In *Encyclopedia of Artificial Intelligence*. Information Science Reference, 2007.

Distributed Constraint Reasoning

Marius C. Silaghi, *msilaghi@fit.edu*
Makoto Yokoo, *yokoo@is.kyushu-u.ac.jp*

INTRODUCTION

Distributed constraint reasoning is concerned with modeling and solving naturally distributed problems. It has application to the coordination and negotiation between semi-cooperative agents, namely agents that want to achieve a common goal but would not give up private information over secret constraints. When compared to centralized constraint satisfaction (CSP) and constraint optimization (COP), here one of the most expensive operations consists in communication and major problems arise from coherence and privacy. We review approaches based on asynchronous backtracking and depth-first search spanning trees.

Distributed constraint reasoning started as an outgrowth of research in constraints and multi-agent systems. Take the sensors network problem in Figure 1, defined by a set of geographically distributed sensors that have to track a set of mobile nodes. Each sensor can watch only a subset of its neighborhood at a given time. Three sensors need to simultaneously focus on the same mobile node in order to locate it. Approaches modeling

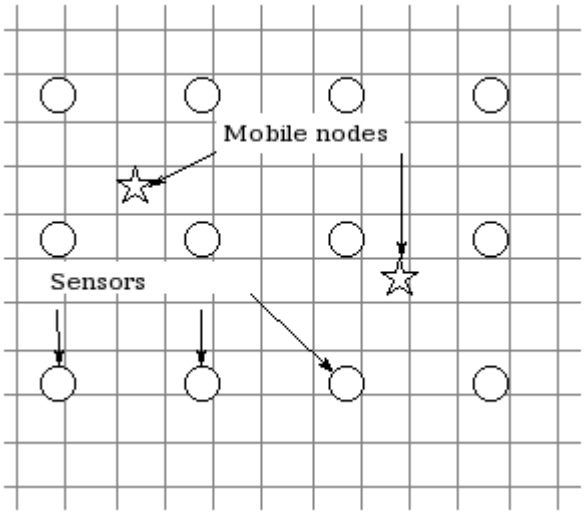

Figure 1: Sensor Network.

and solving this problem with distributed constraint reasoning are described in (Bejar, Domshlak, Fernandez, Gomes, Krishnamachari, Selman, &Valls, 2005).

There are two large classes of distributed constraint problems. The first class is described by a set of *Boolean relations* (aka *constraints*) on possible assignments of variables, where the relations are distributed among agents. They are called distributed constraint satisfaction problems (DisCSPs). The challenge is to find assignments of variables to values such that all these relations are satisfied. However, the reasoning process has to be performed by collaboration among the agents. There exist several solutions to a problem, and ties have to be broken by some priority scheme. Such priorities may be imposed from the problem description where some agents, such as government agencies, are more important than others. In other problems it is important to ensure that different solutions or participants have equal chances, and this property is called *uniformity*. When no solution exists, one may still want to find an assignment of the variables that conflict as few constraints as possible. The second class of problems refers to numerical optimization described by a set of functions (*weighted constraints*) defined on assignments of variables and returning positive numerical values. The goal is to find assignments that minimize the objective function defined by the sum of these functions. The problems obtained in this way are called distributed constraint optimization problems (DisCOPs). Some problems require a fair distribution of the amount of dissatisfaction among agents, minimizing the dissatisfaction of the most unsatisfied agent.

There are also two different ways of distributing a problem. The first way consists of distributing the data associated with it. It is defined in terms of which agents know which constraints. It can be shown that any such problem can be translated into problems where all non-shared constraints are *unary* (constraints involving only one variable), also called *domain constraints*. Here one can assume that there exists a single unary constraint for each variable. It is due to the fact that any second unary constraint can be reformulated on a new variable, required to be equal to the original variable. The agent holding the unique domain constraint of a variable is called the *owner of that variable*. Due to the availability of this transformation many solutions focus on the case where only the unary constraints are not shared by everybody (also said to be private to the agents that know them). Another common simplification consists in assuming that each agent has a single unary constraint (i.e., a single variable). This simplification does not reduce the generality of the addressable problems since an agent can participate in a computation under several names, e.g., one instance for each unary constraint of the original agent. Such false identities for an agent are called pseudo-agents (Modi, Shen, Tambe, & Yokoo, 2005), or abstract agents (Silaghi & Faltings, 2005).

