

Dynamic Distributed Constraint Reasoning

Robert N. Lass, Evan A. Sultanik and William C. Regli

Drexel University
Department of Computer Science
3141 Chestnut St.
Philadelphia, PA 19104
{urlass, eas28, regli}@cs.drexel.edu

Abstract

What local action can agents take, without the benefit of global knowledge, to produce the best global solution? Many dynamic distributed systems can be modeled using techniques from distributed constraint reasoning, however, existing work in the distributed constraint reasoning community does not address the true dynamism inherent in many real-world systems.

Introduction

When deploying a dynamic distributed system, one must always weigh the latency incurred by network communication against the algorithmic benefit of using a centralized approach. In some cases, the problem may be changing so fast or the communications overhead so expensive that a centralized algorithm will not be able to maintain stability due to dynamism and the fact that the system state is distributed across the network (Dijkstra 1974). It is therefore imperative to emphasize *local* decision making and autonomy over a centralized analogue, insofar as it is possible.

Sensor networks, for example, often communicate over an ad-hoc networks. Such networks suffer from high amounts of bit and frame errors, requiring constant retransmission of data. Moreover, mobile ad-hoc networks (MANETs) suffer high packet losses and frame error rates, resulting in less than 50% of the theoretical maximum throughput being achieved (Xylomenos *et al.* 2001). Since MANET network topology is inherently in flux, the additional messaging overhead of a centralized control approach can have a significant negative effect on the controllability, stability, and overall robustness of the system.

As a motivating scenario consider a series of networked sensors, as depicted in Figure 1. The sensors are fixed in position (*i.e.* guarding the perimeter of a building or the length of a national border) and they may even have been randomly dispersed, as in an ad-hoc sensor network (Perkins 2000). Each sensor has limited rotational degrees of freedom and has a limited sensing range. Various events can occur that may dynamically change the situation: an object such as an automobile could partially or totally obstruct a sensor; a sensor might fail; a new sensor might become available; a

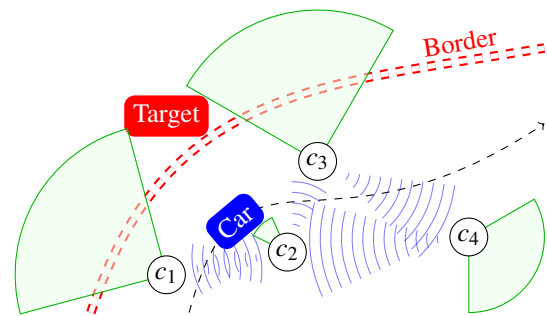


Figure 1: An ad-hoc sensor network guarding a border must reorient after a car blocks c_2 's view of the target.

subset of the sensors might have locomotion (*e.g.* mounted on a moving vehicle); or the attention of multiple sensors might be required to investigate a target. How can the sensors coordinate to maximize coverage in spite of any such events? The purpose of this paper is to formalize dynamic, distributed, constraint reasoning (DDCR) problems such as these, and the types of cooperative strategies that could be used to solve them—ultimately challenging the community with this problem domain.

Background

Constraint programming (CP) is a paradigm in which problems are solved by satisfying constraints between variables rather than executing a series of statements. There are two primary methods for handling problem dynamism in CP. Often, the static representation of the problem is updated and completely re-solved. At the other end of the spectrum, Assumption-based Truth Maintenance Systems (ATMS) (de Kleer 1986) are used to handle dynamic CP problems. In the middle (polynomial space) is the technique of Nogood Recording (NR) (Schiex & Verfaillie 1993). The primary shortcoming of these approaches is that they are designed for centralized computation. It is not obvious how to execute these algorithms distributedly, nor how to handle agents with temporally incompatible views of the problem.

