Distributed Problem Solving

William Yeoh, Makoto Yokoo

■ *Distributed problem solving is a subfield within multiagent systems, where agents are assumed to be part of a team and collaborate with each other to reach a common goal. In this article, we illustrate the motivations for distributed problem solving and provide an overview of two distributed problem-solving models, namely distributed constraint-satisfaction problems (DCSPs) and distributed constraint-optimization problems (DCOPs), and some of their algorithms.*

Broadly, distributed problem solving is a subfield within multiagent systems, where the focus is to enable multiple agents to work together to solve a problem. These agents are often assumed to be cooperative, that is, they are part of a team or they are self-interested but incentives or disincentives have been applied such that the individual agent rewards are aligned with the team reward.

We illustrate the motivations for distributed problem solving with an example. Imagine a decentralized channel-allocation problem in a wireless local area network (WLAN), where each access point (agent) in the WLAN needs to allocate itself a channel to broadcast such that no two access points with overlapping broadcast regions (neighboring agents) are allocated the same channel to avoid interference. Figure 1 shows example mobile WLAN access points, where each access point is a Create robot fitted with a wireless CenGen radio card. Figure 2a shows an illustration of such a problem with three access points in a WLAN, where each oval ring represents the broadcast region of an access point.

This problem can, in principle, be solved with a centralized approach by having each and every agent transmit all the relevant information, that is, the set of possible channels that the agent can allocate itself and its set of neighboring agents, to a centralized server. However, this centralized approach may incur unnecessary communication cost compared to a distrib-

 ISSN 0738-4602

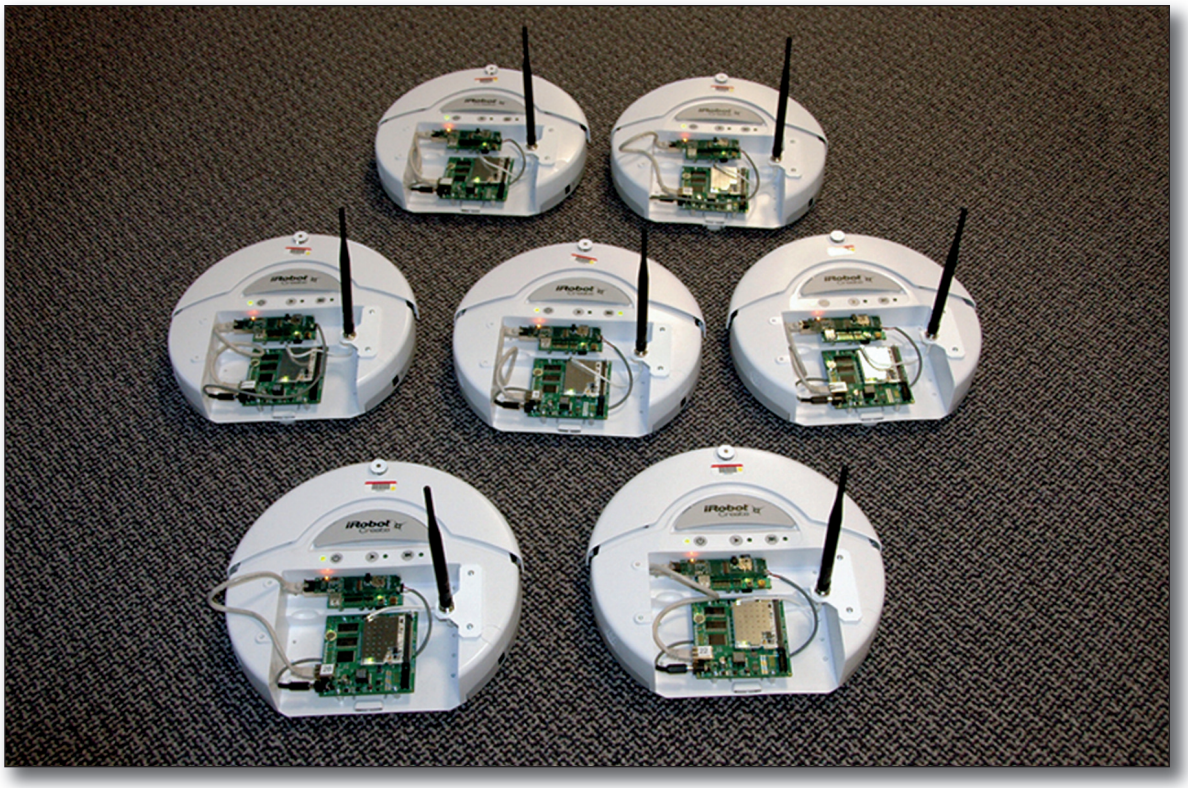

Figure 1. Mobile WLAN Access Points.

uted approach. For example, agents in the centralized approach need to send information to a centralized server, which can be many hops away in the WLAN. On the other hand, agents in a distributed approach need only send information to their neighboring agents, which are one hop away. Additionally, a distributed approach can also take advantage of parallelism to solve the problem faster. For example, if the WLAN is composed of two disjoint networks, then the two problems can be solved in parallel in a distributed approach but must be solved in sequence in a centralized approach. A distributed approach also removes single points of failure, such as the centralized server, which increases robustness.

Although there are many distributed problem-solving models, we focus our scope in this article on distributed constraint-reasoning (DCR) models such as distributed constraint-satisfaction problems (DCSPs)[1] and distributed constraint-optimization problems (DCOPs). The DCR models have a rich history and have been used to model a wide variety of distributed problems including the distributed scheduling of jobs in a job shop (Sycara et al. 1991), the distributed scheduling of meetings (Maheswaran et al. 2004; Zivan 2008), the distributed allocation of targets to sensors in a network (Zhang et al. 2003; Zivan, Glinton, and Sycara 2009), the distributed allocation of resources in disaster evacuation scenarios (Lass et al. 2008), the distributed management of power distribution networks (Kumar, Faltings, and Petcu 2009), the distributed generation of coalition structures (Ueda, Iwasaki, and Yokoo 2010), and the distributed coordination of logistics operations (Léauté and Faltings 2011).

A DCSP or DCOP can be visualized as a graph, where nodes are agents and edges are constraints that represent interactions between neighboring agents. If we model the decentralized channel allocation problem as a DCSP, then a constraint is unsatisfied if the two agents sharing that constraint choose the same channel. The constraint is satisfied otherwise. The goal in a DCSP is to find an allocation of channels to all agents such that all constraints are satisfied. If we model the problem as a DCOP, then a constraint incurs a cost of infinity if the two agents sharing that constraint choose the same channel. The constraint incurs a finite cost otherwise. Each pair of nonconflicting channels typically has a different cost to reflect the channel preferences. The channel preferences might arise due to interference from external sources (Clifford and Leith 2007). The goal in a DCOP is to find an allocation of channels to all agents such that the sum of the costs of all constraints is minimized.

