Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents

Susan E. Lander and Victor R. Lesser
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{lander,lesser}@cs.umas8.edu

Abstract

In our research, we explore the role of negotiation for conflict resolution in distributed search among heterogeneous and reusable agents. We present *negotiated search,* an algorithm that explicitly recognizes and exploits conflict to direct search activity across a set of agents. In negotiated search, loosely coupled agents interleave the tasks of 1) local search for a solution to some subproblem; 2) integration of local subproblem solutions into a shared solution; 3) information exchange to define and refine the shared search space of the agents; and 4) assessment and reassessment of emerging solutions. Negotiated search is applicable to diverse application areas and problem-solving environments. It requires only basic search operators and allows maximum flexibility in the distribution of those operators. These qualities make the algorithm particularly appropriate for the integration of heterogeneous agents into application systems. The algorithm is implemented in a multi-agent framework, TEAM, that provides the infrastructure required for communication and cooperation.

1 Introduction

The current state of knowledge-based technology is such that almost every application system is built from scratch. In order to move beyond the prohibitive cost of constantly reinventing, rerepresenting, and reimplementing the wheel, researchers are beginning to examine the feasibility of building application systems with reusable agents [Neches *et* a/., 1991]. A reusable agent is designed to work without *a priori* knowledge of the agent set in which it will be embedded, instead using a flexible, reactive approach to cooperation. Although this flexibility can lead to inefficient problem solving, an agent can often gather information about the agent set as problem solving progresses to improve efficiency.

This research was supported by ARPA under ONR Contract #N00014-92-J-1698. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

Multi-agent systems do not traditionally acknowledge the role of conflict among agents as a driving force in the control of problem-solving activity. In reusable-agent systems, however, conflict is inevitable since agents are implemented at different times by different people and in different environments. We present a distributed-search algorithm, *negotiated search,* that uses conflict as a source of control information for directing search activity across a set of heterogeneous agents in their quest for a mutually acceptable solution.

The negotiated-search algorithm has been successfully incorporated into two implemented systems. In [Lander and Lesser, 1992b], we describe distributed search in the context of a seven-agent steam condenser design system and discuss how different operator/agent assignments within the negotiated-search algorithm affect problem solving. In [Lander and Lesser, 1992a], a two-agent contract negotiation system is presented, and negotiated search is compared to a search strategy that is tailored to characteristics of that environment. Through analysis of the environment and search algorithms, we show the versatility and effectiveness of negotiated search in reusable-agent systems while also pointing out that customized search strategies are inflexible but can improve system performance when they can be applied. In this paper, we describe negotiated search from an application-independent perspective.

The need for a flexible algorithm to support reusability and heterogeneity motivates particular aspects of negotiated search:

- Conflict, negotiation, and democratic determination of acceptability are integral parts of the algorithm.
- Agent coordination is accomplished through clearly defined individual roles in the evolution of a shared solution. These roles are realized as operators that accomplish state transitions on shared solutions.
- Operators represent standard and widely available search and information-assimilation capabilities. A particular agent may instantiate all defined operators or some subset of defined operators.
- Whenever possible, feedback is used to refine the perceived search spaces of individual agents to more closely reflect the true composite search space.

TEAM agents are not hostile and will not intentionally

mislead or otherwise try to sabotage another agent's reasoning. They are cooperative in the sense that an agent is willing to contribute both knowledge and solutions to other agents as appropriate and to accept solutions that are not locally optimal in order to find a mutually-acceptable solution. Each agent is a stand-alone system with specific capabilities that allow it to be included in an integrated multi-agent system. We assume that agents can be heterogeneous in architecture, inference engines, evaluation criteria and priorities for solutions, and in long-term knowledge. Each agent does its own internal scheduling and has private data, knowledge, and history mechanisms.

In negotiated search, agents interleave the tasks of 1) local search for a solution to some subproblem; 2) integration of local subproblem solutions into a shared solution (the *composite* solution);[1] 3) negotiation to define and refine the shared search space of the agents; and 4) assessment and reassessment of emerging solutions.

