

Constraints and Agents

Confronting Ignorance

Peggy S. Eaton, Eugene C. Freuder, and Richard J. Wallace

■ Research on constraints and agents is emerging at the intersection of the communities studying constraint computation and software agents. Constraint-based reasoning systems can be enhanced by using agents with multiple problem-solving approaches or diverse problem representations. The constraint computation paradigm can be used to model agent consultation, cooperation, and competition. An interesting theme in agent interaction, which is studied here in constraint-based terms, is confronting ignorance: the agent's own ignorance or its ignorance of other agents.

Constraints and agents have a natural synergy. On the one hand, agent behavior, for example, negotiation, can be modeled as constraint satisfaction and optimization. On the other hand, agents can be used to accomplish constraint satisfaction and optimization, for example, to solve distributed scheduling problems. Agents offer opportunities to apply the constraint computation paradigm and present challenges to extend the paradigm.

Constraint computation provides a general problem-solving framework (see the Constraints sidebar). Agents are self-directed problem-solving entities (see the Agents sidebar).

Software agents can benefit by using constraint computation to improve the efficiency of individual agent problem solving (Tambe 1996; Wellman 1994); assist in knowledge acquisition (Freuder and Wallace 1997); or model the difficult issues of negotiation, collaboration, and competition among agents with differing interests (Freuder and Eaton 1997; Liu and Sycara 1994a).

Constraint-based-reasoning systems can be enhanced by using software agents to improve performance by combining the expertise of multiple, heterogeneous problem solvers

(Petrie, Jeon, and Cutkosky 1997; Andreoli et al. 1997); improve solution quality when the different interests of multiple agents are necessary (Freuder and Eaton 1997); improve the performance of constraint-satisfaction methods by distributing the problem over multiple agents (Petrie, Jeon, and Cutkosky 1997; Eaton and Freuder 1996; Liu and Sycara 1994a); or improve system reliability by providing redundancy (Eaton and Freuder 1996; Cheng, Lee, and Wu 1996).

Background

We briefly survey some of the recent work on constraints and agents. We do not include distributed, concurrent, or parallel constraint-satisfaction systems. Parallel and concurrent constraint-satisfaction systems do not possess software agent characteristics as we define them. In particular, even though components in these systems might exchange data, they are not complete problem solvers but cogs within a problem-solving system. Constraint-agent technology extends the work on distributed constraint-satisfaction systems to provide autonomous entities capable of representing the diverse interests of heterogeneous systems during cooperative problem solving. We are also limiting the scope of this article by largely excluding robotic agents.

Nwana and Ndumu (1997) categorize software agents into six agent types: (1) collaborative constraint, (2) interface constraint, (3) information and internet constraint, (4) mobile constraint, (5) reactive constraint, and (6) hybrid constraint. We separate constraint agents into these categories and add a seventh category, infrastructure, to encompass the development of tools, languages, and frameworks to assist software engineers in using con-

Collaborative Constraint Agents	
Collaborative scheduling agents	Anke, Staudte, and Dilger 1997; Liu and Sycara 1994a; Miyashita 1997; Murthy et al. 1997; Musliner and Boddy 1997; Solotorvsky and Gudes 1997
Agent ordering during problem solving	Armstrong and Durfee 1997
Constraint-based discourse agents	Donaldson and Cohen 1997
Multiple diverse viewpoints	Eaton and Freuder 1996
Compromise strategies	Freuder and Eaton 1997
Improvement of asynchronous backtrack search	Havens 1997
Distributed valued constraint satisfaction	Lemaitre and Verfaillie 1997
Collaboration and resource requirements	Neiman et al. 1994
Distributed constraint optimization	Parunak et al. 1997
Minimization of intraagent constraint violations	Saks et al. 1997
Tracking of dynamic team activities	Tambe 1996
Modeling of interactions using markets	Wellman 1994
Interface Constraint Agents	
Constraint acquisition during matchmaking	Freuder and Wallace 1997
Mixed-initiative constraint-satisfaction problems	Jackson and Havens 1995
Information and Internet Constraint Agents	
Constraint-based knowledge broker agents	Andreoli et al. 1997, 1995
Java constraint library	Torrrens, Weigel, and Faltings 1997
Mobile Constraint Agents	
Mobile knowledge brokers	Andreoli et al. 1997
Solving of constraint-satisfaction problems over internet	Torrrens, Weigel, and Faltings 1997
Reactive Constraint Agents	
Cooperative constraint-satisfaction problem agents	Clearwater, Huberman, and Hogg 1991;
Specification of reactive agents	Liu and Sycara 1994b, 1993 Mali 1997
Hybrid and Heterogeneous Constraint Agents	
Reactive and deliberative collaboration	Anderson 1997
Reactive and collaborative agents	Carlson, Gupta, and Hogg 1997
Framework for constraint agents	Nareyak 1997
Distributed, concurrent engineering design	Orbst 1997; Petrie, Jeon, and Cutokosky 1997
Interface and collaborative agents	Tate 1997
Infrastructure	
Evaluation of agent problem solving	Mammen and Lesser 1997
Framework for reactive agents	Shvetsov, Nesterenko, and Starovit 1997
Information integration using context	Bressan and Goh 1997
Constraint extensions to VMRL	Diehl 1997
Web search techniques	Gilbert, Eidhammer, and Jonassen 1997
Constraint logic programming language	Lawal, Gilbert, and Letichevsky 1997
Java constraint library	Torrrens, Weigel, and Faltings 1997

Table 1. Summary of Constraint Agent Applications.

straint agents. Table 1 shows constraint agents mapped into these categories. (For more melding of constraints and agents, see Freuder [1997] and Fruhwirth et al. [1997].)

Collaborative Constraint Agents

Collaborative constraint agents are exemplified by the distributed meeting scheduling work of Liu and Sycara (1994a), the work on agent compromise strategies of Freuder and Eaton (1997), Donaldson and Cohen's (1997) work on turn taking during discourse, and Neiman et al.'s (1994) work on the exchange of abstracted messages among agents.

In the distributed meeting scheduling work by Liu and Sycara (1994a), *calendar agents* know their owner's schedule and scheduling preferences. Calendar agents negotiate with each other to schedule meetings. This system uses a flexible but centralized approach by appointing a task agent to coordinate a particular meeting. The *task agent* is the agent that has the most constrained schedule in the group. Agents exchange scheduling constraints and preferences as they search for a joint solution. A global solution is reached by the limited exchange of local information. If a conflict occurs, the agents can relax their constraints based on their own preferences and schedule. The goal of Liu and Sycara's work is similar to the goal of our work (Freuder and Eaton 1997), where agents find a solution that is not only feasible but is a high-quality solution for the group.

Constraint computation is useful for modeling the dynamic nature of discourse as explored by Donaldson and Cohen (1997). Turn taking among agents participating in discourse results in multiple goals being tracked and resolved; local repair techniques are used to generate solutions to the dynamic constraint-satisfaction problem (CSP).

Agents in the distributed airport resource management system negotiate by exchanging messages concerning their resource requirements (Neiman et al. 1994). The agents make local decisions about whether to contact other agents for a resource. Conflicts are managed by local relaxation strategies.

Interface Constraint Agents

The efficiency of asynchronous backtrack search is improved by committing to value selections made by human users in work by Jackson and Havens (1995). This algorithm reuses the valid part of previous solutions in an attempt to maintain a user's preferences for particular values. When the assignment cannot be made, the user is given an explanation.