Our emphasis on DCR models in this article comes at the unfortunate cost of neglecting many other topics within the distributed problem-solv-

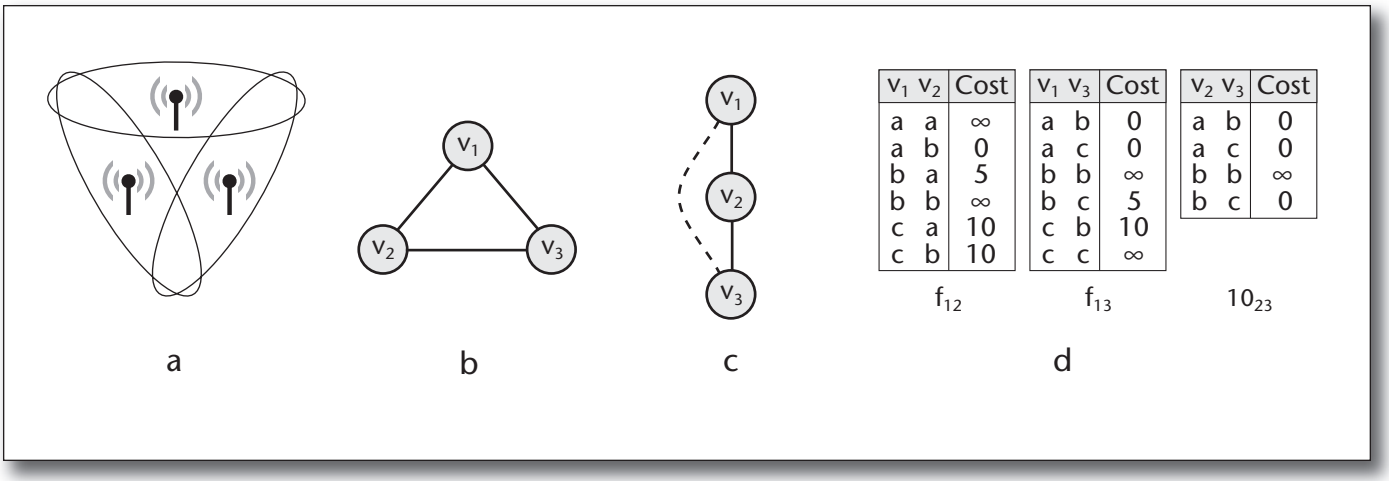

Figure 2. Example WLAN Problem.

(a) Illustration (b) Constraint Graph (c) Pseudotree.

ing subfield. For example, in this article, we assume each agent already has its own subtask. In our decentralized channel allocation problem, the subtask of each agent is to assign its own channel. However, in general, there can be a situation where a team of agents has a global task to accomplish, but how that task can be divided into subtasks and which agent should be responsible for each subtask are not predetermined. Thus, the agents must decide how to do so. The research topic related to such a situation is called task sharing. A classic, representative approach in task sharing is the contract net protocol (Davis and Smith 1983). Also, when the subtask of each agent is complex and interdependent with other subtasks, rather than solving its subtask independently, agents should provide each other with the information of partial solutions as they work toward a solution. Such an approach is called result sharing, which can improve the accuracy of the obtained solution since agents can cross-check their partial solutions and obtain a better overall view (Lesser and Corkill 1981). In this article, we first describe centralized constraint-reasoning models before describing their distributed counterparts. We then describe other distributed problem-solving models and conclude this article.

Centralized Constraint Reasoning

There are two types of centralized constraint-reasoning models, namely constraint-satisfaction problems (CSPs) and constraint-optimization problems (COPs).[2] Both CSPs and COPs are defined by a tuple $\langle V, D, F \rangle$, where $V = \{v_i, \ldots\}$ is a finite set of variables; $D = \{D_i, \ldots\}$ is a finite set of domains, where D_i is the finite set of possible values for variable v_i; and $F = \cup F_i$ is a finite set of constraints, where F_n is the set of n-ary constraints. We assume that all constraints in this article are binary constraints, that is, they involve only two variables, for simplicity. Thus, $F = F_2 = \{f_{ij} \ldots\}$, where $f_{ij} : D_i \times D_j$ is a binary constraint between variables v_i and v_j. Each constraint in a CSP returns a boolean value of satisfied or unsatisfied and each constraint in a COP returns a numeric cost. A solution for the problem is an assignment of values to variables. An optimal solution is an assignment where all constraints are satisfied if the problem is a CSP or the sum of the costs of all constraints is minimized if the problem is a COP. Finding optimal solutions in both problems is NP-hard.

Both problems can be visualized as a constraint graph, where nodes are variables and edges are constraints between variables. Figure 2b shows the graph for our example decentralized channel allocation problem and figure 2d shows the cost of each pair of variables for a COP. If the problem is a CSP, we assume that the constraint is satisfied when its cost is a finite amount. We will use this example throughout this article.

CSP Algorithms

There are two classes of CSP algorithms, namely inference algorithms and search algorithms. Inference algorithms repeatedly simplify the CSP into an equivalent CSP that is typically easier to solve while search algorithms enumerate the space of possible value assignments. We will describe complete and incomplete algorithms that have DCSP counterparts for each class. Table 1 shows the algorithms described in this article.

Complete CSP Inference Algorithms

We describe bucket elimination (BE) (Dechter 1999) as a representative algorithm of this class. It first constructs a pseudotree, which is a spanning tree of the constraint graph with the property that edges in the constraint graph connect a vertex with one of its ancestors or descendants in the con-

Constraint reasoning models	CSP	COP	DCSP	DCOP
Complete inference algorithms	BE	BE	DPOP	DPOP
Incomplete inference algorithms	AC-3	W-AC3	DisAC-9	Distributed SAC
Complete search algorithms	BT	AOBB	ABT	ADOPT
Incomplete search algorithms	BA	BA	DBA	DBA

Table 1. Centralized and Distributed Constraint-Reasoning Algorithms.

v_1	v_2	v_3	v_1	v_2	v_1	v_2	v_1
a	a	b	a	a	**a**	**b**	**a**
a	a	c	**a**	**b**	b	a	b
a	**b**	**c**	b	a	c	a	c
b	a	c	b	b			
b	b	c	c	a			
c	a	b					
a			**b**		**c**		**d**

Table 2. CSP Variable Elimination Phase.

a. $f_{13} \bowtie f_{23}$. b. $(f_{13} \bowtie f_{23}) \Downarrow v_3$. c. $f_{12} \bowtie (f_{13} \bowtie f_{23}) \Downarrow_3$. d. $(f_{12} \bowtie (f_{13} \bowtie f_{23}) \Downarrow v_3) \Downarrow v_2$

straint tree (Freuder and Quinn 1985, Bayardo and Miranker 1995). An edge of the constraint graph that is not part of the pseudotree is called a backedge. Figure 2c shows a pseudotree for our example problem, where the dotted line is a backedge. The algorithm then starts the variable elimination phase by repeatedly eliminating a variable upwards from the leaves of the pseudotree. It eliminates a variable by substituting the constraints involving the eliminated variable with new constraints that do not involve that variable. Tables 2a and 2b show the two steps of the algorithm eliminating variable v_3. It first joins the constraints that involve variable v_3, namely constraints f_{13} and f_{23}, by taking all combinations of pairs of values with the same value for variable v_3, resulting in constraint $f_{13} \bowtie f_{23}$. For example, the pairs of values ($v_1 = a$, $v_3 = b$) from constraint f_{13} and ($v_2 = a$, $v_3 = b$) from constraint f_{23} are combined into the tuple ($v_1 = a$, $v_2 = a$, $v_3 = b$) in constraint $f_{13} \bowtie f_{23}$. The algorithm then projects that constraint down on variable v_3 by removing the values of variable v_3 from the constraint, resulting in constraint $(f_{13} \bowtie f_{23}) \Downarrow v_3$.