Distributed constraint reasoning generally refers to distributed constraint satisfaction (DisCSP) and distributed constraint optimization (DCOP). DisCSP was formalized

in (Yokoo *et al.* 1998) and (Zhang & Mackworth 1991). DCOP is analogous to DisCSP, but allows the constraints to take on any cost (rather than just a binary cost), and tries to minimize the total cost of the solution. An early algorithm that has been employed to solve distributed constraint reasoning problems is self-stabilizing distributed branch and bound (Yahfoufi & Dowaji 1996).

Petcu has evaluated Dynamic DCOPs in a few recent works (Petcu & Faltings 2005a; 2005b) as a way to mitigate the latency inherent with a centralized solution. Petcu proposes that self-stabilization, as proposed by Dijkstra (Dijkstra 1974), is a fundamental property needed for DD-COP (Petcu & Faltings 2005b). Self-stabilization is a property of an algorithm that is able to start in an arbitrary state and eventually reach a so-called “good” state. Presented is an method composed of three self-stabilizing algorithms: DFS, utility propagation, and value propagation. Petcu also defines a distributed continuous time combinatorial optimization problem and proposes an algorithm based on SDPOP for solving the problem based on the notion of taking into account the cost of transitioning to a new solution versus the benefit provided by the new solution (Petcu & Faltings 2005a). The algorithm is also resilient to variable dynamism, and assignments that are irreversible after a specified time. In these formalizations of the problem, it is simply stated that the sets of agents, variables, and constraint relation functions can vary over time. An issue with the DPOP variants is that they implicitly assume that knowledge of such set variances is available to all agents.

In the field of discrete event systems (DES) techniques have also been developed to generate decentralized controllers in certain cases (Kozak & Wonham 1995). There exists work that suggests some multiagent systems planning problems can be solved using DES (Seow, Ma, & Yokoo 2004), but it is still unclear as to whether there is application in distributed constraint reasoning problems. Two common methods for modeling DES are finite automata and petri nets. Methods have been investigated for representing decentralized control (Kozak & Wonham 1995) and constraint reasoning (Portinale 1997) problems using these modeling techniques, however, the resulting models grow exponentially with respect to the number of variables and domain size, thus rendering them infeasible.

Problem Definition & Formalization

Although there has been much interest in the burgeoning field of distributed constraint reasoning (Zhang & Mackworth 1991; Yokoo *et al.* 1998; Modi *et al.* 2003; Petcu & Faltings 2004) and even the dynamic variant of our focus (Petcu & Faltings 2005b; 2005a), there is no widely used theoretical framework for distributed constraint reasoning (dynamic or not) that takes into account the fact that each agent may not have perfect knowledge. Constructing such a framework requires separately modeling each agent’s belief of the world, which may not be the same as the global view or what an omniscient entity would know.

Definition 1. There are four components to a *Distributed Constraint Reasoning Problem* (DCR): a set of *agents*, a set

of *variables* that are to be assigned values by the agents, a set of *domains* that contain the values that may be assigned to said variables, and a set of *constraints* over the variables’ assignments. The objective is to have the agents assign values (taken from the domains) to their variables such that some metric over the resulting constraints’ values is either satisfied, minimized, or maximized.

Definition 2. Functionally, *events* are all composed from the following primitives: **Increase** the cost of a constraint; **Decrease** the cost of a constraint; **Give Ownership** of a variable to an agent; **Add** a value to the domain of a variable; and **Remove** a value from the domain of a variable. For example, removal of a variable is functionally equivalent to removing all of the elements from its domain. Likewise, deleting an agent is functionally equivalent to giving ownership of its variables to another agent. The algorithm used by agents to solve the problem will add other events, like those used in DCR algorithms such as value update messages from Adopt (Modi *et al.* 2003), or VALUE messages in DPOP (Petcu & Faltings 2004).