A *matchmaker agent* provides potential solu-

tions (the suggestions) to a user based on partial knowledge (Freuder and Wallace 1997). The matchmaker gathers information about the user based on the user's evaluation of the suggestions provided by the matchmaker. Problem solving proceeds by iterating between the matchmaker and the user until the user is satisfied.

Information and Internet Constraint Agents

Collaborating *constraint-based knowledge broker agents* manage electronic documents for users (Andreoli et al. 1997). Brokers are agents that can process search requests for electronic information or delegate the search activity to other brokers. The agents use signed feature constraints to represent partially specified information. Information communicated among agents includes values, variables, and constraints. The agents use a common language. The life span of the agents varies depending on the application. Agents can be persistent and reuse previously retrieved information. Agents can also be terminated after queries have been satisfied.

Mobile Constraint Agents

Andreoli et al. (1997) propose the use of tools, such as ODYSSEY by General Magic, to support the implementation of mobile broker agents. ODYSSEY supports the movement agents across the network. For example, Andreoli et al.'s broker agent travels to a site, but rather than return with information that has been retrieved, the agent remains at the site processing queries from the user based on current local information. An advantage is lower communication costs because the local broker agent can perform local processing and summarize results before transmission.

Preliminary work on the JAVA constraint library provides a simple mobility mechanism for running constraint agents in a client-server mode of operation on the internet (Torrens, Weigel, and Faltings 1997).

Reactive Constraint Agents

The concurrent, *reactive CSP agents* in Liu and Sycara (1994a) and Clearwater, Huberman, and Hogg (1991) are simple agents performing local computations based on the state of variables with which they are concerned. The agents communicate with each other by changing the values of the variables to maintain consistency. Agent coordination occurs because shared variables are modified. A global solution emerges as a result of this simple agent coordination.

An important application field of agent technology is the facilitation of commerce on the World Wide Web.

Hybrid Constraint Agents and Heterogeneous Systems

Hybrid software agents combine one or more of the previously defined types. *Heterogeneous agent systems* are a collection of the various agent types that interoperate with each other using a common communication language.

PROCESSLINK is a system consisting of constraint specialists cooperating to solve concurrent engineering design problems (Petrie, Jeon, and Cutkosky 1997). The goal is to provide application programming interfaces (APIs) for legacy tools (computer-aided design systems, simulators, and so on) that can communicate with each other. Agents communicate with each other using KQML (the knowledge query and manipulation language). The tools are integrated using a bookkeeper, the REDUX agent, that keeps track of the activities associated with the multiple problem solvers and performs dependency-directed backtracking. The constraint manager agent performs consistency management and works together with the redux agent to explain inconsistencies when backtracking is necessary.

Infrastructure

A variety of tools, languages, and frameworks are being developed to support agents and constraints. Mammen and Lesser (1997) propose a general framework for the evaluation of MASs. The proposed tool allows the adjustment of problem parameters and gathers data as specified by the user. MINT, a web-based tool discussed in Bressan and Goh (1997), integrates information from different and possibly conflicting information sources using context information. Diehl (1997) extends VRML (internet standard for three-dimensional [3D] content) to include constraints used to describe relationships among objects in a 3D scene. The JAVA constraint library is a collection of JAVA classes to facilitate client-side constraint problem solving (Torrens, Weigel, and Faltings 1997).

Case Studies from the University of New Hampshire Constraint Computation Center

Software agents need to acquire knowledge. They need to exchange knowledge. They need to use this knowledge cooperatively and competitively. They might need to operate in an environment in which complete a priori disclosure of all relevant knowledge is impractical or undesirable.

In the remainder of this article, we illustrate three important agent interaction modes with examples from our own work: (1) consulta-

tion, (2) cooperation, and (3) competition. For consultation, we explore suggestion strategies; for cooperation, sharing strategies; and for competition, compromise strategies.

We briefly survey work we have begun that addresses the following intriguing questions: How can agents elicit a human's unarticulated desires in an effective and unobtrusive manner? How can agents cooperate to overcome their individual limitations when they do not want to allow unrestricted access to each other? How can agents compromise to work together despite differing desires, which they might not want to reveal? We believe that constraint technology supports promising preliminary answers to each of these questions. An interesting theme has emerged in our work, which we might call *confronting ignorance*: The consulting agent in our example must overcome its own ignorance of the customer to overcome the customer's ignorance of the consultant's service. The cooperating agents in our example must compensate for their individual ignorance of portions of their joint problem to arrive at a complete solution. The competing agents in our example must deal with their ignorance of each other's priorities.

Constraint information can be communicated at several levels: Our consulting agents exchange constraints and solutions, our cooperating agents exchange inferences, and our competing agents exchange choices or rankings. Knowledge, once acquired, can be propagated naturally.

An intriguing aspect of ignorance here is that agents might regard ignorance as desirable but still need to cope with the limitations it imposes. Agents might want to reveal as little as possible of their needs, their knowledge, or their priorities for privacy or proprietary reasons. At the same time, greater openness might enable their needs to be better fulfilled, their problems to be solved more easily, and their priorities to be better met. A key research issue then will be to develop different strategies that provide the best balance of privacy and openness for different needs.

Consultation

An important application field of agent technology is the facilitation of commerce on the World Wide Web. Search agents are already being used to locate web sites that offer specific services. In addition, content-focused matchmaker agents can provide advice to internet consumers (people or other agents) about complex products (Gomez et al. 1996). At present, the reigning paradigm for such agents is

the deep interview, illustrated by the PERSON-ALOGIC web site (Krantz 1997), where the primary mode of interaction is the MATCHMAKER query. MATCHMAKER poses a series of (essentially) multiple-choice questions about product features, and the customer's answers allow MATCHMAKER to select products that meet his/her specifications. We approach the problem of product selection using a constraint-based paradigm with a very different form of interaction.

In our paradigm, the primary mode of interaction is the *suggestion*, made by MATCHMAKER to the customer, where each suggestion is a possible product. The secondary mode of communication is the *correction*, made by the customer to the MATCHMAKER, indicating how the product selected by MATCHMAKER fails to meet the customer's needs.

The objective here is to model a situation in which customers do not enter the interaction with a fully explicit description of their needs. They might be unfamiliar with what is available in the marketplace. They cannot list all their requirements up front, but they can recognize what they do not want when they see it. We believe this arrangement to be a common form of customer conduct. (Picture yourself browsing through a store or a catalogue or interacting with a salesclerk.)

In our approach, MATCHMAKER's knowledge base and the customer's needs are both modeled as networks of constraints. A suggestion corresponds to a solution to the current CSP as it is understood by MATCHMAKER. A correction specifies the customer constraints that the proposed solution violates. Repeating the cycle of suggestion and correction allows MATCHMAKER to improve its picture of the customer's problem until a suggestion provides a satisfactory solution for the customer. The problem of both acquiring and solving a CSP has been termed the *constraint acquisition and satisfaction problem* (CASP) by Freuder (1995), where the basic suggestion-correction model was suggested but not implemented.

Matchmaking and Suggestion Strategies

MATCHMAKER can facilitate this interaction by an appropriate choice of suggestions, or tentative solutions. For example, some suggestion strategies can lead to a satisfactory customer solution more easily than others, that is, with fewer iterations of the suggestion-correction cycle. However, ease of use is not the only evaluation criterion. In an environment in which MATCHMAKER has an ongoing relationship with the customer, it might be desirable for MATCH-

MAKER to learn as much as possible about the customer's constraints to facilitate future interactions. In our implementation, it is possible—in fact, it proves experimentally the norm—for MATCHMAKER to come up with a satisfactory solution before acquiring all the customer constraints because constraints can be satisfied fortuitously by a suggestion. Thus, we use the number of customer constraints acquired by MATCHMAKER as another performance metric when comparing suggestion strategies. (Conversely, the customer might not want to reveal more information than necessary to complete the transaction. In this case, the second criterion is useful in gauging the degree to which the number of customer constraints acquired is minimized.)