This phase continues until every variable except the root of the pseudotree is eliminated, at which point the algorithm starts the variable assignment phase by repeatedly assigning values to variables downwards from the root. The algorithm assigns the root a value from its set of values that do not violate any constraint and breaks ties randomly. In our example, variable v_1 is assigned the value *a,* which is shown in boldface in table 2d. The child of the root is then assigned a value from its set of values given the value assignment of the root. In our example, variable v_2 is assigned the value *b,* which is shown in boldface in table 2c. This phase continues until all the leaves are assigned values given the values of all their ancestors in the pseudotree, at which point the CSP is solved.

Incomplete CSP Inference Algorithms

We describe arc-consistency algorithm #3 (AC-3) (Mackworth 1977), which simplifies a CSP into an arc-consistent CSP, as a representative algorithm of this class. A CSP is arc consistent if every value in the domain of every variable is viable. A value of a variable is viable iff there exists a support for that value with respect to each binary constraint involving that variable. Pairs of values are supports for each other with respect to the constraint between their respective variables if they satisfy that constraint. Therefore, arc-consistent CSPs typically have variables with smaller domain sizes, which makes them easier to solve. If the domain of

v_1	v_2	v_3	Cost
a	a	b	0
a	a	c	0
a	**b**	**c**	**0**
b	a	c	5
b	b	c	5
c	a	b	10

a

v_1	v_2	Cost
a	a	0
a	**b**	**0**
b	a	5
b	b	5
c	a	10

b

v_1	v_2	Cost
a	**b**	**0**
b	a	10
c	a	20

c

v_1	Cost
a	**0**
b	10
c	20

d

Table 3. COP Variable Elimination Process.

a. $f_{13} \bowtie f_{23}$. b. $(f_{13} \bowtie f_{23}) \Downarrow v_3$. c. $f_{12} \bowtie (f_{13} \bowtie f_{23}) \Downarrow v_3$. d. $(f_{12} \bowtie (f_{13} \bowtie f_{23}) \Downarrow v_3) \Downarrow v_2$

one variable in an arc-consistent CSP is empty, then the CSP does not have an optimal solution. If the domains of all variables contain exactly one value, then an assignment of those values to their respective variables is a unique optimal solution for the CSP. Otherwise, there are multiple optimal solutions for the CSP and a complete algorithm is needed to find one.

Complete CSP Search Algorithms

We describe a depth-first branch-and-bound algorithm called Backtracking (BT) as a representative algorithm of this class. Like BE, it also operates on a pseudotree. It sequentially assigns values to variables downwards from the root of the pseudotree until no value can be assigned to a variable without violating a constraint. The algorithm then backtracks, that is, changes the value of the most recently assigned variable. This process continues until it has assigned a value to all variables, in which case it has found an optimal solution, or it has tried all possible combination of values, in which case it reports that there are no optimal solutions.

Incomplete CSP Search Algorithms

We describe a hill-climbing algorithm called the Breakout Algorithm (BA) (Morris 1993) as a representative algorithm of this class. It starts with an initial assignment of values to all variables and an initial weight of one to all constraints. It uses the sum of the weights of all violated constraints in a solution as the evaluation of the solution. If the initial assignment violates some constraint, then the algorithm chooses a variable and changes its value such that the evaluation of the solution is reduced. This process continues until the evaluation of the solution is zero, in which case it has found an optimal solution, or the time limit of the algorithm is reached, in which case it returns the best solution found. When the algorithm detects that it is in a local minima, it increases the weight of all violated constraints by one so that the evaluation of the current solution is larger than its neighboring solutions.

COP Algorithms

Like CSP algorithms, there are also two classes of COP algorithms, namely inference algorithms and search algorithms. We will describe complete and incomplete algorithms that have DCOP counterparts for each class.

Complete COP Inference Algorithms

We describe an extension of BE (Dechter 1999), which we described earlier, as a representative algorithm of this class. Like BE, its extension also eliminates variables upwards from the leaves of the pseudotree in the variable elimination phase and assigns values to variables downwards from the root of the pseudotree in the variable assignment phase.

The difference in the variable elimination phase is that each new constraint that substitutes the constraints involving the eliminated variable now has costs. Tables 3a and 3b show the two steps of the algorithm eliminating variable v_3. It first joins the constraints that involve variable v_3, namely constraints f_{13} and f_{23}, resulting in constraint $f_{13} \bowtie f_{23}$. The cost in each row is the sum of the corresponding costs in constraints f_{13} and f_{23}. For example, the cost of $(v_1 = c, v_2 = a, v_3 = b)$ is the sum of the cost of $(v_1 = c, v_3 = b)$ in constraint f_{13} and the cost of $(v_2 = a, v_3 = b)$ in constraint f_{23}. It then projects that constraint down on variable v_3 by removing the values of variable v_3 from the constraint, resulting in constraint $(f_{13} \bowtie f_{23}) \Downarrow v_3$. The cost in each row is the minimum over all corresponding costs in the preprojected constraint. For example, the cost of $(v_1 = a, v_2 = a)$ is the minimum over the costs of $(v_1 = a, v_2 = a, v_3 = b)$ and $(v_1 = a, v_2 = a, v_3 = c)$ in the unprojected constraint.

The difference in the variable assignment phase is that each variable is now assigned a value such that the cost is minimized given the values of all its ancestors. In our example, variable v_1 is assigned the value a, which has the minimal cost, as shown

in table 3d. This phase continues until all the leaves are assigned values given the values of all their ancestors in the pseudotree, at which point the COP is solved.

Incomplete COP Inference Algorithms

We describe an extension of AC-3, which we described earlier, called weighted AC-3 (W-AC3) (Larrosa and Schiex 2004), as a representative algorithm of this class. Like AC-3, W-AC3 also simplifies a COP into a soft arc-consistent COP. The definition of soft arc consistency is similar to the definition of arc consistency with the exception of the definition of supports. In COPs, members in a pair of values are supports for each other with respect to the constraint between their respective variables if the cost of that pair of values in the constraint is finite (Cooper and Schiex 2004, Larrosa and Schiex 2004).

Complete COP Search Algorithms

We describe an extension of BT, which we described earlier, called AND/OR Branch-and-Bound (AOBB) (Marinescu and Dechter 2009), as a representative algorithm of this class. Like BT, AOBB also assigns values to variables in a depth-first manner. The difference is that it now maintains lower and upper bounds and uses them to backtrack. The lower bound is the sum of costs of all constraints involving variables with assigned values, and the upper bound is cost of the best solution found so far. The algorithm backtracks when the lower bound is no smaller than the upper bound.

Incomplete COP Search Algorithms

We describe a simple extension of BA (Morris 1993), which we described earlier, as a representative algorithm of this class. Like BA, its extension also starts with an initial assignment of values to all variables and repeatedly changes the value of a variable such that the evaluation of the solution is reduced. The difference is that it now uses the sum of constraint costs in a solution as the evaluation of the solution.