Definition 3. A *Context* is an assignment of values to variables for a DCR problem. Essentially, a context is a (possibly partial) solution to a problem. Contexts may be represented one of two ways, depending on which is more succinct for the problem being represented:

1. a set of ordered pairs, such as $\{\langle v_1, 5 \rangle, \langle v_2, 9 \rangle\}$, meaning that variable v_1 has been assigned value 5 and variable v_2 has been assigned value 9; or
2. a function mapping the set of all variables to the set of all possible values (also including the possibility of a variable not receiving assignment).

Definition 4. Two contexts are said to be *Compatible* if they do not disagree on the assignment of any variables. This is the standard definition of context compatibility in terms of static DCR, and is usually caused by one agent proposing a variable assignment based upon an inaccurate or out of date belief of another agent’s assignment. In dynamic DCR, however, contexts can additionally be rendered incompatible due to any of the problem-altering events listed above. We shall call this *temporal incompatibility*.

Definition 5. Each agents’ view of the “Dynamic Distributed Constraint Reasoning Problem” (DCRP) can be formalized as a tuple $a_i \equiv \langle A_i, V_i, \mathcal{D}_i, f_i, \alpha_i, \sigma_i \rangle$:

- a_i this agent;
- A_i is an ordered set of agents about which a_i knows;
- V_i is an ordered set of variables, $\{v_1, v_2, \dots, v_{|V_i|}\}$ about which a_i knows;
- \mathcal{D}_i is an ordered set of domains, $\{D_1, D_2, \dots, D_{|V_i|}\}$, where each $D \in \mathcal{D}_i$ is a finite and/or ordered set containing the values a_i believes the associated variable in V_i may be assigned;
- f_i is a function mapping all possible contexts (*i.e.* partial solutions) to a cost. The costs could be in any metric space.

Two important special cases exist (1) if the costs are binary, this is a *dynamic distributed constraint satisfaction problem* (DDisCSP) (2) if the costs are in $N_0 \cup \infty$, the prob-

lem is a *dynamic distributed constraint optimization problem* (DDCOP). These are analogous to the well-known static DisCSP and DCOP problems (see Definition 1). When the context is represented as a set of ordered pairs, f_i can be formalized as follows:

$$f_i : \bigcup_{S \in \mathcal{P}(V)} \prod_{v_i \in S} (\{v_i\} \times D_i) \rightarrow M,$$

where “ $\mathcal{P}(V)$ ” denotes the power set of V and “ \prod ”, “ \times ” represent the Cartesian product and M is some metric space;

- α_i is a function $\alpha_i : V_i \rightarrow A_i$ mapping variables to their associated agent. $\alpha_i(v_i) \mapsto a_j$ implies that it is agent a_j 's responsibility to assign the value of variable v_i . Note that it is not necessarily true that α_i is either an injection or surjection; and
- σ_i is an operator that aggregates the f_i costs for all combinations of assignments in a given context. In the special cases of DDCOP and DDisCSP, this is summation and conjunction, respectively.

When an event is sensed by an agent, it *processes* the event. The event may change any of the items listed in Definition 2. For example, if a_i senses an event indicating that agent, a_j no longer has one of the previous values in its domain, it will remove that value from $D_j \in \mathcal{D}_i$.

Research issues that arise are (1) Knowledge and information about changes must be disseminated to relevant agents. (*e.g.* If a sensor goes down, when will other agents find out?) If all agents start off with a global view of the problem, then $\forall \langle a_i, a_j \rangle, a_i \equiv a_j$. Once an event occurs that not all agents see, $\exists \langle a_i, a_j \rangle, a_i \not\equiv a_j$. In this situation, temporally incompatible contexts (see Definition 4) may force the agent who sensed the event to forward it to agents it is exchanging messages with. (2) The state may change so rapidly that the algorithm never stabilizes (Dijkstra 1974). For example, how well can the agents monitor the border if the sensors go up and down every few seconds? If events occur more rapidly than they can be sent or agents can send messages, no algorithm will be able to optimally solve the problem. (3) Agents may be able to predict events. (*e.g.* Agents may know that sensors have a certain maintenance schedule, during which time they are unavailable.) If agents are able to predict the occurrence of events in advance, they may be able to solve for periods of rapid change before they actual occur, thus making an intractable problem tractable. (4) Events will change the optimal solution. If an agent has a stable solution, and processing the event does not make the solution inconsistent, no further action is needed. We will call this a *neutral event*. If an agent has a stable solution and processing the event makes the solution inconsistent, then we call this a *disruptive event*. If the event creates a situation where the agent must restart its algorithm (*i.e.* it cannot re-use any of the previous computation), then we call this a *destructive event*.