If our goal is to limit the length of the dialogue between two agents, one strategy is to try to find solutions that are more likely to satisfy constraints between variables, even though these constraints are not currently in MATCHMAKER's CSP representation. This is consistent with the additional policy of maximizing satisfaction, in terms of the number of satisfied constraints, at each step in the dialogue. An alternative, and possibly perverse, approach is to maximize constraint violations. Here, the strategy is to find solutions that violate as many constraints as possible so that more constraints are incorporated into MATCHMAKER's set from the start.

For algorithms that use complete or exhaustive search, selecting values less likely to be in conflict with values in other variables is a promising method for maximizing satisfaction. For hill-climbing or heuristic repair methods, a method in the same spirit is *solution reuse*, that is, starting each search with the solution obtained earlier after revising information about conflicts based on the last customer communication. In each case, a method that conforms to the strategy of maximizing violations is the converse of the method for maximizing satisfaction. For complete search, we can choose values that are most likely to be in conflict. A corresponding hill-climbing technique is to search each time from a new location, that is, with a new set of initial values.

As suggested earlier, a second goal in matchmaking might be to learn as much (or as little) as possible about the customer during an interaction. Here, learning about the customer means learning the customer's constraints. Intuitively, strategies that maximize satisfaction should minimize the number of constraints learned, and violation strategies should maximize this quantity.

Constr. Probs.	Val. Ord.	Iterats. to Sol.	Violats. / Iter.	Undisc. Constr.	Sol. Sim.	Time (sec.)
2, .4	lex	8	8	59	.72	.03
	max	6	5	85	.83	.02
	min	8	11	37	.61	.02
	shuf	14	8	15	.22	.02
.2, .8	lex	12	14	69	.54	.55
	max	15	9	105	.65	.43
	min	11	17	47	.48	.53
	shuf	19	11	21	.24	.41

Notes. Iterats = number of iterations before a customer solution was found. Violats = mean number of customer violations on one iteration. Undisc constr = mean number of undiscovered customer constraints at end of a run. Sol sim = mean similarity (proportion of common values) of successive solutions found during a run.

Table 2. Matchmaking Dialogue Statistics for Different Value Orderings.

Comparing Suggestion Strategies

We illustrate these remarks with some simulations of customer-MATCHMAKER dialogues using random CSPs, from which MATCHMAKER and customer problems were derived by randomly selecting constraints from the original problem. For brevity, we limit our account to complete search algorithms although comparable results have been found with hill-climbing methods.

In these tests, we simplified the matchmaking dialogue in the following ways: We assumed that both MATCHMAKER's and the customer's constraints were drawn from the same universe. Hence, when the MATCHMAKER was apprised of a constraint violation, it did not have to decide what the constraint actually was, that is, the set of acceptable tuples. Moreover, constraints known to MATCHMAKER are assumed to be a proper subset of the customer's (implicit) constraints. (In an overall customer-MATCHMAKER dialogue, such constraints might be determined by preliminary questioning.) We also assumed that on each iteration of the communication cycle, the customer gave MATCHMAKER the complete set of constraints violated by the last solution.

In these simulations, each problem was tested five times, making five dialogues. At the beginning of each dialogue, the full constraint set was scanned, and with probability pb , a given constraint was added to both MATCHMAKER's and the customer's constraint sets. If the constraint was not chosen, then with proba-

bility pc , it was added to the customer's set. Four sets of values were used for pb and pc , respectively: 0.2 and 0.4, 0.2 and 0.8, 0.4 and 0.4, 0.4 and 0.8.

Different suggestion strategies were realized by ordering domain values in specific ways prior to search. To maximize constraint satisfaction, values in each domain were ordered by maximum average promise (max-promise), where *promise* is the relative number of supporting values in each adjacent domain (Geelen 1992), and these proportions are averaged across all adjacent domains. A violation strategy was obtained simply by reversing this order for each domain, which gave an ordering by minimum average promise (min-promise). Another violation strategy was to shuffle the domains before each search, that is, to order each domain by random sequential selection of its values. Lexical ordering of values served as a control.

Representative results for the different value orderings are shown in table 2. In one case ($pb = 0.2$, $pc = 0.4$), the satisfaction strategy, max-promise, found acceptable solutions after fewer iterations of the dialogue in comparison with either the lexical ordering or the constraint-violation strategies. However, with more customer constraints in relation to MATCHMAKER constraints, a violation strategy, min-promise, was more efficient in this respect than max-promise. In both cases, min-promise was the most effective in uncovering violations quickly, as reflected in the measure of

violations for each iteration. However, the shuffling procedure found more violations across the whole dialogue.

The trade-off expected with max-promise was clear-cut: This procedure uncovered far fewer customer constraints than any other. This trade-off was also found for the shuffling procedure. Although it was the least efficient ordering, it uncovered the most constraints.

Interestingly, the min-promise ordering required relatively few iterations but found more violations for each iteration on average than the other orderings, and in this respect, it tended to overcome the trade-off between efficiency and constraint discovery. In fact, when there were more customer constraints not in the initial MATCHMAKER set, this ordering was better than the satisfaction ordering, max-promise, on both metrics.

Further insight into the performance of different methods can be obtained by considering curves for measures, such as the number of undiscovered constraints, across an entire dialogue. An example is given in figure 1 for one problem where $pb = 0.2$ and $pc = 0.8$. Here, it can be seen that during the early iterations, min-promise finds the most constraints, but it levels out more quickly than shuffle. It also finds a completely satisfactory solution more quickly; so, its curve is shorter. Consequently, the curve for shuffle falls below the other curve on the eighth iteration. The curve for max-promise remains well above the other two throughout the dialogue.

From these results, it appears that either max- or min-promise can be used to limit dialogue length, depending on the characteristics of the problem. If we want MATCHMAKER to discover as much as it can about the customer while we bound the dialogue, min-promise should be used, but if the object is to find solutions with a minimum amount of prying, then max-promise is the best method.

Cooperation

Cooperative problem-solving agents can solve problems on their own and cooperate with group members when another agent possesses special expertise that can benefit the group, when an agent has redundant capabilities that increase the reliability of the group, or when the task is simply too large for one agent. A team of cooperating constraint-based reasoning agents might be able to solve a CSP even when members of the team have underconstrained representations by sharing information obtained during problem solving (Eaton and Freuder 1996). Each agent in a team of

Constraints

Many problems can be viewed naturally as constraint-satisfaction (or optimization) problems (CSPs) (Freuder and Mackworth 1994; Tsang 1993). In such problems, we seek to find values for problem variables that satisfy or optimize restrictions on value combinations. A computer configuration problem, for example, might require us to choose a power supply and a fan such that the fan is powerful enough to dissipate the heat generated by the power supply. Applications are found in many fields of AI, including planning, design, diagnosis, temporal reasoning, vision, and language.

More formally, CSPs are characterized by (1) a set of problem variables, (2) a domain of potential values for each variable, and (3) constraints that specify which combinations of values are permitted (consistent). These problems are often represented by constraint networks, whose nodes correspond to variables and whose links correspond to constraints.