In the remainder of the paper, we first motivate the development of our negotiated-search model by presenting an intuitive description of negotiation and, from this foundation, constructing an algorithmic model of the negotiation process. The next section details negotiated search from a state-based perspective similar to that used by von Martial to describe negotiation protocols in distributed planning [von Martial, 1992]. We then present seven basic negotiated-search operators. The final section briefly describes the status of the implementation and extensions to this model that are not covered in this paper.

2 An Initial Perspective on Negotiation

In this section, we begin with an intuitive description of negotiation:

> One agent generates a proposal and other agents review it. If some other agent doesn't like the proposal, it rejects it and provides some feedback about what it doesn't like. Some agent may generate a counter-proposal. If so, the other agents (including the agent that generated the first proposal) then review the counter-proposal and the process repeats. As information is exchanged, conflicts become apparent among the agents. Agents may respond to the conflicts by incrementally relaxing individual preferences until some mutually acceptable ground is reached.

This example captures the primary characteristics that one would expect to see:

- proposals are generated by one or more agents
- agents evaluate proposals based on their individual criteria for solution acceptability
- agents provide feedback about what they like or don't like about particular proposals, resulting in a progressively better understanding of the shared requirements for solutions over time
- agents can play different roles in the negotiation process, e.g., an agent can be a reviewer for another agent's proposal and then be a generator for a counter-proposal
- conflicts exist among the agents' requirements for acceptable solutions
- agents incrementally relax their solution requirements to reach agreement
- the decision to accept or not accept a proposal is a joint, democratic process

Some extensions to the definition are required. For example, it assumes that a proposal becomes a solution when it is accepted by all agents. However, this assumption rules out situations in which high-level problems are decomposed and each agent works on some subproblem. In this case, the proposal an agent makes does not represent a complete solution but rather some component of a solution that interacts with other components through shared attributes. Evaluation is then indirect since an agent cannot evaluate proposals for interacting components that are outside of its domain of expertise. In negotiated search, an agent evaluates an external interacting-component proposal by creating and evaluating a compatible local proposal (i.e., one that has the same values for shared attributes), thereby focusing on how the external proposal affects local quality.

Although a proposal includes the information required to implement a solution, it provides only a surface-level view of the reasoning that went into creating it. It is sometimes possible to make guesses about other agents' requirements that could be used in generating counter-proposals. However, in the general case of reusable agents, external local evaluation criteria for solutions cannot be predicted, nor can they be inferred from the "snapshot" provided by a proposal. For proposals and counter-proposals to be related, there must be a deeper understanding of the shared search space of the agents. This understanding is achieved through a feedback system that can be separate from the proposals.

3 Negotiated Search

Artificial intelligence researchers have previously used the term negotiation with respect to conflict resolution and avoidance [Adler *et* a/., 1989; Klein, 1991; Lander and Lesser, 1992a; Sycara, 1985; Werkman, 1992], task allocation [Cammarata *et* a/., 1983; Durfee and Montgomery, 1990; Davis and Smith, 1983], and resource allocation [Adler *et* a/., 1989; Conry *et* a/., 1992; Sathi and Fox, 1989; Sycara *et* a/., 1991]. Negotiation is sometimes treated as an independent process that is used to select one of a set of existing alternative solutions [Zlotkin and Rosenschein, 1990] rather than as an inherent part of a solution-generation process. It can be difficult under conditions where agents are hostile and unwilling to share private information [Sycara, 1985]. Negotiation can occur among peers [Cammarata *et* a/., 1983; Lander and Lesser, 1992b], through a mediator or arbitrator [Sycara, 1985; Werkman, 1992], or hierarchically through an organization [Durfee and Montgomery, 1990; Davis and Smith, 1983]. It can occur at

[1] Sat hi similarly uses the term *compoaition* as the name of a specific search operator that combines local information [Sathi and Fox, 1989]

either the domain or control level of problem-solving. Laasri et. al. describe the *recursive negotiation model,* a general model of multi-agent problem solving that details various situations that can potentially benefit from negotiation [Laasri *et al,* 1992]. In examining this model, it becomes clear that negotiation is a pervasive process that remains relatively untapped by current computational systems. In developing the negotiated-search model, we have tried to capture the key requirements for negotiation without restricting the domain, task decomposition, or organizational model of the agent set.