Example Formulation & Analysis

For each sensor, $c_i \in C$, create an agent $a_i \in A$ and give it ownership of one variable $v_i \in V$. The domain of each vari-

		Messaging	
		PERFECT	IMPERFECT
Event Sensing	PERF.	SDPOP	Adopt Variant
	IMPR.	DDCR Adapter	DDCR Adapter

Table 1: A possible partitioning of DDCR algorithms

able is the set of all pairs $\langle \theta_x, \theta_y \rangle$, where each θ is between 0 and 360 in steps of 15.

The cost function, f , for a given sensor, c , is the sum of all targets within the range of c that are not in view of any sensor. In a practical system, there are at least two obvious ways this could be implemented. First, each target could be equipped with a simple networked computing device. The sensors could notify targets when they are looking at them periodically. If a target did not receive any messages after a fixed length of time, it would know it is unmonitored. To calculate f , each sensor could query each target in range. A second method, useful when the targets are adversarial, is to have each sensor communicate with every other sensor that can see each target in its own range.

α and σ remain as defined in the formalization at the beginning of this section.

How well can an algorithm perform, given these restrictions? If a system changes slowly enough, one approach is to use an algorithm from the DCR community to re-solve the problems every time it changes. If privacy and bandwidth are not a concern, each agent could send its partial problem to a central location and employ an algorithm from the CP community. If the problem is small enough, techniques from DES could be used to compute an optimal distributed controller. In any of these cases, this would guarantee an optimal solution.

In some environments, such as communication over a wired enterprise network, agents may be able to freely interchange messages. In other environments such as mobile ad-hoc networks with poor link quality, however, message interchange may be very expensive. In mobile robotics with limited battery power, changing the solution may be costly. What types of environments are suited to which algorithms? Given that the best algorithm is often a function of the agent's environment, the measure of effectiveness must be tied to the measures of performance when evaluating the algorithms. Certain levels of performance may preclude an optimal level of effectiveness.

One way of classifying algorithms is based on the assumptions that need to be satisfied in order for the algorithm to function properly. For example, as depicted in Table 1 the space of algorithms could be divided based on whether or not the algorithm assumes events are instantaneously sensed by all participating agents, and whether or not the algorithm assumes perfect message delivery.

There are a number of trade-offs that an agent functioning in a dynamic environment may have to make. As an example using the border guarding scenario, consider the situation where an agent, a is constrained with an agent, b . The sensor controlled by b is viewing a busy road, and is constantly changing its position due to obstruction by moving

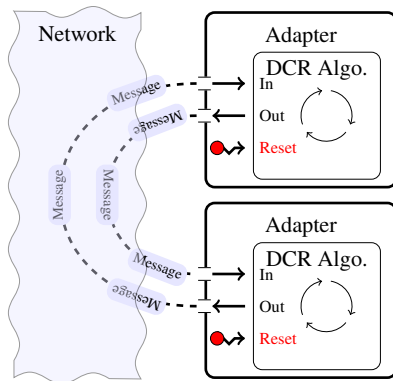


Figure 2: The DDCR ADAPTER approach to solving dynamic distributed constraint reasoning problems.

automobiles. a is constrained with several other variables with relatively stable values. It might be best for a to ignore its constraint with b , rather than change its value rapidly, upsetting the other agents with which it is constrained, causing a chain-reaction of instability across the network.