Distributed Constraint Reasoning

There are two types of distributed constraint-reasoning models, namely distributed CSPs (DCSPs) and distributed COPs (DCOPs), which extend CSPs and COPs, respectively. Both DCSPs and DCOPs are defined by a tuple $\langle A, V, D, F, \alpha \rangle$, where V, D, and F are sets of variables, domains, and constraints, respectively, like in CSPs and COPs. $A = \{a_i, \ldots\}$ is a finite set of agents and α: $V \rightarrow A$ is a function that maps each variable to an agent that owns it. In this article, we assume that each agent owns exactly one variable. Thus, we will use the terms *agents* and *variables* interchangeably. Each agent is responsible for assigning values to its variable based only on its knowledge of the constraints involving its variable and messages that it can exchange with other agents. These messages can be delayed by a finite amount of time but are never lost.

DCSP Algorithms

Like CSP algorithms, there are also two classes of DCSP algorithms, namely inference and search algorithms. We will describe one complete and one incomplete algorithm for each class.

Complete DCSP Inference Algorithms

We describe an extension of BE, which we described earlier, called Distributed Pseudotree Optimization Procedure (DPOP) (Petcu and Faltings 2005b), as a representative algorithm of this class. DPOP was actually designed to solve DCOPs, but it can be used to solve DCSPs as well. Like BE, DPOP also operates on a pseudotree. Thus, it first calls existing distributed pseudotree construction algorithm like Distributed DFS (Hamadi, Bessière, and Quinqueton 1998) to construct a pseudotree.

At a high level, the key ideas of DPOP are as follows: When BE eliminates a variable, it knows the projected constraint because it is a centralized algorithm. Unfortunately, each agent in DPOP is only aware of the constraints in the original problem that it is involved in. Thus, each agent sends UTIL messages containing its projected constraint after it eliminates its variable. When BE assigns a value to a variable, it also knows the good values for the current unassigned variable, that is, the values that do not violate any constraints given the value assignment of all its ancestors. Unfortunately, each agent in DPOP is only aware of its own value assignment and it thus does not know its good values. Thus, each agent sends VALUE messages containing its value after it assigns itself.

We now describe DPOP in more detail. There are two phases in the operation of DPOP. The first phase, called the UTIL phase, is similar to the CSP variable-elimination phase. The difference now is that each agent eliminates its own variable and sends a UTIL message containing the projected constraint up to its parent agent. In our example, agent v_3 performs the join and projection operations as shown in tables 2a and 2b and sends a UTIL message containing the projected constraint to its parent agent v_2. This phase continues until the root agent receives a projected constraint from each of its child agents in the pseudotree.

At the end of the variable elimination phase, DPOP starts the second phase, called the VALUE phase, which is similar to the CSP variable assignment phase. The difference now is that each agent assigns itself a value and sends a VALUE message containing its value down to each of its descendant agents that it shares a constraint with. The root agent starts this phase by assigning itself a value from its set of values that do not violate any

constraint. In our example, agent v_1 assigns itself the value a as shown in table 2d and sends a VALUE message containing its value to its child agent v_2. Upon receipt of the message, agent v_2 assigns itself a value from its set of values given that agent v_1 assigned itself the value in the VALUE message. In our example, agent v_2 assigns itself the value b given that agent v_1 assigned itself the value a. This phase continues until the leaf agents assign themselves values given the values of all their ancestor agents in the pseudotree, at which point the DCSP is solved.

The number of messages sent between agents in DPOP is linear in the number of agents. However, the size of the messages and the memory requirement of each agent are exponential in the induced width of the DCOP.

Incomplete DCSP Inference Algorithms

There exist distributed arc-consistency-based algorithms, such as distributed arc-consistency algorithm #9 (DisAC-9) (Hamadi 2002). Like AC-3, DisAC-9 also simplifies a DCSP into an arc-consistent DCSP. The difference is that this simplification is now done in a distributed manner; each agent knows only about the constraints involving its variable and must thus communicate with neighboring agents to exchange information.

Complete DCSP Search Algorithms

A classic complete search algorithm for DCSP is an extension of BT, which we described earlier, called asynchronous backtracking (ABT) (Yokoo et al. 1992, 1998). Like BT, ABT also assigns values to variables in a depth-first manner and backtracks when a constraint is violated. The difference is that they are done in a distributed and asynchronous manner.

At a high level, the key ideas of ABT are as follows: When BT assigns a value to a variable, it knows the current value assignment of all variables because it is a centralized algorithm and it thus knows the good values for the current unassigned variable, that is, the values that do not violate any constraints given the value assignment of all its ancestors. Unfortunately, when ABT assigns a value to a variable, each agent is only aware of its own value assignment and it thus does not know its good values. Thus, each agent sends OK? messages containing its value after it assigns itself. When BT backtracks, it can also infer the cause of the constraint violations because it knows the current value assignment of all variables. Unfortunately, when ABT backtracks, the agent to which ABT backtracks does not know the cause of the constraint violations since it does not know the assumed value assignments of the ancestor agents of the backtracking agent. Thus, each agent sends NOGOOD messages containing its assumed value assignments of its ancestor agents when it backtracks.

We now describe ABT in more detail. Each agent maintains an agent view, which is a set of agent-value pairs that represents the agent's assumption of its ancestor agents' assigned values. The algorithm starts by having the root agent assign itself a value and sends an OK? message containing its value to each of its descendant agents that it shares a constraint with. Upon receipt of an OK? message, the receiving agent updates its agent view to reflect the latest value of the sending agent. If there exists a value in its domain that does not violate any constraint given that its ancestors are assigned values according to its agent view, it assigns itself that value and sends an OK? message to each of its descendant agents that it shares a constraint with. If no such value exists, then it backtracks by sending a NOGOOD message containing its agent view to an ancestor agent. Upon receipt of a NOGOOD message, if they are compatible, that is, they agree on all common agent-value pairs, then the receiving agent changes its value to a different value that does not violate any constraint given the assumption that agents are assigned values according to their own agent view and the agent view in the NOGOOD message. If they are incompatible, then the receiving agent backtracks. This process continues until either all agents are assigned values and no NOGOOD messages are sent, at which point the algorithm has found an optimal solution, or the root agent receives a NOGOOD message for each of its values, at which point the algorithm can conclude that there is no optimal solution for the problem.

The number of messages sent between agents in ABT is exponential in the number of agents. However, the size of the messages and the memory requirement of each agent are linear in the number of agents. Researchers have developed various extensions of this algorithm including extensions with dynamic ordering of agents in the pseudotree (Yokoo 1995; Zivan and Meisels 2005).

Incomplete DCSP Search Algorithms

A classic incomplete DCSP search algorithm is an extension of BA, which we described earlier, called the distributed breakout algorithm (DBA) (Hirayama and Yokoo 2005). Like BA, DBA also starts with an initial assignment of values to all variables and an initial weight of one to all constraints. The difference is that this assignment is now done in a distributed manner; each agent assigns itself a value randomly, sends that value in OK? messages to its neighboring agents and assigns a weight of one to each constraint that it is involved in. Additionally, each agent now sums up the weights of violated constraints that it is involved in only and uses that as the evaluation of its value. Each agent also calculates its gain, that is, the possible improvement in the evaluation if it changes its value, and sends an IMPROVE message containing

that gain to each of its neighboring agents. To guarantee that the (global) solution improves, only the agent with the highest gain amongst its neighboring agents can change its value. However, note that two nonneighboring agents can change their values concurrently. Researchers have developed various extensions of this algorithm including extensions where any agent with a positive gain can change its value with a given probability (Fitzpatrick and Meertens 2003; Zhang and Wittenburg 2002).