Inference and search methods are applied during problem solving. A central form of inference, arc consistency, can be used to eliminate some inconsistent values (values that cannot be part of a solution) from the domains of the variables. Some search methods, generally based on backtracking, are systematic and complete, guaranteeing that a solution (or optimal solution) will be found. Others, generally forms of hill climbing, are stochastic and incomplete but can quickly find satisficing solutions.

Variations in the inference and search methods exploit features or characteristics of CSPs in an effort to achieve greater problem-solving efficiency. A common performance measure for constraint algorithms is the number of constraint checks performed, the number of times we have to ask if a combination of values is consistent.

The constraint-satisfaction paradigm has been extended in many directions. For example, CSP representations and algorithms have been extended to handle overconstrained problems (for example, Freuder and Wallace [1992]), dynamic problems (for example, Schiex and Verfaillie [1993] and Mittal and Falkenhainer [1990]), and distributed problems (for example, Yokoo, Ishida, and Kuwabara [1992]).

Constraint computation more generally draws on results from other disciplines, such as operations research and discrete mathematics. A variety of constraint solving methods have been embodied in powerful constraint languages and systems.

cooperating constraint agents is a constraint-based reasoner with a constraint engine, a representation of the CSP, and a coordination mechanism.

Compensating for Agent Ignorance

Cooperation among constraint agents can compensate for incomplete knowledge and permit a solution because the agents have different representations of a CSP. This agent-oriented technique uses the exchange of partial information rather than the exchange or comparison of an entire CSP representation when constraints are missing. Agent designers might want to avoid the complete exchange of representations

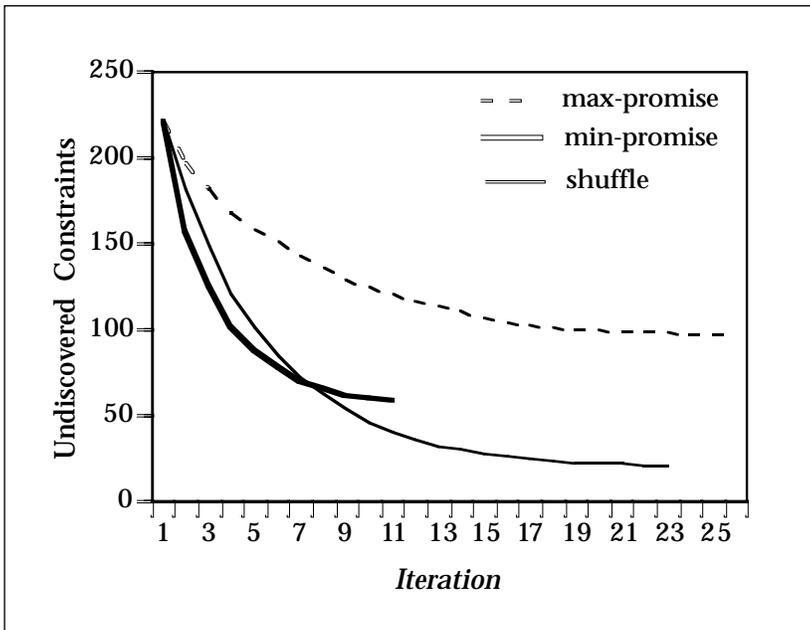


Figure 1. Undiscovered Customer Constraints after Successive Iterations with Three Value Orderings Representing Different MATCHMAKER Strategies. Means are based on five dialogues with one problem.

because agents developed by different companies might contain proprietary knowledge. A group of cooperating constraint-based reasoning agents might be able to solve a CSP even when members of the team have underconstrained representations of the problem.

Agents can exchange messages about constraint violations or required assignments of values to variables. The members of the group receiving a message must be able to translate the message given the context of their own representation. Information sharing among the members of the team is modeled as the translation of no-goods.

Sharing Inferences

The utility of sharing information among agents with different viewpoints is demonstrated in an experiment where three constraint agents cooperate to solve a logic puzzle problem. Consider the following logic puzzle, called the Flumm Four, taken from a commercial booklet of logic puzzles (Schuster 1995, p. 8):

The Flumm Four is a new punk rock group from Liverpuddle. The four musicians (Furple, Icky, Muke, and Slyme) play bass, drums, guitar, and keyboard, not necessarily in that order. They've taken the music world by storm with four hit songs from their first album. Each of these songs—Grossery Store, Heartburn Hotel, Love Is Garbage, and 2 Yuk 4 Words—was

written by a different member of the band. Can you match each musician with his instrument and his song?

1- Icky and Muke are, in one order or the other, the composer of 2 Yuk 4 Words and the keyboardist.

2- Neither Slyme nor the composer of Heartburn Hotel plays guitar or bass.

3- Neither the bass player (who isn't Icky) nor the drummer wrote Love Is Garbage.

Logic puzzle problems can be represented as a CSP using a constraint network; the variables are the nodes of the network, and the edges of the network are the constraints among the variables. All constraints in this network are inequality constraints. A solution to the logic puzzle is a labeling of the network so that the constraints are not violated. Figure 2 shows the three different representations used by the agents in this experiment.

The constraint agents are autonomous and are able to communicate with the other agents working on the logic puzzle. Each agent contains a different representation of the problem, a problem-solving engine, and a facility for communication with other agents working on the problem.

In this example, no agent can solve the problem alone because of missing information in each representation. CSP representations can be missing constraints for a variety of reasons, including constraints that are purposefully left out of the representation because they are difficult to describe or because an engineer makes a mistake when building the representation. The representations used by agent 2 and agent 3 are both underconstrained because rule 1 cannot easily be represented using a standard binary constraint between the two variables. Without this additional information, agent 2 and agent 3 cannot individually solve the problem. The representation used by agent 1 is underconstrained because an arbitrary constraint was left out when the engineer created the representation. Note that the constraint between Heartburn and Guitar was left out of agent 1's representation. Although the agents cannot solve the problem as individuals, they can cooperate and find a solution to the logic puzzle.

Each agent runs a simple consistency and search algorithm. Agents cooperate by sending messages containing the inconsistent (no-good) values discovered during the preprocessing phase of solving the CSP. *Arc consistency* is a preprocessing constraint inference method used to reduce the search effort. The domains of the variables can be reduced during the pre-

