Cooperative Learning over Composite Search Spaces: Experiences with a Multi-agent Design System*

M V Nagendra Prasad[1], Susan E. Lander[2] and Victor R. Lesser[1],

[1]Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{nagendra,lesser}@cs.umass.edu

[2]Blackboard Technology Group, Inc.
401 Main Street
Amherst, MA 01002
lander@bbtech.com

Abstract

We suggest the use of two learning techniques — short term and long term — to enhance search efficiency in a multi-agent design system by letting the agents learn about non-local requirements on the local search process. The first technique allows an agent to accumulate and apply constraining information about global problem solving, gathered as a result of agent communication, to further problem solving within the same problem instance. The second technique is used to classify problem instances and appropriately index and retrieve constraining information to apply to new problem instances. These techniques will be presented within the context of a multi-agent parametric-design application called STEAM. We show that learning conclusively improves solution quality and processing-time results.

Introduction

In this article, we study machine-learning techniques that can be applied within multi-agent systems (MAS) to improve solution quality and processing-time results. A ubiquitous problem with multi-agent systems that use cooperative search techniques is the "local perspective" problem. Constraining information is distributed across the agent set but each individual agent perceives a search space bounded only by its local constraints rather than by the constraints of all agents in the system. This problem could be easily addressed if all expertise could be represented in the form of explicit constraints: the constraints could be collected and processed by a centralized constraint-satisfaction algorithm. However, the most compelling reasons for building multi-agent systems make it unlikely that the agents are that simple. More commonly, agents are complex systems in their own right and their expertise is represented by a combination of declarative and procedural knowledge that cannot be captured as a set of explicit constraints. Therefore, each agent must operate independently to solve some subproblem associated with an overall task and the individual solutions must be integrated into a globally consistent solution. An agent with only a local view of the search space cannot avoid producing subproblem solutions that conflict with other agents' solutions and cannot always make intelligent decisions about managing conflicts that do occur.

We suggest the use of two learning techniques — short term and long term — to address the local perspective problem. The first technique allows an agent to accumulate and apply constraining information about global problem solving, gathered as a result of agent communication, to further problem solving within the same problem instance. The second technique is used to classify problem instances and appropriately index and retrieve constraining information to apply to new problem instances. These techniques will be presented within the context of a multi-agent parametric-design application called STEAM.

The remainder of this article is organized as follows: We first formalize our view of distributed search spaces for multi-agent systems and briefly present the STEAM prototype application (Lander 1994; Lander & Lesser 1996). The subsequent section introduces two learning mechanisms used to enhance search efficiency. Next, we present experimental results that demonstrate the effectiveness of the learning mechanisms. In the last two sections, we discuss related work and present our conclusions.

Distributed Search Spaces

Search has been one of the fundamental concerns of Artificial Intelligence. When the entire search space is confined to a single logical entity, the search is centralized. In contrast, distributed search involves a state space along with its associated search operators and control regime partitioned across multiple agents. Lesser (Lesser 1990; 1991) recognizes distributed search as a framework for understanding a variety of issues in Distributed Artificial Intelligence(DAI) and Multi-agent Systems (MAS).

*This material is based upon work supported by the National Science Foundation under Grant Nos. IRI-9523419 and EEC-9209623. The content of this paper does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

In distributed search, multiple agents are required to synergistically search for a solution that is mutually acceptable to all of them and satisfies the constraints on global solutions. However, the constraining information is distributed across the agent set and each individual agent perceives a search space bounded only by its local constraints rather than by the constraints of all the agents in the system. Thus we need to distinguish between a *local search space* and the *composite search space.*

A local search space is private to an agent and is defined by the domain values of the parameters used to constrain the local search. A set of local constraints along with the problem specification, define the local solution space as a subset of the local search space. More formally, for each agent $\mathcal{A}_i$ the following can be defined:

- The parameter set $\mathcal{P}_i = \{ p_{ij}, 1 \leq j \leq x_i \}$ with respective domains $\mathcal{D}_{ij}$, $1 \leq j \leq x_i$ from which the parameters take their values. $\mathcal{D}_{i1} \times \mathcal{D}_{i2} \times \ldots \times \mathcal{D}_{ix_i}$ defines the local search space for agent $\mathcal{A}_i$. A domain can be a set of discrete values, real values, labels or intervals.