Several researchers have developed algorithms and heuristics for constraint-directed distributed search in situations involving multiple homogeneous agents [Sathi and Fox, 1989; Sycara *et al,* 1991; Yokoo *et* a/., 1992],[2] We extend this work to handle situations where heterogeneous agents may have different or multiple local problem-solving paradigms, instantiate different search operators, and where agents may not be able to provide specific information to other agents or understand information received from other agents. The negotiated-search algorithm is particularly suitable to this style of problem solving because 1) the required search operators represent standard search capabilities; 2) the search operators can be flexibly assigned across the agent set according to the search capabilities of each agent; and 3) agents use incremental relaxation of solution requirements to reach mutual acceptability as an inherent part of problem solving.

3.1 The Search Process

Search is initiated by a problem specification that details the form of a solution and values, preferences, or constraints on some attributes of that solution. This specification is placed in a centralized shared memory as are emerging composite solutions.[3] Some agent(s) uses constraining information from the specification and its local solution requirements to propose an initial partial solution called a *base proposal* The base proposal is then extended and evaluated by other agents during future processing cycles. When a particular solution cannot be extended by some agent due to conflicts with existing solution attributes, there are two possible outcomes: 1) if the conflict is caused by the violation of some hard (non-relaxable) requirement, the solution path is pruned (e.g., arc 5 in Figure 1); or 2) if the conflict is caused by the violation of some soft (relaxable) solution requirement, the solution is saved and viewed as a potential compromise (e.g, arc 9 in Figure 1). In the first case, no more work will be done on that solution, and, to the extent that the violated requirement can be communicated to and assimilated by other agents, future counter-proposals will not violate that same requirement. In the second case, the violated requirement may eventually be relaxed and, if that happens, the potential compromise will become a viable solution again. Future counter-proposals will take the violated requirement into account but are not guaranteed to avoid the same conflict, since other alternatives may be worse.

In both of the above cases, conflict is used as the trigger for the communication of feedback information. In multi-agent systems, it is always problematic to decide what information should be exchanged and when that exchange should take place. In general, agents want to minimize the amount of information they share since it is expensive both to communicate information and to assimilate information. On the other hand, sharing information that will specifically help another agent avoid future conflicts is generally cost effective since it eliminates the expense of generating unproductive solution paths [Lander, 1993]. In negotiated search, an agent that receives conflict information from another agent can choose whether or not to prune its own search to respect that information (see Section 4.5).

Multiple solution paths can be concurrently investigated in negotiated search. Agents are free to initiate solutions at any time either because there aren't any promising solutions in the current solution set or because they have no other work to do. Advantages to maintaining multiple paths include exploiting the potential for concurrent activity and having the ability to directly compare different potential compromises. There are disadvantages to concurrently exploring multiple solution paths however: there will be multiple partial solutions that have to be stored at all times, requiring additional memory resources. There is also overhead involved in focusing on a promising solution path at a particular point in problem solving, both from the local and global perspectives, and in managing the links between solution components along each path. The number of open solution paths is highly dependent on the domain, the number of agents, and the control policies of individual agents. This number can be controlled through parameter settings in TEAM and through the specification of which negotiated-search operators will be active for each agent in the agent set.

3.2 A State-Based View of Negotiated Search

Figure 1 provides a state-based view of the transition of a composite (shared) solution from its initial state (a problem specification) to a termination state (an infeasible solution, an unacceptable solution, or a complete acceptable solution). In this figure, states are defined in terms of three attributes of composite solutions: *acceptability, completeness,* and *search-state.* The possible values for *acceptability* are *acceptable, unacceptable,* and *infeasible.* Possible values for *completeness* are *complete* and *incomplete.* Note that *complete* means that all agents have had the opportunity to extend or critique the solution. A solution with all required components can still be waiting for critiques from other agents and is not considered complete in that case. *Search-state* can take the values *initial* or *closed.*

A negotiated-search operator is a search function applied by an agent. Each operator has a generic form that is expressed in an agent language defined by TEAM, spec-

[2] Agents may control different resources and have different constraints on solutions, but they share a single underlying problem-solving paradigm and knowledge representation.

[3] Each agent also has a local short-term memory where it stores intermediate results and/or component proposals that are linked to composite solutions in shared memory.