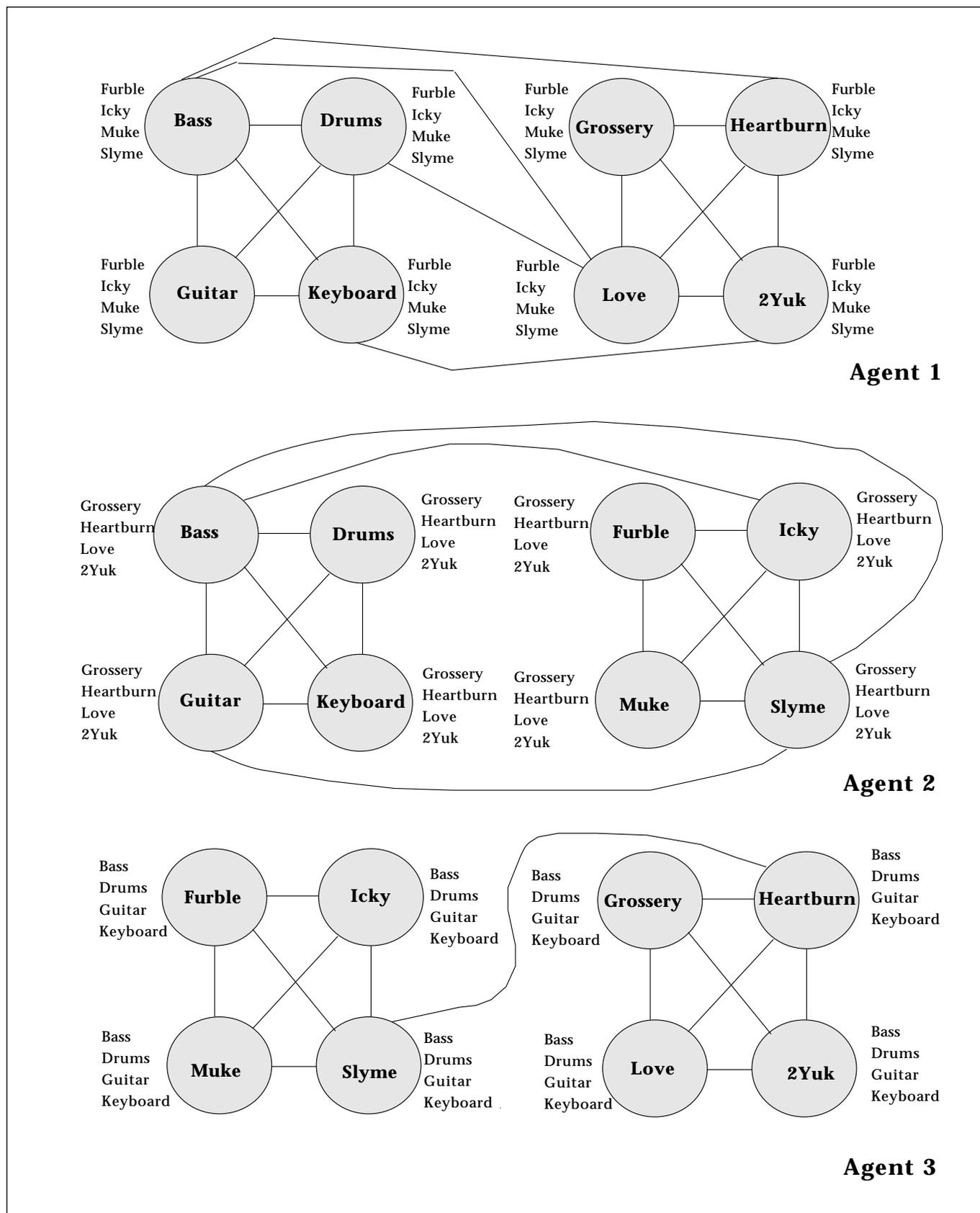


Figure 2. Multiple Representations of the Flumm Four Logic Puzzle.

Agents

Software agent technology is a broad area of research that has produced a variety of definitions for *agent*. Comprehensive discussions of software agents can be found in Wooldridge and Jennings (1994), Nwana (1996), and Nwana and Ndumu (1997). Nwana defines *software agents* as those agents that embody at least two of the following three characteristics: (1) autonomous functions, (2) collaborative problem solving, and (3) learning ability.

Autonomous agents are able to solve problems on their own, can be decentralized, and can be distributed geographically.

The elements of communication and cooperation are fundamental to the definition of agents because it is the group problem-solving abilities of a collection of agents that make them unique among other types of software. Collaborative problem-solving agents are able to communicate with others to coordinate group activity.

Although learning ability is important to the development of intelligent agents capable of adapting to dynamic environments, few agents have this characteristic.

Evaluating the success of software agents can be difficult because the performance measures are rather ad hoc and particular to the specific implementation. There are some common metrics used to evaluate the effectiveness of a particular agent or team of agents, among them the number of messages exchanged during problem solving, individual and group solution quality, central processing unit time, comparisons with centralized approaches, degree of autonomy, degree of collaboration, and learning ability.

processing phase if the arc-consistency algorithm identifies values that are inconsistent. The values identified as no good are sent to the other agents when the arc-consistency algorithm has finished. In the Flumm Four example, agent 3 discovers several no-goods during preprocessing, including the knowledge that the author of Heartburn does not play the Guitar. Agent 3 sends the no-good list to agent 1 and agent 2. Similarly, the other agents send their no-good list to the team. During problem solving, an agent incorporates messages received from others into their own representations, and the arc-consistency algorithm is run again on the modified representation. This process continues until there are no changes in the domains; then the agent searches for a solution.

Competition

We describe several joint problem-solving strategies for a team of multiinterest constraint-based reasoning agents (Freuder and Eaton 1997). Agents yield in one area of conflict to gain concessions in another area. For example, in a distributed meeting scheduling

problem, individuals with different preferences might be willing to give up the choice of location if allowed to choose the time of the meeting. Compromise benefits the overall solution quality by allowing each agent an opportunity to participate in the solution. We demonstrate the utility of this approach using a collection of agents collaborating on random graph coloring problems. We propose several simple metrics for evaluating solutions from the perspective of individual agents and additional metrics for evaluating the solutions as compromises. Finally, we experimentally evaluate the performance of the strategies with respect to the metrics.

Compromise Strategies and Metrics

Compromise strategies are useful when agents have competing goals but perceive an advantage to working together in a fair way and can be useful in application areas where user preferences play an important role, such as scheduling, intelligent user interfaces, and telecommunication (Maes 1994). The joint problem-solving strategies we propose emphasize the emergence of a solution among a team of agents:

The *turn-taking strategy* is a simple strategy where agents take turns assigning values to variables.

The *average-preference strategy* is an a priori computation of the average-preference utility for the values of each variable across the agents. The value selected for assignment has the highest-average preference utility. The individual metric computation is based on the original set of preference utilities for the agent.

The *concession strategy* is a variation of turn taking, where agents making a variable assignment will concede their turn to another team member if the other agent has a higher preference utility for a variable-value assignment.

The lowest-score strategy was designed to minimize disparity among the agents. The team members keep track of their scores for the current labeling of a problem. When a variable is to be assigned a value, the agent with the lowest score chooses based on its own preferences.

The selection of an appropriate compromise strategy is dependent on the type of group interaction desired and the type of measurements used. We propose simple metrics to compare how well individual agents fare and how well the group performs when using each compromise strategy. We also compare the performance of the strategies in terms of efficiency.

Individual agents compute the quality of a solution by summing the preference utilities associated with the values assigned to the vari-