The second way of distributing a problem is in terms of who may propose instantiations of a variable. In such an approach each variable may be assigned a value solely by a subset of the agents while the other agents are only allowed to reject the proposed assignment. This distribution is similar to restrictions seen in some societies where only the parliament may propose a referendum while the rest of the citizens can only approve or reject it. Approaches often assume the simultaneous presence of both ways of distributing the problem. They commonly assume that the only agent that can make a proposal on a variable is the agent holding the sole unary constraint on that variable, namely its owner (Yokoo, Durfee, Ishida, & Kuwabara, 1998). When several

agents are allowed to propose assignments of a variable, these authorized agents are called *modifiers* of that variable. An example is where each holder of a constraint on a variable is a legitimate modifier of that variable (Silaghi & Faltings, 2005).

BACKGROUND

The first addressed challenge concerned the development of *asynchronous algorithms* for solving distributed problems. Synchronization forces distributed processes to run at the speed of the slowest link. Algorithms that do not use synchronizations, namely where participants are at no point aware of the current state of other participants, are flexible but more difficult to design. With the exception of a few *solution detection* techniques (Yokoo & Hirayama, 2005), (Silaghi & Faltings, 2005), most approaches gather the answer to the problem by reading the state of local agents after the system becomes idle and reaches the so called *quiescence* state (Yokoo et al., 1998). Algorithms that eventually reach quiescence are also called *self-stabilizing* (Collin, Dechter, & Katz, 1991). A *complete* algorithm is an algorithm that guarantees not to miss any existing solution. A *sound* algorithm is a technique that never *terminates* in a suboptimal state.

Another challenge picked by distributed constraint reasoning research consists of providing privacy for the sub-problems known by agents (Yokoo et al., 1998). The object of privacy can be of different types. The *existence of a constraint* between two variables may be secret as well as the *existence of a variable* itself. Many approaches only try to ensure the *secrecy of the constraints*, i.e., the hiding of the identity of the *valuations* that are penalized by that constraint. For optimization problems one also assumes a need to keep secret the amount of the penalty induced by the constraint. As mentioned previously, it is possible to model such problems in a way where all secret constraints are unary (approach known as having *private domains*). Some problems may have both secret and public constraints. Such public constraints may be used for an efficient preprocessing prior to the expensive negotiation implied by secret constraints. Solvers that support guarantees of privacy at any cost employ *cryptographic multi-party computations* (Yao 1982). There exist several cryptographic technologies for such computations, and some of them can be used interchangeably by distributed problem solvers. However, some of them offer information theoretical security guarantees (Shamir, 1979) being resistant to any amount of computation, while others offer only cryptographic security (Cramer, Damgaard, & Nielsen, 2000) and can be broken using large amounts of computation or quantum computers. The result of a computation may reveal secrets itself and its damages can be reduced by being careful in formulating the query to the solver. For example, less information is lost by requesting the solution to be picked randomly than by requesting the first solution. The computations can be done cryptographically by a group of *semi-trusted servers*, or they can be performed by participants themselves. A third issue in solving distributed problems is raised by the size and the dynamism of the system.

In *Encyclopedia of Artificial Intelligence*. Information Science Reference, 2007.

DISTRIBUTED CONSTRAINT REASONING

Framework

A common definition of a distributed constraint optimization problem (DisCOP) (Modi et al., 2005) consists of a set of variables $X=\{x_1, ..., x_n\}$ and a set of agents $A=\{A_1, ..., A_n\}$, each agent A_i holding a set of constraints. Each variable x_i can be assigned only with those values which are allowed by a domain constraint D_i. A constraint Φ_j on a set of variables X_j is a function associating a positive numerical value to each combination of assignments to the variables X_j. The typical challenge is to find an assignment to the variables in X such that the sum of the values returned by the constraints of the agents is minimized.