Proposed Solution & Future Work

In this section we propose a simple extension to existing DCR algorithms to cover the southern quadrants of Table 1, which there are currently no known algorithms for. This algorithm is proposed as a starting point for the community, and we do not mean to imply that this is the best solution.

The algorithm, DDCR Adapter (Figure 2), is similar to the adapter design pattern used in software engineering (Gamma *et al.* 1995). Consider an interface to a known DCR algorithm, such as Adopt, that has some knowledge about the internals of the algorithm. When any message is received, such as an event or a standard Adopt message, the interface acts as a filter, and takes appropriate action. If it is a neutral event, no action is required. If it is a disruptive event, it will modify the algorithm in the appropriate fashion. For example, if the cost of a constraint increases, the adapter can notify Adopt of the cost increase, and Adopt can handle the event naturally (*i.e.* it can update f_i to use the new cost, and continue execution). If it is a destructive event, the interface “presses a reset button,” effectively starting the algorithm over again. This requires relaying a restart message between all of the adapters, so that other agents will know that they must restart the algorithm as well. Determining on which iteration of the algorithm could be accomplished by timestamping messages using Lamportian clocks, but the actual implementation is outside the scope of this paper.

This paper introduced a type of problem called dynamic, distributed constraint reasoning. This problem draws largely from the field of distributed constraint reasoning, but adds the complicating factor of dynamism. We partitioned the space of possible DDCR algorithms into four possible categories, and presented a simple extension to existing DCR algorithms, DDCR Adapter, to cover to two categories for which no current algorithm existed.

This paper presents a challenge to the community, and

DDCR Adapter as a starting point. Future work includes better techniques and algorithms for solving DDCR problems, as well as new applications of DDCR to existing, real-world problems. Another challenge is to measure the performance of existing algorithms in different environments, which has proved challenging in the DCR community.

References

- de Kleer, J. 1986. An assumption-based truth maintenance system. *Artificial Intelligence* 28(2):127–162.
- Dijkstra, E. W. 1974. Self-stabilizing systems in spite of distributed control. *Comms. of the ACM* 17(11):643–644.
- Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing.
- Kozak, P., and Wonham, W. 1995. Fully decentralized solutions of supervisory control problems. *IEEE Trans. on Automatic Control* 40(12):2094–2097.
- Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *Proc. of AAMAS*, 161–168.
- Perkins, C. E. 2000. *Ad Hoc Networking*. Addison-Wesley.
- Petcu, A., and Faltings, B. 2004. A distributed, complete method for multi-agent constraint optimization. In *Proceedings of the Fifth International Workshop on DCR*.
- Petcu, A., and Faltings, B. 2005a. R-DPOP: Optimal solution stability in continuous-time optimization. In *Proceedings of the Sixth International Workshop on DCR*.
- Petcu, A., and Faltings, B. 2005b. S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of AAI*, 449–454.
- Portinale, L. 1997. Modeling and solving constraint satisfaction problems through petri nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, 348–366. Springer-Verlag.
- Schiex, T., and Verfaillie, G. 1993. Nogood recording for static and dynamic constraint satisfaction problems. *Tools with Artificial Intelligence* 48–55.
- Seow, K.-T.; Ma, C.; and Yokoo, M. 2004. Multiagent planning as control synthesis. In *Proceedings of AAMAS*, 972–979. IEEE Computer Society.
- Xylomenos, G.; Polyzos, G.; Mahonen, P.; and Saaranen, M. 2001. TCP performance issues over wireless links. *IEEE Communications Magazine* 39(4):52–58.
- Yahfoufi, N., and Dowaji, S. 1996. A self-stabilizing distributed branch-and-bound algorithm. *The IEEE 15th Annual Phoenix Conf. on Computers and Comm.* 246–252.
- Yokoo, M.; Durfee, E.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 10(5):673–685.
- Zhang, Y., and Mackworth, A. K. 1991. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, 394–397.