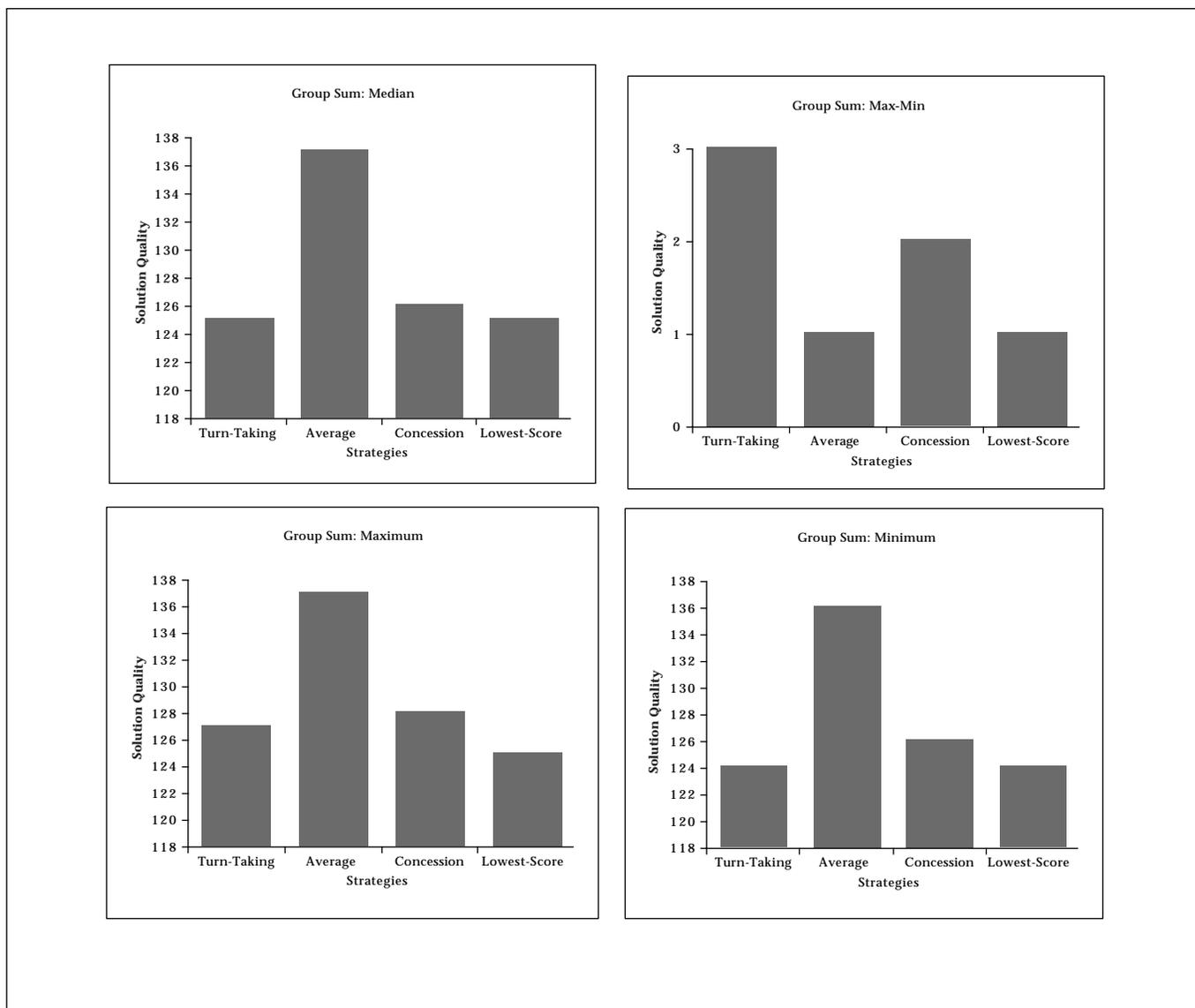


Figure 3. Group Metrics.

ables. The individual sum measure can be used as an indicator of an agent's preference for a particular problem solution and is the sum of the preference utilities for each variable-value assignment in the solution. These metrics are similar to the metrics proposed by Rosen-schein and Zlotkin (1994) for evaluating two-agent negotiation protocols.

The team computes the quality of a solution by summing each agent's solution quality. The strategies are then compared using the following group metrics across the set of experiments: (1) the median of individual metric for each agent, (2) the maximum solution quality of the group, (3) the minimum solution quali-

ty of the group, (4) the disparity (the difference between maximum and minimum solution qualities), and (5) the number of constraint checks. In addition, we consider the efficiency of the compromise strategies by comparing the number of constraint checks generated during problem solving.

Comparing Compromise Strategies

The compromise strategies and metrics are evaluated using solvable random coloring problems. *Coloring problems* are representative of scheduling and resource-allocation problems. In these problems, colors must be assigned to variables so that related variables

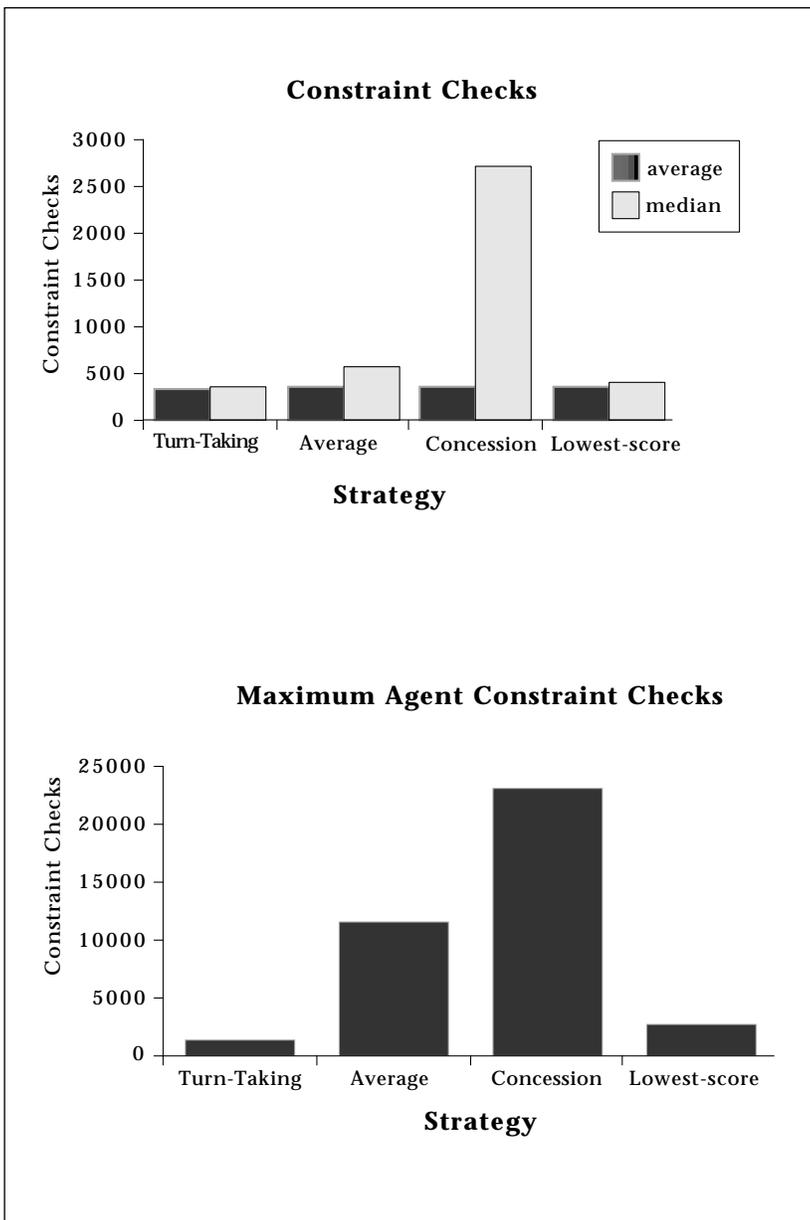


Figure 4. Group Constraint Check Metric.

do not have the same color. Agents can have different preferences for colors assigned to variables. The experiments consist of a set of 100 random coloring problems.

Agents use identical search algorithms and CSP representations except for their preference vectors. The preference vectors are randomly assigned when the problem representation is created. The problem representation of each agent is augmented with a preference utility for each value associated with a variable. The preference utilities are assigned on a scale of 1 to the maximum domain size. For example, if the domain size is 6, the maximum preference

value 6 can only be assigned to one of the domain values of this variable.