A tuple of assignments is also called *partial solution*. A restriction often used with DisCOPs requires that each agent A_i holds only constraints between xi and a subset of the previous variables, $\{x_1,...,x_{i-1}\}$. Also, for any agent A_i, the agents $\{A_1, ..., A_{i-1}\}$ are the *predecessors* of A_i and the agents $\{A_{i+1}, ..., A_n\}$, are its *successors*.

> To understand the generality and limitations of this restriction, consider a conference organization problem with 3 variables x_1 (time), x_2 (place), and x_3 (general chair) and 3 constraints Φ_{12} (between x_1 and x_2), Φ_{23} (between x_2 and x_3), and Φ_{13} (between x_1 and x_3), where *Alice* has Φ_{12}, *Bob* enforces Φ_{23}, and *Carol* is interested in Φ_{13}, Figure 3.
>
> This problem can be modeled as a DisCOP with 4 agents. Alice uses two agents, A_1 and A_2. The original participant is called *physical agent* and the agents of the model are called *pseudo-agents*. Bob uses the agent A_3 and Carol uses an agent A_4. The new variable x_4 of the agent A_4 is involved in a ternary constraint Φ_{134} with x_1 and x_3. The constraint Φ_{134} is constructed such that its projection on x_1 and x_2 is Φ_{13}.

However the restricted framework cannot help general purpose algorithms to learn and exploit the fact that agents A_1 and A_2 know each other's constraints. It also requires finding an optimal value for the variable x_4, which is irrelevant to the query. To avoid aforementioned limitations some approaches remove the restriction on which variables can be involved in the constraints of an agent and can obtain some improvements in speed (Silaghi & Faltings, 2005). Other frameworks typically used with hill-climbing solvers, with solvers that reorder agents, and with arc consistency, assume that each agent A_i knows all the constraints that involve the variable x_i. This implies that any constraint between two variables x_i and x_j is known by both agents A_i and A_j. In general, a problem modeled as a DisCOP where any private constraint may be hold by any agent can be converted to its dual representation in order to obtain a model with this framework. When penalties for constraint violation can only take values in $\{0,\infty\}$, corresponding to *{true, false}*, one obtains distributed constraint satisfaction problems.

A *protocol* is a set of rules about what messages may be exchanged by agents, when they may be sent, and what may be contained in their payload. A *distributed algorithm* is an implementation of a protocol as it specifies an exact sequence of

operations to be performed as a response to each event, such as start of computation or receipt of a message. Autonomous self-interested agents are more realistically expected to implement protocols rather than to strictly adhere to algorithms. Protocols can be theoretically proved correct. However experimental validation and efficiency evaluation of a protocol is done by assuming that agents strictly follow some algorithm implementing that protocol.