  A parameter is a *shared parameter* if it belongs to parameter sets of more than one agent. More formally, let $\mathcal{AS}(p)$ represent the set of agents that have parameter p as a part of their parameter set.

$$\mathcal{AS}(p) \quad = \quad \{\mathcal{A}_i \mid p \in \{p_{ij}, 1 \leq j \leq x_i\}\}$$

  "p" is a shared parameter iff $\mid \mathcal{AS}(p) \mid \; > 1$.

- Hard constraints $\mathcal{HC}_i^t = \{\mathcal{HC}_{ij}^t, 1 \leq j \leq y_i\}$ that represent the solution requirements that have to be satisfied for any local solution that agent $\mathcal{A}_i$ produces at time t.

- Soft constraints $\mathcal{SC}_i^t = \{\mathcal{SC}_{ij}^t, 1 \leq j \leq z_i\}$ that represent the solution preferences of agent $\mathcal{A}_i$ at time t. Soft constraints can be violated or modified without affecting the ability of the agents to produce globally acceptable designs.

The set of hard and soft constraints, $\mathcal{C}_i^t = \mathcal{HC}_i^t \cup \mathcal{SC}_i^t$, defines a local solution space $S_i^t \subseteq \mathcal{D}_1^i \times \mathcal{D}_2^i \times \ldots \times \mathcal{D}_{x_i}^i$.

$$S_i^t \quad = \quad Space(\mathcal{C}_i^t)$$

In constraint-optimization problems, not all constraints need to be satisfied. Hard constraints are necessarily satisfied and soft constraints are satisfied to the extent possible. Soft constraints may have a varying degree of flexibility with some being softer than the others. Some of them may be relaxed when an agent is not able satisfy all of them together. When $S_i^t = \phi$ the problem is over-constrained with respect to $\mathcal{A}_i$. In this situation, domain-specific strategies are used to relax one of the soft constraints c_{ik}.

$$S_i^{t'} = Space(\mathcal{C}_i^t - c_{ik})$$

How soft constraints are relaxed can strongly effect the system performance. Lander (Lander 1994) discusses some studies related to this issue. We will not attempt to further discuss this in the present paper.

A local solution $s_k \in S_i^t$ for agent $\mathcal{A}_i$, consists of an assignment of values $v_{ij} \in \mathcal{D}_{ij}$ to parameters p_{ij}, $1 \leq j \leq x_i$. Each of the local solutions may have utility measures attached to it to facilitate selection of a preferred solution from S_i^t. Agent $\mathcal{A}_i$ prefers solution $s_m \in S_i^t$ over solution $s_n \in S_i^t$ if the utility of s_m is more than the utility of s_n.

We also define the notion of *projection* that will be used in the later sections to formalize the description of the learning algorithms. For each local solution $s_i = \{p_{i1} = v_{i1}, p_{i2} = v_{i2}, \ldots p_{ix_i} = v_{ix_i}\} \in S_i^t$ and a set of parameters X, we can define a projection $(s_i \downarrow X)$ as follows:

$$(s_i \downarrow X) \quad = \quad \{p_{ij} = v_{ij} \mid p_{ij} \in X\}$$

For a solution space S_i^t, projection $(S_i^t \downarrow X)$ is defined as

$$(S_i^t \downarrow X) \quad = \quad \{(s_i \downarrow X) \mid s_i \in S_i^t\}$$

Constraints can be explicitly represented as in, for example, (run-speed $\leq$ 3600) or they may be implicit as in procedural representation. An example of a procedurally embedded constraint may look as follows:

```
if (run-speed <= 3000) then
    water-flow-rate
      = max (50  water-flow-rate)
end if
```

In this constraint, the `run-speed` parameter implicitly constrains the `water-flow-rate`. In complex systems like STEAM, it is often the case that such implicit constraints are not easily discernible and, even when they are, it may not be possible to represent them in a declarative form that is suitable for sharing with other agents.