DCOP Algorithms

Like COP algorithms, there are also two classes of DCOP algorithms, namely inference and search algorithms. We will describe one complete and one incomplete algorithm for each class.

Complete DCOP Inference Algorithms

As BE can be used to solve both CSPs and COPs, DPOP, which we described earlier for solving DCSPs, can also be used to solve DCOPs. There are the same two phases in the operation of the version of DPOP that solves DCOPs, where the UTIL phase is similar to the COP variable-elimination phase and the VALUE phase is similar to the COP variable-assignment phase.

The number and size of messages and the memory requirement of each agent in this version of DPOP are the same as those in the version that solves DCSPs. Researchers have developed various extensions of this algorithm including extensions that trade between memory requirement and computation time (Petcu and Faltings 2007a; Petcu, Faltings, and Mailler 2007) or solution optimality (Petcu and Faltings 2005a) and an extension that speeds up DPOP by using function filtering (Brito and Meseguer 2010).

Incomplete DCOP Inference Algorithms

There exist distributed versions of arc-consistency-based algorithms called distributed soft arc-consistency (SAC) algorithms (Matsui et al. 2009; Gutierrez and Meseguer 2010). Like W-AC3, these algorithms also simplify a DCOP into a soft arc-consistent DCOP. The difference is that this simplification is now done in a distributed manner; each agent knows only about the constraints involving its variable and must thus communicate with neighboring agents to exchange information.

Complete DCOP Search Algorithms

A classic complete DCOP search algorithm is called Asynchronous Distributed Optimization (ADOPT) (Modi et al. 2005). ADOPT can be considered as a variation of AOBB, but it was developed before it. Like AOBB, ADOPT also uses lower and upper bounds to backtrack. The difference is that it is now done in a distributed and asynchronous manner.

At a high level, the key ideas of ADOPT are similar to those of ABT: Each agent sends VALUE messages (instead of OK? messages) containing its value after it assigns itself and it sends COST messages (instead of NOGOOD messages) containing its assumed value assignments of its ancestor agents when it backtracks. The difference is that COST messages also include the lower and upper bounds in addition to the assumed value assignments of the ancestor agents.

We now describe ADOPT in more detail. Each agent v_i maintains its current value d_i; its current context X_i, which is a set of agent-value pairs that represents the agent's assumption of the current values of its ancestor agents; the lower and upper bounds LB_i and UB_i, which are bounds on the optimal cost OPT_i given that its ancestors take on their respective values in X_i; the lower and upper bounds $LB_i(d)$ and $UB_i(d)$ for all values $d \in D_i$, which are bounds on the optimal costs $OPT_i(d)$ given that agent v_i takes on the value d and its ancestor agents take on their respective values in X_i; and the lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ for all values $d \in D_i$ and child agents v_c, which are its assumption on the bounds LB_c and UB_c of its child agents v_c with context $X_i \cup (v_i, d)$. The optimal costs are calculated using:

$$OPT_i(d) = \delta_i(d) + \sum_{v_c \in C_i} OPT_c$$

$$OPT_i = \min_{d \in D_i} OPT_i(d)$$

for all values $d \in D_i$, where C_i is the set of child agents of agent v_i and $\delta_i(d)$ is the sum of the costs of all cost constraints between agent v_i and its ancestor agents given that agent v_i takes on the value d and the ancestors take on their respective values in X_i.

At the start, each agent v_i initializes its current context X_i to $\emptyset$, lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ to 0 and ∞, respectively. For all values $d \in D_i$ and all child agents v_c, agent v_i calculates the remaining lower and upper bounds and takes on its best value using:

$$\delta_i(d) = \sum_{(v_j, d_j) \in X_i} f_{ij}(d, d_j)$$

$$LB_i(d) = \delta_i(d) + \sum_{v_c \in C_i} lb_i^c(d)$$

$$UB_i(d) = \delta_i(d) + \sum_{v_c \in C_i} ub_i^c(d)$$

$$LB_i = \min_{d \in D_i} \{LB_i(d)\}$$

$$UB_i = \min_{d \in D_i} \{UB_i(d)\}$$

$$d_i = \arg\min_{d \in D_i} \{LB_i(d)\}$$

Agent v_i sends a VALUE message containing its value d_i to each of its descendant agents that it shares a constraint with. It also sends a COST message containing its context X_i and its bounds LB_i and UB_i to its parent agent. Upon receipt of a VALUE message, if its current context X_i is compatible

with the value in the VALUE message, it updates its context to reflect the new value of its ancestor agent and reinitializes its lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$. Upon receipt of a COST message from child agent v_c, if its current context X_i is compatible with the context in the message, then it updates its lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ to the lower and upper bounds in the message, respectively. Otherwise, the COST message is discarded. After processing either message, it recalculates the remaining lower and upper bounds and takes on its best value using the above equations and sends VALUE and COST messages. This process repeats until the root agent v_r reaches the termination condition $LB_r = UB_r$, which means that it has found the optimal cost.

The number and size of messages and the memory requirement of each agent in ADOPT are the same as those in ABT. Researchers have developed various extensions of this algorithm including an extension that incorporates elements of ABT like NOGOOD messages (Silaghi and Yokoo 2009), an extension that uses depth-first search instead of best-first search (Yeoh, Felner, and Koenig 2010), an extension that trades between computation time and solution optimality (Yeoh, Sun, and Koenig 2009), and an extension that trades between memory requirement and computation time (Matsui, Matsuo, and Iwata 2005; Yeoh, Varakantham, and Koenig 2009).

Incomplete DCOP Search Algorithms

As BA can be used to solve both CSPs and COPs, DBA, which we described earlier for solving DCSPs, can also be used to solve DCOPs. Like the version that solves DCSPs, each agent in this version also assigns itself a value randomly and sends an OK? message containing that value to each of its neighboring agents, calculates its gain from changing values, sends an IMPROVE message containing that gain to each of its neighboring agents, and changes its value if it has the highest gain amongst its neighboring agents. The difference is that each agent now uses the cost of a constraint as the weight of that constraint in the evaluation of its current value.

Recent Extensions

Researchers have extended DCR models and algorithms in various directions to more accurately model and solve real-world problems. We now describe some of these extensions.

Privacy-Preserving Algorithms

One of the motivations for modeling problems like the distributed scheduling of meetings with the DCR model is the preservation of privacy. For example, two users that do not need to schedule a meeting between them should not have access to each other's meeting time preferences. Unfortunately, privacy loss is often unavoidable because the agents exchange messages that include aggregated constraint cost information. The exception is if the agents use obfuscation or encryption methods (Yokoo, Suzuki, and Hirayama 2005; Faltings, Léauté, and Petcu 2008; Léauté and Faltings 2009, 2011). Researchers have thus introduced several metrics for measuring privacy loss in DCR algorithms (Greenstadt, Pearce, and Tambe 2006) as well as extensions of DCR algorithms that preserve more privacy (Greenstadt, Grosz, and Smith 2007; Greenstadt 2009).