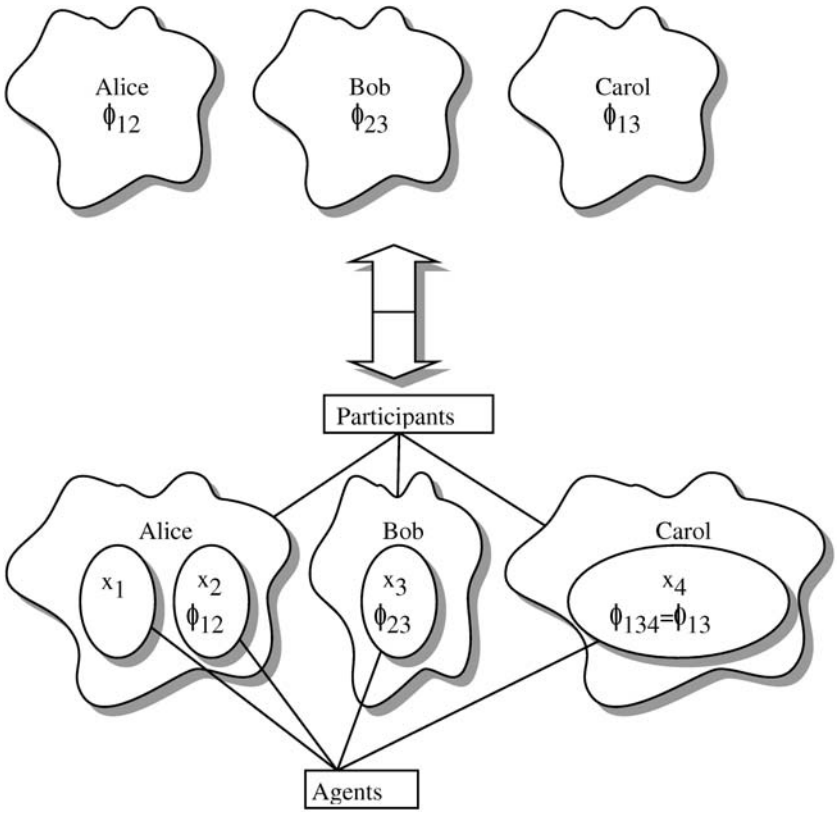

Figure 2: Translating between DisCOP frameworks

Efficiency metrics

The simplest metric for evaluating DisCOP solvers uses the time from the beginning of a distributed computation to its end. It is possible only with a real distributed system (or a very realistic simulation). The network load for benchmarks is evaluated by counting the total number of messages exchanged or the total number of bytes exchanged. The total time taken by a simulator yields the efficiency of a DisCOP solver when used as a weighted CSP solver. Another common metric is given by the *highest logic clocks* (Lamport, 1978) occurring during the computation. Lamport's logic clocks associate a cost with each message and another cost with each local computation. When the cost assigned to each message is 1 and the cost for local computations is 0, the obtained value gives the *longest sequential chain of causal messages* (Silaghi & Faltings, 2005). When all message latencies are identical, this metric is equivalent to the number of

rounds of a simulator where at each round an agent handles all messages received in the previous round (Yokoo et al., 1998). If the cost assigned to each message is 0 and the cost of a constraint check is 1, the obtained value gives the number of *non-concurrent constraint checks* (NCCC) (Meisels, Kaplansky, Razgon, & Zivan, 2002). When a constraint check is assumed to cost a fraction of a message then the obtained value gives the *equivalent NCCCs*. One can evaluate the actual fraction between message latencies and constraint checks in the operating point (OP) of the target application (Silaghi & Yokoo, 2007). However many distributed solvers do not check constraints directly but via *nogoods* and there is no standardized way of accounting the handling of the latter ones.

Techniques

Solving algorithms span the range between full centralization, where all constraints are submitted to a central server that returns a solution, through incremental centralization (Mailler & Lesser, 2004), to very decentralized approaches (Walsh, Yokoo, Hirayama, & Wellman, 2003).

The Depth-First Search (DFS) spanning trees of the constraint graph proves useful for distributed DisCOP solvers. When used as a basis for ordering agents, the assignment of any node of the tree makes its subtrees independent (Collin, Dechter, & Katz, 2000). Such independence increases parallelism and decreases the complexity of the problem. The structure can be exploited in three ways. Subtrees can be explored in parallel for an opportunistic evaluation of the best branch, reminding of iterative A* (Modi et al., 2005). Alternatively a branch and bound approach can systematically evaluate different values of the root for each subtree (Chechetka & Sycara, 2006). A third approach uses dynamic programming to evaluate the DFS trees from leaves towards the root (Petcu & Faltings, 2006).

Asynchronous usage of *lookahead* techniques based on maintenance of *arc consistency* and *bound consistency* require handling of interacting data structures corresponding to different concurrent computations. Concurrent consistency achievement processes at different depths in the search tree have to be coordinated giving priority to computations at low depths in the tree (Silaghi & Faltings, 2005).

The concept at the basis of many asynchronous algorithms is the *nogood*, namely a self contained statement about a restriction to the valuations of the variables, inferred from the problem. A *generalized valued nogood* has the form $[R,c,T]$ where T specifies a set of partial solutions $\{N_1,...,N_k\}$ for which the set of constraints R specifies a penalty of at least c. A common simplification, called *valued nogood* (Dago & Verfaille, 1996), refers to a single partial solution, $[R,c,N]$. Priority induced vector clock timestamps called *signatures* can be used to arbitrate between conflicting assignments concurrently proposed by several modifiers (Silaghi & Faltings, 2005). They can also handle other types of conflicting proposals, such as new ordering.