Soft constraints in STEAM are associated with four levels of flexibility – 1 to 4 – with constraints at level 4 representing the highest preference. An agent tries to satisfy as many of the constraints at highest level as possible. Solutions are evaluated based on the constraint level, the number of constraints satisfied at that level, and, finally, on the cost of the design. Solutions satisfying more constraints at a higher level are preferred. If there are multiple design components satisfying the same number of constraints at the highest level, a component with least cost is chosen.

The composite search space CS is a shared search space derived from the composition of the local search spaces of the agents. The desired solution space for the multi-agent system is a subset of the composite search space. Parameters of the composite search space consist of:

$$p_{Gk} \quad \subseteq \quad \bigcup_{i=1}^{i=n} \bigcup_{j=1}^{j=x_i} p_{ij}$$

$$\mathcal{D}_{Gk} = \bigcap \mathcal{D}_{ij} \mid \mathcal{A}_i \in \mathcal{AS}(p_{Gk}) \wedge p_{ij} = p_{Gk}$$

where $1 \leq k \leq g$.

The parameter set for the composite space represents the features of the overall design that are constrained by the composite search space. The domain for a parameter in this set is derived as the intersection of all the domains of the corresponding parameters in the agents that have this parameter as a shared parameter.

STEAM: A Multi-agent Design System

STEAM is a parametric design system that produces steam condensers where a motor powers a pump that injects water into a heat-exchanger chamber. High temperature steam input into this chamber is output as condensed steam. STEAM consists of seven agents for designing the components of such steam condensers: `pump-agent`, `heat-exchanger-agent`, `motor-agent`, `platform-agent`, `vbelt-agent`, `shaft-agent`, `system-frequency-critic`

Each agent takes the responsibility to either design a component of a steam condenser for which it possesses expertise or critique an evolving partial design. STEAM is a globally cooperative system which implies that any local performance measures may be sacrificed in the pursuit of better global performance. Thus an agent is willing to produce a component that is poorly rated locally but may participate in a design of high overall quality.

STEAM is a distributed search system. Each agent performs local search based upon the implicit and explicit constraints known within its local context. It has its own local state information, a local database with static and dynamic constraints on its design components and a local agenda of potential actions. The search in STEAM is performed over a space of partial designs. A partial design represents a partial solution in the composite search space. It is initiated by placing a problem specification in a centralized shared memory that also acts as a repository for the emerging composite solutions (i.e. partial solutions) and is visible to all the agents. Any design component produced by an agent is placed in the centralized repository. Some of the agents initiate base proposals based on the problem specifications and their own internal constraints and local state. Other agents in turn extend or critique these proposals to form complete designs. The evolution of a composite solution in STEAM can be viewed as a series of state transitions. For a composite solution in a given state, an agent can play a role like initiate-design, extend-design or critique-design. An agent can be working on several composite solutions concurrently.

Learning Efficient Search

Problem solving in STEAM starts with agents possessing only local views of the search and solution spaces. Given such a limited perspective of the search space, an agent cannot avoid producing components that conflict with other agents' components. This section introduces two machine learning techniques to exploit the situations that lead to conflicts so as to avoid similar conflicts in future.

Conflict Driven Learning (CDL)

CDL has been presented as negotiated search in Lander (Lander 1994). Below, we reinterpret this process as a form of learning and provide a formal basis for the learning mechanisms. As discussed previously, purely local views that agents start out with are unlikely to lead to composite solutions that are mutually acceptable to all of them. When an agent attempts to extend or critique a partial design, it may detect that the design violates some of its local constraints. Let the set of parameters shared by agents $\mathcal{A}_i$ and $\mathcal{A}_j$ be X_{ij}. Then agent $\mathcal{A}_j$ trying to extend or critique a solution $s_i^m \in S_i^t$ detects a conflict iff

$$(s_i^m \downarrow X_{ij}) \notin (S_j^t \downarrow X_{ij})$$

The violated constraints can be either explicit or implicit. Explicit constraints can be shared and an agent detecting violations of explicit constraints generates feedback to the agents that proposed the partial design involved in the conflict. In STEAM, explicit constraints are limited to simple boundary constraints of the form $(x < n)$, $(x \leq n)$, $(x > n)$, or $(x \geq n)$ that specify maximum or minimum values for a parameter. If x is a shared parameter, then an explicit constraint on x can be shared with other agents. Such a conflict-driven exchange of feedback on non-local requirements allows each agent to develop an approximation of the composite (global) search space that includes both its local perspective and explicit constraining information that it assimilates from other agents. This type of "negotiated search" can be viewed as *learning by being told* and is short-term in nature—the exchanged information is applied only to the current problem instance. The following lemma shows that each exchange in CDL progressively improves an agent's view of the composite search space.