Dynamic DCR models

Many multiagent coordination problems like the distributed management of power distribution networks occur in dynamically changing environments. For example, power lines might fail or power requirements of a district might change. Researchers have thus extended DCR models to dynamic DCR models like dynamic DCSPs (Mailler 2005; Omomowo, Arana, and Ahriz 2008) and dynamic DCOPs (Petcu and Faltings 2005c; Sultanik, Lass, and Regli 2009; Yeoh et al. 2011). A typical model of a dynamic DCR problem is a sequence of (static) DCR problems with changes from one DCR problem to the next one in the sequence. The advantage of this approach is that solving a dynamic DCR problem is no harder than solving multiple (static) DCR problems. Other related extensions include a continuous-time model where agents have deadlines to choose their values (Petcu and Faltings 2007b) and a model where agents can have imperfect knowledge about their environment (Lass, Sultanik, and Regli 2008).

Multiobjective DCR Models

Many multiagent coordination problems like the distributed planning of truck routes can have multiple objectives that need to be optimized. For example, two possible conflicting objectives might be the length of the route and the financial cost of the route (due to tolls); short routes are expensive and long routes are cheap. Researchers have thus extended DCOPs to multiobjective DCOPs, where each pair of values in a constraint has multiple costs, one for each objective (Delle Fave et al. 2011). The goal in such a problem is to find a Pareto-optimal solution. Other related extensions include resource-constrained DCOPs, where each pair of values in a constraint has an additional resource cost (Bowring, Tambe, and Yokoo 2006; Matsui et al. 2008). The goal in such a problem is to find an optimal solution among the set of solutions whose sum of resource costs is within an upper bound.

Quantified DCR Models

Many multiagent coordination problems like the distributed tracking of targets in a sensor network can include adversarial agents. For example, the target might try to avoid detection and is thus adversarial. Researchers have thus extended DCR

models to quantified DCR models (Baba et al. 2010, Matsui et al. 2010). Quantified DCR models allow some agents in the problem to be adversarial, that is, they can assign themselves any value regardless whether the value will increase or decrease the overall solution cost. The goal in such a problem is to find an optimal solution given that all the adversarial agents will assign themselves the worst possible values, which increase the overall solution cost the most. Other related extensions include an algorithm that explicitly models the deception of adversarial agents (Lisy et al. 2010).

Other Models

Aside from distributed constraint-reasoning models, researchers have developed other distributed problem-solving models to capture different characteristics of common distributed problems. We now describe some of these models.

DEC-POMDPs

Partially observable Markov decision processes (POMDPs) (Smallwood and Sondik 1973) have been shown to be popular models for modeling centralized sequential decision making under uncertainty problems. As a result, decentralized POMDPs (DEC-POMDPs) (Bernstein et al. 2002) have emerged as a natural extension for modeling decentralized sequential decision making under uncertainty problems. An example application is the cooperative multirobot navigation problem, where the robots (agents) need to cooperatively get to their respective goal locations. They have actuators for locomotion, but their movements depend on factors like wheel alignment and slippage. They also have sensors for sensing their environment, but their sensory data depend on factors like sensor noise. Despite the uncertainty, they need to find a sequence of actions that will result in the largest likelihood of reaching their respective goal locations.

DCR models are ill-suited for modeling this problem as they do not take uncertainty into account. Thus, a DEC-POMDP is a better-suited model in such a problem. Unfortunately, the increased richness of DEC-POMDPs compared to DCR models comes at a price of higher complexity — finding optimal DEC-POMDP solutions is NEXP-hard (Bernstein et al. 2002). Researchers have thus proposed specialized models that leverage sparse agent interactions to improve the scalability of DEC-POMDP algorithms (Nair et al. 2005; Kumar and Zilberstein 2009; Oliehoek et al. 2008; Velagapudi et al. 2011).

Auctions

Auction-based approaches are also popular for modeling decentralized coordination problems (Koenig, Keskinocak, and Tovey 2010). In an auction-based algorithm, the auctioneer calls for bids for tasks/resources and the agents bid according to their valuation of the tasks and their capabilities of performing those tasks. Generally, each agent maximizes its own payoff, which is defined as its income (from performing its task) minus its cost (of performing its task). Thus, the primary difference between auctions and the DCR models is that auction-based algorithms expect that coordinated behaviors of agents emerge through the individual optimization of each agent. However, unlike game-theoretic approaches, we usually assume that the agents always bid truthfully independent of the game mechanism. Auction-based approaches have been applied to solve the distributed processing of streaming data (An, Douglis, and Ye 2008), the distributed allocation of resources in cloud computing centers (An et al. 2010), and the distributed allocation of roles in RoboCup soccer (Frías-Martínez, Sklar, and Parsons 2004).

Conclusions

In this article, we gave a brief overview of distributed problem solving. Due to space limitations, we focused on distributed constraint-reasoning (DCR) models, which are problems where each agent has a fixed, finite set of possible actions, and the agents try to find a combination of individual actions that satisfies or optimizes some global criteria. Distributed problem solving has a long research history and the topics covered in this article are very limited. We encourage readers to refer to other articles that discuss other issues in distributed problem solving (Lesser and Corkill 1981; Davis and Smith 1983; Durfee 1999). Researchers have developed several testbeds (Ezzahir et al. 2007; Sultanik, Lass, and Regli 2007; Léauté, Ottens, and Szymanek 2009) for running DCR algorithms including DisChoco[3], DCOPolis[4], and FRODO[5]. These testbeds include implementations for a variety of popular DCR algorithms (including most of the algorithms described in this article), evaluation domains and evaluation metrics. We encourage readers who are interested in distributed constraint reasoning to download and try out these testbeds.

Notes

1. Researchers have also used DisCSPs to refer to distributed CSPs and DCSPs to refer to dynamic CSPs.

2. COPs are also known as weighted CSPs (Schiex, Fargier, and Verfaillie 1995; Bistarelli et al. 1999).

3. www.lirmm.fr/coconut/dischoco.

4. www.dcopolis.org.

5. sourceforge.net/projects/frodo2.

References

An, B.; Douglis, F.; and Ye, F. 2008. Heuristics for Negotiation Schedules in Multiplan Optimization. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 551–558. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

An, B.; Lesser, V.; Irwin, D.; and Zink, M. 2010. Automated Negotiation with Decommitment for Dynamic Resource Allocation in Cloud Computing. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems,* 981–988. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Baba, S.; Iwasaki, A.; Yokoo, M.; Silaghi, M.; Hirayama, K.; and Matsui, T. 2010. Cooperative Problem Solving Against Adversary: Quantified Distributed Constraint Satisfaction Problem. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*, 781–788. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Bayardo, R., and Miranker, D. 1995. On the Space-Time Trade-Off in Solving Constraint Satisfaction Problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence,* 558–562. San Francisco: Morgan Kaufmann Publishers.

Bernstein, D.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research* 27(4): 819–840.