ADOPT-ing is an illustrative algorithm unifying the basic DisCSP and DisCOP solvers ABT (Yokoo et al., 1998) and ADOPT (Modi et al., 2005). It works by having each agent concurrently chose for its variable the best value given known assignments of

predecessors and cost estimations received from *successors* (Silaghi & Yokoo 2007). Each agent announces its assignments to interested successors using **ok?** messages. Agents are interested in variables involved in their constraints or nogoods. When a nogood is received, agents announce new interests using **add-link** messages. A forest of DFS trees is dynamically built. Initially each agent is a tree, having no ancestors. When a constraint is first used, the agent adds its variables to his ancestors list and defines his *parent* in the DFS tree as the closest ancestor. Ancestors are announced of their own new ancestors. Nogoods inferred by an agent using resolution on its nogoods and constraints are sent to targeted predecessors and to its parent in the DFS tree using **nogood** messages, to guarantee optimality. Known costs of DFS subtrees for some values can be announced to those subtrees using **threshold** nogoods attached to **ok?** messages.

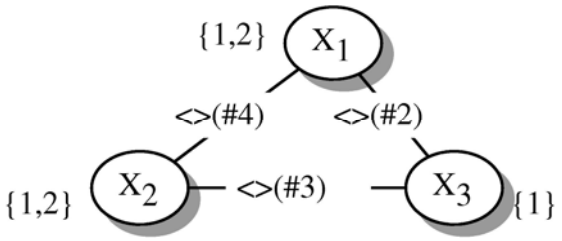

Figure 3: The constraint graph of a DisCOP. The fact that the penalty associated with not satisfying the constraint $x_1 \neq x_2$ is 4, is denoted by the notation (#4).

Example

An asynchronous algorithm could solve the problem in Figure 2 using the trace in Figure 3. In the messages of Figure 3, constraints are represented as Boolean values in an array. The i^{th} value in this array set to *T* signifies that the constraints of A_i are used in the inference of that nogood. The agents start selecting values for their variables and announce them to interested lower priority agents. The first exchanged messages are **ok?** messages sent by A_1 to both successors A_2 and A_3 and proposing the assignment x_1=1. A_2 sends an **ok?** message to A_3 proposing x_2=2.

1. A_1	**ok?**$\langle x_1, 1\rangle$	A_2, A_3
2. A_2	**ok?**$\langle x_2, 2\rangle$	A_3
3. A_3	**nogood**$([\|F, F, T\|, 2, \langle x_1, 1\rangle], 3, \{A_1\})$	A_1
4. A_1	**ok?**$\langle x_1, 2\rangle$	A_2, A_3
5. A_2	**ok?**$\langle x_2, 1\rangle$	A_3
6. A_3	**nogood**$([\|F, F, T\|, 3, \langle x_2, 1\rangle], 3, \{A_1, A_2\})$	A_2
7. A_2	**nogood**$([\|F, T, T\|, 3, \langle x_1, 2\rangle], 2, \{A_1\})$	A_1
8. A_1	**ok?**$\langle x_1, 1\rangle$	A_2
9. A_1	**ok?**$\langle x_1, 1\rangle$ **threshold** $[\|F, F, T\|, 2, \langle x_1, 1\rangle]$	A_3
10. A_2	**ok?**$\langle x_2, 2\rangle$	A_3

Figure 4: Simplified trace of an asynchronous solver (ADOPT-ing (Silaghi & Yokoo, 2007)*) on the problem in Figure 3.*

A_3 detects a conflict with x_1, inserts A_1 in its DFS tree ***ancestors*** list, and sends a nogood with cost 2 to A_1 (message 3). A_1 answers the received nogood by switching its assignment to a value with lower current estimated value, x_1=2 (message 4). A_2 reacts by switching x_2 to its lowest cost value, x_2=1 (message 5). A_3 detects a conflict with x_2 and inserts A_2 in its ***ancestors*** list, which becomes $\{A_1, A_2\}$. A_3 also announces the conflict to A_2 using the **nogood** message 6. This nogood received by A_2 is combined with the nogood locally inferred by A_2 for its value 2 due to the constraint $x_1 \neq x_2$(#4). That inference also prompts the insertion of A_1 in the ***ancestors*** list of A_2. The obtained nogood is therefore sent to A_1 using message 7. A_1 and later A_2 switch their assignments to the values with the lowest cost, attaching the latest nogoods received for those values as threshold nogoods (messages 8, 9 and 10). At this moment the system reaches *quiescence*.