Let the set of constraints communicated as feedback by agent $\mathcal{A}_j$ to $\mathcal{A}_i$ at time t upon detecting a conflict be $\mathcal{FC}_j$.

Lemma: $(\mathcal{CS} \downarrow \mathcal{P}_i) \subseteq (S_i^{t'} = \text{Space}(C_i^t \cup \mathcal{FC}_j)) \subset S_i^t$

The lemma says that $\mathcal{A}_i$'s view of the composite search space with the new conflict information assimilated is a refinement over the previous one with respect to the relevant portion of the actual composite search space.

The design leading to a conflict is abandoned and the agents pursue other designs but with an enhanced knowledge of composite solution requirements from there on.

Exchange of explicit constraints does not guarantee that the agents will find a mutually acceptable solution because of the presence of implicit constraints that cannot be shared. Thus, even if all the explicit constraints that lead to conflicts are exchanged by time t_f, the resulting view of agent $\mathcal{A}_i$ of the composite search space is still an approximation of the true composite search space:

$$(\mathcal{CS} \downarrow \mathcal{P}_i) \sim S_i^{t_f}$$

However, to the extent that an agent's local view approaches the global view, an agent is likely to be more effective at proposing conflict-free proposals.

Case-Based Learning (CBL)

Agents can also learn to predict composite solution requirements based on their past problem-solving experience. We endow agents with capabilities for *Case-based learning (CBL)* to accumulate local views of the composite search space requirements across many design runs. This can be viewed as long-term learning—the learned information is available for retrieval with future problem instances.

During the learning phase, the agents perform their search with conflict-driven learning as discussed above. However, at the end of each search, an agent stores the problem specification and the non-local constraints it received as feedback from the other agents as an approximation of the non-local requirements on the composite solution space for that problem specification. After the agents learn over a sufficiently large training set, they can replace the process of assimilating feedback with learned knowledge. When a new problem instance is presented to the agent set, it retrieves the set of non-local constraints that are stored under a past problem specification that is similar to the present problem specification and adds them to the set of local requirements at the start of the search. Thus agents can avoid communication to achieve approximations of the composite search space.

$$(\mathcal{CS} \downarrow \mathcal{P}_i) \sim S_i^{t=0}$$

where $S_i^{t=0}$ is defined by the local domain constraints and the constraints of the similar past case from the case base

$$S_i^{t=0} = Space(C_i^0 + \mathcal{PC}_{1NN})$$

We use the 1-Nearest-Neighbor (1NN) algorithm based on the Euclidean metric to obtain the most similar past case.

Experimental Results

In this section, we will empirically demonstrate the merits of learning composite search-space requirements, both short-term and long-term. In order to demonstrate the merits of our learning methods, we experimented with 3 search strategies as described below:

- *Blind Search (BS)*: No learning is applied. When an agent detects a conflict in a particular design, it chooses another design to pursue. Agents do not communicate any information.
- *Conflict-Driven Learning (CDL)*: An agent that detects a conflict generates a feedback that is assimilated by the recipient agents. The recipients use the conflict information to constrain future searches within a single design run: for example, when proposing or extending alternative designs.
- *Case-Based Learning (CBL)*: The agents use previously accumulated cases to start their problem-solving with an awareness of the non-local constraints. Agents do not communicate during problem-solving. Each agent uses a 1-NN algorithm to find the case most similar to the present problem solving instance and initializes its non-local requirements with the constraints in the case. We ran the algorithm at different case base sizes: 50, 100 150, 200.