Bistarelli, S.; Montanari, U.; Rossi, F.; Schiex, T.; Verfaillie, G.; and Fargier, H. 1999. Semiring-Based CSPs and Valued CSPs: Basic Properties and Comparison. *Constraints* 4(3): 199–240.

Bowring, E.; Tambe, M.; and Yokoo, M. 2006. Multiply-Constrained Distributed Constraint Optimization. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1413–1420. New York: Association for Computing Machinery.

Brito, I., and Meseguer, P. 2010. Improving DPOP with Function Filtering. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems,* 141–158. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Clifford, P., and Leith, D. 2007. Channel Dependent Interference and Decentralized Colouring. In *Proceedings of the International Conference on Network Control and Optimization,* Lecture Notes in Computer Science Volume 4465, 95–104. Berlin: Springer.

Cooper, M., and Schiex, T. 2004. Arc Consistency for Soft Constraints. *Artificial Intelligence* 154(1-2): 199–227.

Davis, R., and Smith, R. 1983. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence* 20(1): 63–109.

Dechter, R. 1999. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence* 113(1–2): 41–85.

Delle Fave, F.; Stranders, R.; Rogers, A.; and Jennings, N. 2011. Bounded Decentralised Coordination over Multiple Objectives. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multiagent Systems,* 371–378. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Durfee, E. 1999. Distributed Problem Solving and Planning. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,* ed. G. Weiss. Cambridge, MA: The MIT Press. 121–164.

Ezzahir, R.; Bessière, C.; Belaissaoui, M.; and Bouyakhf, E. H. 2007. DisChoco: A Platform for Distributed Constraint Programming. Paper presented at the 8th International Workshop on Distributed Constraint Reasoning. Hyderabad, India, 8 January.

Faltings, B.; Léauté, T.; and Petcu, A. 2008. Privacy Guarantees Through Distributed Constraint Satisfaction. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology,* 350–358. Los Alamitos, CA: IEEE Computer Society.

Fitzpatrick, S., and Meertens, L. 2003. Distributed Coordination Through Anarchic Optimization. In *Distributed Sensor Networks: A Multiagent Perspective,* ed. V. Lesser, C. Ortiz, and M. Tambe, 257–295. Dordrecht, The Netherlands: Kluwer.

Freuder, E., and Quinn, M. 1985. Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence,* 1076–1078. Los Altos, CA: William Kaufmann, Inc.

Frías-Martínez, V.; Sklar, E.; and Parsons, S. 2004. Exploring Auction Mechanisms for Role Assignment in Teams of Autonomous Robots. In *RoboCup 2004: Robot Soccer World Cup VIII*. Lecture Notes in Computer Science 3276, 532–539. Berlin: Springer.

Greenstadt, R. 2009. An Overview of Privacy Improvements to K-Optimal DCOP Algorithms (Extended Abstract). In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1279–1280. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Greenstadt, R.; Grosz, B.; and Smith, M. 2007. SSDPOP: Improving the Privacy of DCOP with Secret Sharing. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1098–1100. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Greenstadt, R.; Pearce, J.; and Tambe, M. 2006. Analysis of Privacy Loss in DCOP Algorithms. In *Proceedings of the 21st National Conference on Artificial Intelligence,* 647–653. Menlo Park, CA: AAAI Press.

Gutierrez, P., and Meseguer, P. 2010. BnB-ADOPT+ with Several Soft Arc Consistency Levels. In *Proceedings of the 29th European Conference on Artificial Intelligence,* 67–72. Amsterdam: IOS Press.

Hamadi, Y. 2002. Optimal Distributed Arc-Consistency. Constraints 7(3-4): 367–385.

Hamadi, Y.; Bessière, C.; and Quinqueton, J. 1998. Distributed Intelligent Backtracking. In *Proceedings of the 13th European Conference on Artificial Intelligence,* 219–223. Chichester, UK: John Wiley and Sons.

Hirayama, K., and Yokoo, M. 2005. The Distributed Breakout Algorithms. *Artificial Intelligence* 161(1-2): 89–115.

Koenig, S.; Keskinocak, P.; and Tovey, C. 2010. Progress on Agent Coordination with Cooperative Auctions. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence,* 1713–1717. Menlo Park, CA: AAAI Press.

Kumar, A., and Zilberstein, S. 2009. Constraint-Based

Dynamic Programming for Decentralized POMDPs with Structured Interactions. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems,* 561–568. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Kumar, A.; Faltings, B.; and Petcu, A. 2009. Distributed Constraint Optimization with Structured Resource Constraints. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems,* 923–930. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Larrosa, J., and Schiex, T. 2004. Solving Weighted CSP by Maintaining Arc Consistency. *Artificial Intelligence* 159(1-2): 1–26.

Lass, R.; Kopena, J.; Sultanik, E.; Nguyen, D.; Dugan, C.; Modi, P.; and Regli, W. 2008. Coordination of First Responders Under Communication and Resource Constraints (Short Paper). In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems,* 1409–1413. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Lass, R.; Sultanik, E.; and Regli, W. 2008. Dynamic Distributed Constraint Reasoning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence,* 1466–1469. Menlo Park, CA: AAAI Press.

Léauté, T., and Faltings, B. 2009. Privacy-Preserving Multiagent Constraint Satisfaction. In *Proceedings of the 2009 IEEE International Conference on Privacy, Security, Risk and Trust,* 17–25. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Léauté, T., and Faltings, B. 2011. Coordinating Logistics Operations with Privacy Guarantees. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence,* 2482–2487. Menlo Park, CA: AAAI Press.

Léauté, T.; Ottens, B.; and Szymanek, R. 2009. FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization. Paper presented at the IJCAI-09 Distributed Constraint Reasoning Workshop, Pasadena, CA, 13 July.

Lesser, V., and Corkill, D. 1981. Functionally-Accurate, Cooperative Distributed Systems. *IEEE Transactions on Systems, Man and Cybernetics* SMC11(1): 81–96.

Lisy, V.; Zivan, R.; Sycara, K.; and Péchoucek, M. 2010. Deception in Networks of Mobile Sensing Agents. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1031–1038. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Mackworth, A. 1977. Consistency in Networks of Relations. *Artificial Intelligence* 8(1): 99–118.

Maheswaran, R.; Tambe, M.; Bowring, E.; Pearce, J.; and Varakantham, P. 2004. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Event Scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems,* 310–317. Piscataway, NJ: Institute of Electrical and Electronic Engineers.

Mailler, R. 2005. Comparing Two Approaches to Dynamic, Distributed Constraint Satisfaction. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1049–1056. New York: Association for Computing Machinery.

Marinescu, R., and Dechter, R. 2009. AND/OR Branch-and-Bound Search for Combinatorial Optimization in Graphical Models. *Artificial Intelligence* 173(16–17): 1457–1491.

Matsui, T.; Matsuo, H.; Silaghi, M.; Hirayama, K.; Yokoo, M.; and Baba, S. 2010. A Quantified Distributed Constraint Optimization Problem. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1023–1030. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Matsui, T.; Matsuo, H.; and Iwata, A. 2005. Efficient Methods for Asynchronous Distributed Constraint Optimization Algorithm. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications,* 727–732. Calgary, AB, Canada: International Association of Science and Technology for Development.