FUTURE TRENDS

The main remaining challenges with distributed constraint reasoning are related to efficient ways of achieving privacy and with handling very large problems.

CONCLUSION

The distributed constraint reasoning paradigms allow easy specification of new problems. The notion varies largely between almost any two researchers. It can refer to the distribution of subproblems or it can refer to the distribution of authority in assigning variables. The reason and goals of the distribution vary as well, where either privacy of constraints, parallelism in computation, or size of data are cited as major concern. Most algorithms can be easily translated from one framework to the other, but they may not be appropriate for a new goal.

REFERENCES

Bejar, R., Domshlak, C., Fernandez, C., Gomes, C., Krishnamachari, B., Selman, B., &Valls, M. (2005). Sensor networks and distributed CSP: communication, computation and complexity. *Artificial Intelligence*, 161(1-2):117-147.

Chechetka, A., & Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*, 1427-1429.

Cramer, R., Damgaard, I., & Nielsen, J.B. (2000). *Multi-party Computation from Threshold Homomorphic Encryption*. BRICS RS-00-14.

Collin, Z.; and Dechter, R.; and Katz,S. (1991). On the feasibility of distributed constraint satisfaction. *IJCAI*, 318-324.

Collin, Z., Dechter, R., & Katz,S. (2000). Self-Stabilizing Distributed Constraint Satisfaction. *Chicago Journal of Theoretical Computer Science*, 3(4).

Dago, P., & Verfaillie, G. (1996). Nogood recording for valued constraint satisfaction problems. In *ICTAI*, 132-139

Lamport, L. (1978). Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*. 21(7):558-565.

Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, 438-445.

Meisels, A., Kaplansky, E., Razgon, I., & Zivan, R. (2002). Comparing Performance of Distributed Constraints Processing Algorithms. *DCR*, 86-93.

Modi, P. J., Shen, W.-M., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence Journal* 161(1-2).

Petcu, A., & Faltings, B. (2006). ODPOP: an algorithm for open/distributed constraint optimization. In *AAAI*.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*. 22:612-613.

Silaghi, M.-C., & Faltings, B. (2005). Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence Journal* 161(1-2):25-53.

Silaghi, M.-C., & Yokoo, M. (2007). Dynamic DFS Tree in ADOPT-ing. *AAAI*.

Walsh, W.E., M. Yokoo,M., K. Hirayama, K., & M.P. Wellman, M.P. (2003). On market-inspired approaches to propositional satisfiability. *Artificial Intelligence*. 144: 125-156.

Walsh, T. (2007). Traffic light scheduling: a challenging distributed constraint optimization problem. In *DCR*.

Yao, A. (1982). Protocols for secure computations. *FOCS*. 160-164.

Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. In *IEEE TKDE*, 10(5) 673-685.

Yokoo, M., & Hiramaya, K. (2005). The Distributed Breakout Algorithm. *Artificial Intelligence Journal*. 161(1-2), 229-246.

TERMS AND DEFINITIONS

Agent: A participant in a distributed computation, having its own constraints

Constraint: A relation between variables specifying a subset of their Cartesian product that is not permitted. Optionally it can also specify numeric penalties for those tuples

DisCOP: Distributed Constraint Optimization Problem framework (also DCOP)

DisCSP: Distributed Constraint Satisfaction Problem framework (also DCSP)

Nogood: A logic statement about combinations of assignments that are penalized due to some constraints

Optimality: The quality of an algorithm of returning only solutions that are at least as good as any other solution

Quiescence: The state of being inactive. The system will not change without an external stimulus.