As described previously, the **STEAM** system used in these experiments had seven agents. Four of them — `pump-agent`, `heat-exchanger-agent`, `motor-agent`, and `vbelt-agent` — can either initiate a design or extend an existing partial design. `Platform-agent` and `shaft-agent` can only extend a design and `frequency-critic` always critiques a partial design. Each agent in turn, gets a chance to perform a role during a cycle. The number of cycles represents a good approximation to the amount of search performed by the entire system. Problem specification consisted of three parameters — `required-capacity`, `platform-side-length`, and `maximum-platform-deflection`. Problem solving terminates when the agents produce a single mutually acceptable design. We trained the CBL system with randomly chosen instances and then tested all the three search strategies on the same set of 100 instances different from the training instances.

Table 1 shows the average cost of designs produced by each of the algorithms. Table 2 shows the average number of cycles per design.

Wilcoxon matched-pair signed-ranks test revealed that the costs of designs produced by **STEAM** with CBL and CDL were lower than those produced by **STEAM** with blind search at significance level 0.05. The same test however, revealed no significant difference between the costs of designs produced by **STEAM** with CDL and those produced by **STEAM** with CBL.

CBL was able to produce slightly better designs than CDL because CDL performs blind search initially until it runs into conflicts and gains a better view of the composite search space through the exchange of feedback on these conflicts. CBL on the other hand, starts the problem solving with a good approximation of the global solution space requirements and hence manages to do better than CDL. Even though CDL gains a more

Blind	CDL	CBL-50	CBL-100	CBL-150	CBL-200
7227.2	6598.0	6572.96	6571.54	6526.03	6514.76

Table 1: Average Cost of a Design

Blind	CDL	CBL-50	CBL-100	CBL-150	CBL-200
15.54	12.98	13.26	13.36	13.03	12.94

Table 2: Average number of cycles per design

accurate view of the non-local requirements after all the exchange is done, the fact that the past cases are only an approximation of the present requirements in CBL seems to be offset by the more informed search done in the initial stages.

Our results conclusively demonstrate that conflict-driven learning (CDL) and Case-based Learning (CBL) improve both solution quality and processing time compared to blind search. In addition, once the learning is completed, CBL requires no run-time communication. Note however that CDL is required during the learning phase.

Related Work

We classify the work relevant to the topic on hand into three categories: Distributed Search, Conflict Management and Multi-agent Learning.

Distributed Search has been the explicit focus of research amongst a small group of DAI researchers for the past few years. Yokoo et al. (Yokoo, Durfee, & Ishida 1992), Conry et al. (Conry *et al.* 1991), and Sycara et al. (Sycara *et al.* 1991) have investigated various issues in Distributed Search. However, implicit in all of these pieces of work are the assumptions that the agents have homogeneous local knowledge and representations and tightly integrated system-wide problem-solving strategies across all agents.

Conflict management approaches are very similar to the Conflict Driven Learning mechanism presented here. Klein (Klein 1991) develops a theory of computational model of resolution of conflicts among groups of expert agents. Associated with each possible conflict is an advice for resolving the conflict. An advice is chosen from the set of conflict resolution advice of the active conflict classes to deal with the encountered conflict. We believe that Klein's work provides a general foundation for handling conflicts in design application systems. However, it falls short of embedding such conflict resolution mechanisms into the larger problem solving context that can involve studying issues like solution evaluation, information exchange and learning. Khedro and Genesereth (Khedro & Genesereth 1993) present a strategy called Progressive Negotiation for resolving conflicts among multi-agent systems. Using this strategy, the agents can provably converge to a mutually acceptable solution if one exists. However, the guarantee of convergence relies crucially on explicit declarative representation and exchange of all constraining information. More commonly, STEAM-like systems are aggregations of complex agents whose expertise is represented by a combination of declarative and procedural knowledge that cannot be captured as a set of explicit constraints.