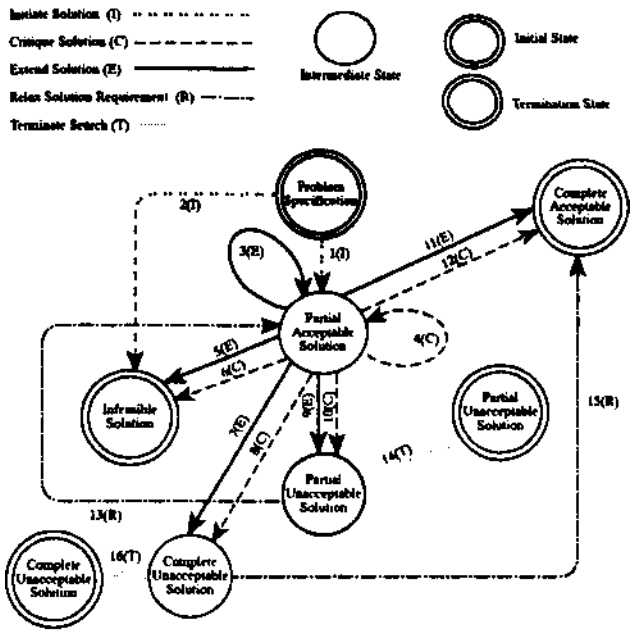

Figure 1: *A State-Based View of Negotiated Search*

ifying its inputs, outputs, and functionality. The decision to apply a particular operator to a problem-solving situation is made by an agent within its local view of the problem-solving situation. The arcs in Figure 1 are negotiated-search operators that can be applied by some agent to a solution.

Each agent instantiates one or more of the negotiated-search operators: *initiate-solution, extend-solution, critique-solution,* and *relax-solution-requirement* In addition, TEAM instantiates the *terminate-search* operator. These operators will be described in detail below, but we provide an overview here to provide a sense of their functionality. *Initiate-solution* is applied by an agent to generate a *base proposal* that will be used as the basis for a new composite solution. *Extend-solution* is applied by an agent to: 1) add a component proposal to a composite solution; 2) evaluate the composite solution from a local perspective; and 3) provide feedback information if conflicts are detected. *Critique-solution* is applied to: 1) evaluate a composite solution (without generating a component proposal); and 2) provide feedback information if conflicts are detected. *Relax-solution-requirement* is applied to: 1) select a local requirement to relax; 2) update the local database to effect the relaxation; and 3) reevaluate existing solutions in light of the relaxation. *Terminate-search* is applied by TEAM to change the state of the problem solving from *initial* to *closed,* thereby changing the termination status of solutions.

The negotiated-search algorithm is applied by a set of agents, *A.* Let $\mathcal{A} = \{A1, A2, A3\}$ and assume that *Al* initiates a solution, *A2* extends the solution, and *A3* critiques some aspect of that solution. We examine a typical search in which a conflict occurs. *A1* first applies the operator *initiate-solution* to a problem specification and produces a partial acceptable solution (arc i). Then *A2* applies *extend-solution* without detecting a conflict. Although the solution now has all components specified, it is not *complete* until all critiques have also been received. Therefore the solution is now partial and acceptable *(arc S)* . A3 next applies *critique-solution,* detects a conflict, and evaluates the solution as unacceptable (are *8).* This solution remains as it is for some amount of time while the agents are working on other solution paths. When further search fails to produce an acceptable solution, *A3* decides to relax the requirement that made this solution unacceptable. The solution is now acceptable to *A3* and, since it was already complete, it reaches the termination state of complete acceptable solution *(arc 15).* In this way, various paths through the state diagram can be achieved by the agent set.

Although the above example describes a sequential ordering of operator applications, TEAM permits concurrency except where there are domain-dependent operator preconditions that force sequential execution. Concurrency requires that TEAM have mechanisms for handling conflicts that occur due to the simultaneous development of extending proposals and criticisms. These mechanisms are discussed in [Lander, 1993].