Figures 3 and 4 show solution qualities for the group. To select a compromise strategy for a team, the designer must consider (1) lowest solution quality generated by an agent on the team, (2) median solution quality—the team score, (3) highest solution quality generated by an agent on the team, (4) disparity of scores among team members, (5) problem-solving performance—constraint checks, and (6) the willingness of the agents to share all the information they possess.

Trade-offs are inevitable; no one strategy wins in every performance area. A team requiring a low disparity among team members and information hiding will select the concession strategy, but a team requiring high problem-solving performance would choose turn taking.

The average preference strategy provides high solution quality and low disparity among agents; this strategy performs well, but there are two issues that must be considered: First, the average preference strategy requires agents to exchange all preference utilities before beginning problem solving. Second, the average preference strategy is not as efficient as other strategies.

The max group metric identifies which strategy produced the highest-average solution quality. The average preference metric performs best.

Disparity (max-min) is the difference in solution qualities over the set of agents, so a low score is best. This measure is useful when we are interested in everyone on the team being equally happy. The average preference strategy and the lowest-score strategy generate solutions in a way that minimizes the disparity of the scores among the agents. Agents using the turn-taking strategy suffer from increased disparities in solution quality.

Conclusion

Constraint technology can help us build agents. Agents can augment constraint technology. Current research continues along these two paths.

Constraint-agent technology is being applied to agent interactions such as collaboration, negotiation, and coordination (Donaldson and Cohen 1997; Freuder and Eaton 1997; Mammen and Lesser 1997; Eaton and Freuder 1996; Liu and Sycara 1994a; Neiman et al. 1994).

Constraint-based reasoning is being explored as a way to gain performance improvements during agent problem solving (Havens

1997; Tambe 1996; Wellman 1994).

Scalability and stability issues where large numbers of agents interact to solve problems need to be addressed because the technology is applied to large, real-world problems (Carlson, Gupta, and Hogg 1997; Freuder and Eaton 1997).

The improvement of problem solving and the quality of solutions generated is the research focus of constraint agents because they are applied to resource-allocation problems, such as distributed scheduling (Anke, Staudte, and Dilger 1997; Miyashita 1997; Musliner and Boddy 1997; Solotorvsky and Gudes 1997; Liu and Sycara 1994a).

The use of constraint agents to tackle the explosion of information on the internet is an active area of research that includes the acquisition of user constraints, the development of mobile constraint agents, and the development of the infrastructure to support the transfer of agents over networks. Infrastructure needs include tools, protocols, and languages for interagent communication (Andreoli et al. 1997; Freuder and Wallace 1997; Shvetsov, Nesterenko, and Starovit 1997; Torrens, Weigel, and Faltings 1997).

Teams of constraint agents and constraint-based reasoning techniques are being explored as a framework for solving distributed, concurrent engineering problems (Obrst 1997; Parunak et al. 1997; Petrie, Jeon, and Cutkosky 1997).

In our Constraint Computation Center, we are exploring agent consultation, cooperation, and competition. Confronting ignorance has emerged as an intriguing theme.

Constraints can naturally arise in the agents community, for example, in modeling negotiation. Agents can naturally arise in the constraints community, for example, in cooperative problem solving. It is important for the two communities to communicate lest they reinvent each other's wheels.

Acknowledgments

This material is based in part on work supported by the National Science Foundation under grant IRI-9504316. We thank Nancy Schuster and Dell Magazines for permission to reproduce material from Dell Logic Puzzles.

References

- Anderson, J. 1997. WAFFLER: A Constraint-Directed Approach to Intelligent Agent Design. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 70–75. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Andreoli, J.; Borghoff, U.; Pareschi, R.; and Schlichter, J. 1995. Constraint Agents for the Information Age. *Journal of Universal Computer Science* 1(12): 762–789.
- Andreoli, J.; Borghoff, U.; Pareschi, R.; Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Constraints and Agents for a Decentralized Network Infrastructure. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 39–44. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Anke, K.; Staudte, R.; and Dilger, W. 1997. Producing and Improving Time Tables by Means of Constraint and Multiagent Systems. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 142–146. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Armstrong, A., and Durfee, E. 1997. Dynamic Prioritization of Complex Agents in Distributed Constraint-Satisfaction Problems. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 8–13. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Bressan, S., and Goh, C. 1997. Semantic Integration of Disparate Information Sources over the Internet Using Constraint Propagation. Paper presented at the Workshop on Constraint Reasoning on the Internet, 29 October–1 November, Schloss Hagenberg, Austria.
- Carlson, B.; Gupta, V.; and Hogg, T. 1997. Controlling Agents in Smart Matter with Global Constraints. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 58–63. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Cheng, B.; Lee, J.; and Wu, J. 1996. Speeding Up Constraint Propagation by Redundant Modeling. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (CP96)*, ed. E. C. Freuder, 91–103. Berlin: Springer-Verlag.
- Clearwater, S.; Huberman, B.; and Hogg, T. 1991. Cooperative Solution of Constraint-Satisfaction Problems. *Science* 254:1181–1183.
- Diehl, S. 1997. Extending VRML by One-Way Equational Constraints. Paper presented at the Workshop on Constraint Reasoning on the Internet, 29 October–1 November, Schloss Hagenberg, Austria.
- Donaldson, T., and Cohen, R. 1997. Constraint-Based Discourse Agents. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 87–92. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Eaton, P. S., and Freuder, E. C. 1996. Agent Cooperation Can Compensate for Agent Ignorance in Constraint Satisfaction. In *Agent Modeling Papers from the 1996 AAI Workshop*, eds. M. Tambe and P. Gmytrasiewicz, 24–29. Technical Report WS-96-02. Menlo Park, Calif.: AAI Press.
- Freuder, E. C. 1997. *Constraints and Agents: Papers from the 1997 AAI Workshop*. Technical Report WS-97-05. Menlo Park, Calif.: AAI Press.
- Freuder, E. C. 1995. Active Learning for Constraint Satisfaction. Paper presented at the AAI-95 Fall Symposium on Active Learning, 10–12 November, Cambridge, Massachusetts.
- Freuder, E., and Eaton, P. 1997. Compromise Strategies for Constraint Agents. In *Constraints and Agents:*