Previous work related to learning in multi-agent systems is limited. Tan (Tan 1993), and Sen and Sekaran (Sen & Sekaran 1994) represent work in multi-agent reinforcement learning systems. While these works highlight interesting aspects of multi-agent learning systems, they are primarily centered around toy problems on a grid world. **STEAM** is one of the few complex multi-agent systems demonstrating the viability of such methods for interesting learning tasks in the domain of problem solving control, which is a notion that is not explicit in the above systems. Nagendra Prasad et al. (NagendraPrasad, Lesser, & Lander 1995) discuss organization role learning in **STEAM** for organizing the control of distributed search process among the agents. This work uses reinforcement learning to let the agents organize themselves to play appropriate roles in distributed search. Shoham and Tennenholtz (Shoham & Tennenholtz 1992) discuss co-learning and the emergence of conventions in multi-agent systems with simple interactions. Shaw and Whinston (Shaw & Whinston 1989) discuss a classifier system based multi-agent learning system for resource and task allocation in Flexible Manufacturing Systems. However, genetic algorithms and classifier systems have specific representational requirements to achieve learning. In many complex real-world expert systems, it may be difficult to achieve such requirements. A related work presented in Weiss (Weiss 1994) uses classifier systems for learning an aspect of multi-agent systems that is different from that presented here. Multiple agents use a variant of Holland's (Holland 1985) bucket brigade algorithm to learn appropriate instantiations of hierarchical organizations for efficiently solving blocks-world problems.

Conclusion

Our paper investigates the role of learning in improving the efficiency of cooperative, distributed search

among a set of heterogeneous agents for parametric design. Our experiments suggest that conflict-driven short-term learning can drastically improve the search results. However, even more interestingly, these experiments also show that the agents can rely on their past problem solving experience across many problem instances to be able to predict the kinds of conflicts that will be encountered and thus avoid the need for communicating feedback on conflicts as in the case of short-term learning (communication is still needed during the learning phase).

The methods presented here address the acute need for well-tailored learning mechanisms in open, reusable-agent systems like STEAM. A reusable agent system is an open system assembled by minimal customized integration of a dynamically selected subset from a catalogue of existing agents. Reusable agents may be involved in systems and situations that may not have been explicitly anticipated at the time of their design. Learning can alleviate the huge knowledge-engineering effort involved in understanding the agent mechanisms and making them work together.

References

Conry, S. E.; Kuwabara, K.; Lesser, V. R.; and Meyer, R. A. 1991. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6).

Holland, J. H. 1985. Properties of bucket brigade algorithm. In *First International Conference on Genetic Algorithms and their Applications*, 1–7.

Khedro, T., and Genesereth, M. 1993. Progressive negotiation: A strategy for resolving conflicts in cooperative distributed multi-disciplinary design. In *Proceedings of the Conflict Resolution Workshop, IJCAI-93.*

Klein, M. 1991. Supporting conflict resolution in cooperative design systems. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1379–1390.

Lander, S. E., and Lesser, V. R. 1996. Sharing meta-information to guide cooperative search among heterogeneous reusable agents. *To appear in IEEE Transactions on Knowledge and Data Engineering.*

Lander, S. E. 1994. *Distributed Search in Heterogeneous and Reusable Multi-Agent Systems.* Ph.D. Dissertation, University of Massachusetts.

Lesser, V. R. 1990. An overview of DAI: Distributed AI as distributed search. *Journal of the Japanese Society for Artificial Intelligence* 5(4):392–400.

Lesser, V. R. 1991. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1347–1362.

Nagendra Prasad, M. V.; Lesser, V. R.; and Lander, S. E. 1995. Learning organizational roles in a heterogeneous multi-agent system. Computer Science Technical Report 95–35, University of Massachusetts.

Sen, S., and Sekaran, M. 1994. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 426–431. Seattle, WA: AAAI.

Shaw, M. J., and Whinston, A. B. 1989. Learning and adaptation in DAI systems. In Gasser, L., and Huhns, M., eds., *Distributed Artificial Intelligence*, volume 2, 413–429. Pittman Publishing/Morgan Kauffmann Pub.

Shoham, Y., and Tennenholtz, M. 1992. Emergent conventions in multi-agent systems: Initial experimental results and observations. In *Proccedings of KR-92.*

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

Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, 330–337.

Weiss, G. 1994. Some studies in distributed machine learning and organizational design. Technical Report FKI-189-94, Institut für Informatik, TU München.

Yokoo, M.; Durfee, E. H.; and Ishida, T. 1992. Distributed cosntraint satisfaction for formalizing distributed problem solving. In *Proceedings of the Twelfth Conference on Distributed Computing Systems.*