the DisGCSP technique described in Chapter 18 (Felfernig *et al.* 2002; Zanker *et al.* 2002). The difference is that in (Dis)GCSPs we correlated the addition of new values to the activation of additional variables. The more exact relation between the two techniques remains to be studied in the future.

Definition 17.25 (Open Constraint Satisfaction Problems) *An Open CSP (OCSP), O, is defined as a set of CSPs, $\langle O(1), O(2), ...\rangle$. A partial order, $\prec$, is defined on O. $O(i) \prec O(j)$ when all the values in any domain of $O(i)$ are also present in the corresponding domain of $O(j)$. $O(i) \prec O(j)$ implies $i < j$. Excepting $O(1)$, all other CSPs in O have a predecessor: $\forall j > 1, \exists i, O(i) < O(j)$.*

An OCSP is exploited by starting with the initial CSP, $O(1)$, and adding relevant values on each failure. Each addition of values corresponds to a jump in O from a CSP $O(i)$ to a CSP $O(j)$ where $O(i) \prec O(j)$.

All the properties described in the previous subsection are inherited. Santiago Macho pursued this research and got interesting experimental results.

17.11 Summary

In this chapter we have learned about an application of distributed protocols. We have first introduced Multi-unit Supervised Combinatorial Exchanges (MUSCEWDPs), a framework for generalized auctions. Generalized English Auctions (GEAs) are applications of AASR to solve MUSCEWDPs, and are generalizations of the well known English Auctions.

We analyze Secure Asynchronous Search (SAS), an extension of AASR that can be use as bidding step in GEAs. While, many practical issues for implementing GEAs remain subject of further research, SAS gives an important image of the type of problems that appear in the application of asynchronous protocols to real negotiation problems.

In the end, we have seen constraint relaxation and Branch and Bound techniques needed with AASR at different steps of the GEA. Some of the properties of GEAs are described in Annex C.