Matsui, T.; Silaghi, M.; Hirayama, K.; Yokoo, M.; and Matsuo, H. 2008. Resource Constrained Distributed Constraint Optimization with Virtual Variables. In *Proceedings of the 23rd AAA Conference on Artificial Intelligence,* 120–125. Menlo Park, CA: AAAI Press.

Matsui, T.; Silaghi, M.; Hirayama, K.; Yokoo, M.; and Matsuo, H. 2009. Directed Soft Arc Consistency in Pseudo Trees. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1065–1072. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Modi, P.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161(1-2): 149–180.

Morris, P. 1993. The Breakout Method for Escaping from Local Minima. In *Proceedings of the 11th National Conference on Artificial Intelligence,* 40–45. Menlo Park, CA: AAAI Press.

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In *Proceedings of the 20th National Conference on Artificial Intelligence,* 133–139. Menlo Park, CA: AAAI Press.

Oliehoek, F.; Spaan, M.; Whiteson, S.; and Vlassis, N. 2008. Exploiting Locality of Interaction in Factored Dec-POMDPs. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems,* 517–524. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Omomowo, B.; Arana, I.; and Ahriz, H. 2008. DynABT: Dynamic Asynchronous Backtracking for Dynamic DisC-SPs. In *Proceedings of the 13th International Conference on Artificial Intelligence: Methodology, Systems, and Applications,* Lecture Notes in Computer Science 5253, 285–296. Berlin: Springer.

Petcu, A., and Faltings, B. 2005a. Approximations in Distributed Optimization. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming,* Lecture Notes in Computer Science 3709, 802–806. Berlin: Springer.

Petcu, A., and Faltings, B. 2005b. A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence,* 1413–1420. San Francisco: Morgan Kaufmann Publishers.

Petcu, A., and Faltings, B. 2005c. Superstabilizing, Fault-

Containing Multiagent Combinatorial Optimization. In *Proceedings of the 20th National Conference on Artificial Intelligence,* 449–454. Menlo Park, CA: AAAI Press.

Petcu, A., and Faltings, B. 2007a. MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence,* 1452–1457. Menlo Park, CA: AAAI Press.

Petcu, A., and Faltings, B. 2007b. Optimal Solution Stability in Dynamic, Distributed Constraint Optimization. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology,* 321–327. Los Alamitos, CA: IEEE Computer Society.

Petcu, A.; Faltings, B.; and Mailler, R. 2007. PC-DPOP: A New Partial Centralization Algorithm for Distributed Optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence,* 167–172. Menlo Park, CA: AAAI Press.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence,* 631–637. San Francisco: Morgan Kaufmann Publishers.

Silaghi, M., and Yokoo, M. 2009. ADOPTing: Unifying Asynchronous Distributed Optimization with Asynchronous Backtracking. *Autonomous Agents and MultiAgent Systems* 19(2): 89–123.

Smallwood, R., and Sondik, E. 1973. The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Operations Research* 21(5): 1071–1088.

Sultanik, E.; Lass, R.; and Regli, W. 2007. DCOPolis: A Framework for Simulating and Deploying Distributed Constraint Reasoning Algorithms. Paper presented at the Eighth International Workshop on Distributed Constraint Reasoning. Hyderabad, India, 8 January.

Sultanik, E.; Lass, R.; and Regli, W. 2009. Dynamic Configuration of Agent Organizations. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence,* 305–311. Menlo Park, CA: AAAI Press.

Sycara, K.; Roth, S.; Sadeh, N.; and Fox, M. 1991. Distributed Constrained Heuristic Search. *IEEE Transactions on Systems, Man and Cybernetics* 6(21): 1446–1461.

Ueda, S.; Iwasaki, A.; and Yokoo, M. 2010. Coalition Structure Generation Based on Distributed Constraint Optimization. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence,* 197–203. Menlo Park, CA: AAAI Press.

Velagapudi, P.; Varakantham, P.; Scerri, P.; and Sycara, K. 2011. Distributed Model Shaping for Scaling to Decentralized POMDPs with Hundreds of Agents. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multiagent Systems,* 955–962. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Yeoh, W.; Felner, A.; and Koenig, S. 2010. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *Journal of Artificial Intelligence Research* 38: 85–133.

Yeoh, W.; Sun, X.; and Koenig, S. 2009. Trading Off Solution Quality for Faster Computation in DCOP Search Algorithms. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence,* 354–360. Menlo Park, CA: AAAI Press.

Yeoh, W.; Varakantham, P.; and Koenig, S. 2009. Caching Schemes for DCOP Search Algorithms. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems,* 609–616. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Yeoh, W.; Varakantham, P.; Sun, X.; and Koenig, S. 2011. Incremental DCOP Search Algorithms for Solving Dynamic DCOPs (Extended Abstract). In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multiagent Systems,* 1069–1070. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Yokoo, M. 1995. Asynchronous Weak-Commitment Search for Solving Distributed Constraint Satisfaction Problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming,* Lecture Notes in Computer Science 976, 88–102. Berlin: Springer.

Yokoo, M.; Durfee, E.; Ishida, T.; and Kuwabara, K. 1992. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In *Proceedings of the 12th International Conference on Distributed Computing Systems,* 614–621. Los Alamitos, CA: IEEE Computer Society.

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.

Yokoo, M.; Suzuki, K.; and Hirayama, K. 2005. Secure Distributed Constraint Satisfaction: Reaching Agreement Without Revealing Private Information. *Artificial Intelligence* 161(1–2): 229–245.

Zhang, W., and Wittenburg, L. 2002. Distributed Breakout Revisited. In *Proceedings of the 18th National Conference on Artificial Intelligence,* 352–357. Menlo Park, CA: AAAI Press.

Zhang, W.; Xing, Z.; Wang, G.; and Wittenburg, L. 2003. An Analysis and Application of Distributed Constraint Satisfaction and Optimization Algorithms in Sensor Networks. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems,* 185–192. New York: Association for Computing Machinery.

Zivan, R. 2008. Anytime Local Search for Distributed Constraint Optimization. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence,* 393–398. Menlo Park, CA: AAAI Press.

Zivan, R., and Meisels, A. 2005. Dynamic ordering for asynchronous backtracking on disCSPs. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming,* Lecture Notes in Computer Science 3709, 32–46. Berlin: Springer.

Zivan, R.; Glinton, R.; and Sycara, K. 2009. Distributed Constraint Optimization for Large Teams of Mobile Sensing Agents. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology,* 347–354. Los Alamitos, CA: IEEE Computer Society.

William Yeoh is an assistant professor of computer science at New Mexico State University. He received his Ph.D. in computer science at the University of Southern California. His research interests include multiagent systems, distributed constraint reasoning, heuristic search, and planning with uncertainty. He was a coorganizer of the International Workshop on Distributed Constraint Reasoning in 2008 and the AAAI Symposium on Multiagent Coordination under Uncertainty in 2011.

Makoto Yokoo is a professor of information science and electrical engineering at Kyushu University, Japan. He received a Ph.D, in information and communication from the University of Tokyo in 1995. His research interests include multiagent systems, constraint reasoning, and game theory. He is currently the president of International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS) and an AAAI fellow.