- Papers from the 1997 AAI Workshop*, 1–7. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Freuder, E. C., and Mackworth, A. K. 1994. *Constraint-Based Reasoning*. Cambridge, Mass.: MIT Press.
- Freuder, E. C., and Wallace, R. J. 1997. Suggestion Strategies for Constraint-Based MATCHMAKER Agents. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 105–111. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Freuder, E. C., and Wallace, R. J. 1992. Partial Constraint Satisfaction. *Artificial Intelligence* 58:21–70.
- Fruhwith, T.; Hermengildo, M.; Tarau, P.; Codognet, P.; and Rossi, F., eds. 1997. Paper presented at the Workshop on Constraint Reasoning on the Internet, 29 October–1 November, Schloss Hagenberg, Austria.
- Garrido, L., and Sycara, K. 1996. Multiagent Meeting Scheduling: Preliminary Experimental Results. Paper presented at the Second International Conference on Multiagent Systems (ICMAS-96), 10–13 December, Kyoto, Japan.
- Geelen, P. A. 1992. Dual Viewpoint Heuristics for Binary Constraint-Satisfaction Problems. In *Proceedings of the 1992 European Conference on Artificial Intelligence*, 31–35. Chichester: Wiley.
- Gilbert, D.; Eidhammer, I.; and Jonassen, I. 1997. STRUCTWEB: Biosequence Searching on the Web Using CLP(FD). Paper presented at the Workshop on Constraint Reasoning on the Internet, 29 October–1 November, Schloss Hagenberg, Austria.
- Gomez, J.; Weisman, D. E.; Trevino, V. B.; and Woolsey, C. A. 1996. Content-Focused MATCHMAKERS. In *Money and Technology Strategies* 2(3).
- Havens, W. 1997. No-Good Caching for Multiagent Backtrack Search. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 26–31. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Jackson, W., and Havens, W. 1995. Committing to User Choices in Mixed-Initiative CSPs. Paper presented at the Fifth Scandinavian Conference on Artificial Intelligence, 29–31 May, Trondheim, Norway.
- Krantz, M. 1997. The Web's Middleman. *Time* 149(7): 67–68.
- Lawal, B.; Gilbert, D.; and Letichevsky, A. 1997. A Web-Based Course Scheduler in Constraint Logic Programming: Interactive Computing with Constraints. Paper presented at the Workshop on Constraint Reasoning on the Internet, 29 October–1 November, Schloss Hagenberg, Austria.
- Lemaitre, M., and Verfaillie, G. 1997. An Incomplete Method for Solving Distributed Valued Constraint-Satisfaction Problems. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 14–20. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Liu, J., and Sycara, K. 1996. Multiagent Coordination in Tightly Coupled Real-Time Environments. Paper presented at the Second International Conference on Multiagent Systems (ICMAS-96), 10–13 December, Kyoto, Japan.
- Liu, J., and Sycara, K. 1995. Exploiting Problem Structure for Distributed Constraint Optimization. Paper presented at the First International Conference on Multiagent Systems (ICMAS), 12–14 June, San Francisco, California.
- Liu, J., and Sycara, K. 1994a. Distributed Meeting Scheduling. Paper presented at the Sixteenth Annual Conference of the Cognitive Science Society, 13–16 August, Atlanta, Georgia.
- Liu, J., and Sycara, K. 1994b. Distributed Problem Solving through Coordination in a Society of Agents. Paper presented at the Thirteenth International Workshop on Distributed Artificial Intelligence, 17–29 July, Lake Quinalt, Washington.
- Liu, J., and Sycara, K. 1993. Distributed Constraint Satisfaction through Constraint Partition and Coordinated Reaction. Paper presented at the Twelfth International Workshop on Distributed Artificial Intelligence, May, Hidden Valley, Pennsylvania.
- Maes, P. 1994. Agents That Reduce Work and Information Overload. *Communications of the ACM* 37(7): 30–40.
- Mali, A. 1997. Constraint-Based Specification of Reactive Multiagent Systems. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 51–57. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Mammen, D., and Lesser, V. 1997. A Test Bed for the Evaluation of Multiagent Communication and Problem-Solving Strategies. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 32–38. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Mittal, S., and Falkenhainer, B. 1990. Dynamic Constraint-Satisfaction Problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 25–32. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Miyashita, K. 1997. Iterative Constraint-Based Repair for Multiagent Scheduling. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 136–141. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Murthy, S.; Rachlin, J.; Akkiraju, R.; and Wu, F. 1997. Agent-Based Cooperative Scheduling. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 112–117. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Musliner, D., and Boddy, M. 1997. Contract-Based Distributed Scheduling for Distributed Processing. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 118–128. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Nareyek, A. 1997. Constraint-Based Agents. In *Constraints and Agents: Papers from the 1997 AAI Workshop*, 45–50. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.
- Neiman, D.; Hildum, D.; Lesser, V.; and Sandholm, T. 1994. Exploiting Metalevel Information in a Distributed Scheduling System. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Nwana, H. 1996. Software Agents: An Overview. *Knowledge Engineering Review* 11(3): 205–244.
- Nwana, H., and Ndumu, D. 1997. An Introduction

to Agent Technology, Volume 1198, 1–26. Berlin: Springer-Verlag.

Obrst, L. 1997. Constraints and Agents in MADE-SMART. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 83–86. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Parunak, V.; Ward, A.; Fleischer, M.; Sauter, J.; and Chang, T. 1997. Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 93–99. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Petrie, C.; Jeon, H.; and Cutkosky, M. 1997. Combining Constraint Propagation and Backtracking for Distributed Engineering. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 76–82. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Rosenschein, J., and Zlotkin, G. 1994. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. Cambridge, Massachusetts: MIT Press.

Saks, V.; Braidic, G.; Kott, A.; and Kirschner, C. 1997. Distributed Medical Evacuation Planning: What Problem Should Each Agent Solve? In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 129–135. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Schiex, T., and Verfaillie, G. 1993. No-Good Recording for Static and Dynamic Constraint-Satisfaction Problems. In Proceedings of the International Conference on Tools with AI, 48–55. Washington, D.C.: IEEE Computer Society.

Schuster, N. 1995. Flumm Four Logic Puzzle. In *Logic Puzzles*, 8. New York: Dell Magazines.

Shvetsov, I.; Nesterenko, T.; and Starovit, S. 1997. Technology of Active Objects. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 64–69. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Solotorvsky, G., and Gudes, E. 1997. Solving a Real-Life Nurses Time Tabling and Transportation Problem Using Distributed CSP Techniques. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 148–153. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Tambe, M. 1996. Tracking Dynamic Team Activity. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 80–87. Menlo Park, Calif.: American Association for Artificial Intelligence.

Tate, A. 1997. Multiagent Planning via Mutually Constraining the Space of Behavior. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 100–104. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Torrens, M.; Weigel, R.; and Faltings, B. 1997. JAVA Constraint Library: Bringing Constraints Technology on the Internet Using the JAVA Language. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 21–25. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. San Diego, Calif.: Academic.

Wellman, M. 1994. A Computational Market Model for Distributed Configuration Design. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 401–407. Menlo Park, Calif.: American Association for Artificial Intelligence.

Wooldridge, M., and Jennings, N. 1994. Agent Theories, Architectures, and Languages: A Survey. In *1994 European Conference on AI Workshop on Agent Theories, Architectures, and Languages*, eds. M. Wooldridge and N. Jennings, 1–39. Berlin: Springer-Verlag.

Yokoo, M.; Ishida, T.; and Kuwabara, K. 1992. Distributed Constraint Satisfaction for DAI Problems. Paper presented at the Tenth European Conference on Artificial Intelligence, 3–7 August, Vienna, Austria.



Peggy S. Eaton is a Ph.D. candidate in the Department of Computer Science at the University of New Hampshire. Her research interests include constraint-based reasoning agents and distributed constraint computation. Her e-mail address is pse@cs.unh.edu.



Eugene C. Freuder is a professor in the Department of Computer Science at the University of New Hampshire and director of its Constraint Computation Center. He is a fellow of the American Association for Artificial Intelligence, editor in chief of *Constraints*, and executive chair of the organizing committee of the International Conference on Principles and Practice of Constraint Programming. He chaired the Workshop on Constraints and Agents at AAAI-97. His e-mail address is ecf@cs.unh.edu.



Richard Wallace received his Ph.D. in psychology from the University of Oregon and an M.S. in computer science from the University of New Hampshire in 1987. Currently, he is working with Eugene Freuder in the area of constraint-based reasoning at the University of New Hampshire. The primary focus of his research in computer science has been on extensions of the constraint-satisfaction paradigm to overconstrained and dynamically changing problems. His e-mail address is rjw@cs.unh.edu.