4 Negotiated Search Operators

In this section, we present a detailed description of the negotiated-search operators. Notice that the operators depicted in Figure 1 work at the surface level of problem solving: they move a particular solution through various states to a termination state. They do not address the issue of feedback and its effect on problem solving. Later in this section, we will present two operators that an agent applies to assimilate conflict information into its knowledge base, thereby refining its view of the search space.

4.1 Initiate-Solution

Initiate-solution is the basic operator for initiating solutions. It is applied within the agent's view of solution requirements: local requirements, those imposed by the problem specification, and any known external requirements learned from other agents. Given these requirements, it creates the *base proposal. Initiate-solution* is executed by one or more agents at system start-up time, and may be repeatedly executed as earlier proposed solutions are rejected by other agents or if alternative solutions are desired. If earlier solutions have been proposed and rejected, the initiating agent may have received conflict information that will influence the generation of new base proposals.

At least one agent must instantiate *initiate-solution;* however, instantiating it at multiple agents is likely to result in a more diverse set of solution paths and more thorough coverage of the composite solution space. Depending on characteristics of the agents and agent set, it may also have a distracting effect. Trade-offs between coverage and distraction are a ubiquitous problem in distributed systems and are discussed generally in [Lesser and Erman, 1980] and specifically with respect to negotiated search in [Lander and Lesser, 1992b],

When no base proposal can be found under the existing set of requirements, an agent can relax requirements

to expand the search space. If there are requirements on solutions that come from information communicated by another agent (external requirements), the initiating agent can ignore one or more of these requirements in its own search. Notice that the other agent does not actually relax the requirements. In this way, each agent chooses the set of requirements, both internal and external, it will attempt to satisfy. When known external requirements are violated, the proposal is suggested as a possible compromise rather than a fully acceptable solution. The external agent that has its requirements violated in the compromise proposal cannot be forced to accept it. Because the selection of a mutually acceptable solution is democratic, each agent votes on the acceptability of a solution. The external agent that has the violated requirement(s) can initially vote that the solution is unacceptable but, if it does not find a better alternative, it may eventually agree to accept this compromise.

If there are no relaxable external solution requirements or if the external requirements are inflexible, an agent can relax some local requirement. If no base proposal can be found at any level of external or internal requirement relaxation, the agent returns a failure along with any conflict information it can generate that describes why it failed. TEAM returns a failure if no agent can generate a new base proposal and all previously created solutions have been found to be *infeasible.*

4.2 Critique-Solution and Extend-Solution

The *critique-solution* operator is applied by an agent to evaluate a partially or fully specified composite solution. The *extend-solution* operator is applied by an agent to extend and evaluate a partially specified composite solution. These two operators will be described jointly because of their similarity. The input for these operators is a composite solution that was initiated by another agent. The output for *critique-solution* is an evaluation, and when a conflict is detected, conflict information. The output for *extend-solution* is a proposal, an evaluation, and, when a conflict exists, conflict information.

The *extend-solution* operator is required in domains where solutions comprise interacting components and each component is developed by an expert agent. The component that an agent develops with *extend-solution* must be compatible with the solution being extended (it must have the same values for solution variables that overlap). The agent executing the operator searches for a compatible proposal under its known solution requirements and the requirements imposed by the assigned parameter values of the solution to be extended.

Although we will not discuss *critique-solution* further, the following discussion of *extend-solution* generally applies to both operators, except that *critique-solution* evaluates the existing composite solution rather than creating and evaluating a compatible proposal. In *extend-solution,* if a compatible proposal is found that does not violate any local solution requirements, it is returned as an *acceptable* proposal. If the best compatible proposal found violates some relaxable (soft) local solution requirements (where the best proposal is one that maximizes local evaluation), it is returned as *unacceptable* along with information that describes the conflict. Although currently unacceptable, future requirement relaxations may change its status and, therefore, the solution is saved as a potential compromise. In the final case, no compatible proposal can be found without violating nonrelaxable (hard) requirements of the executing agent. In this case, the agent fails and the solution path is marked as *infeasible.* Conflict information is returned whenever possible that describes why the path is infeasible, i.e., what hard requirements were violated.

4.3 Relax-Solution-Requirement

Relaxation of solution requirements is a necessary part of negotiated search. In order to terminate problem solving, agents must reach mutual acceptability on one or more solutions. Acceptability is defined as an attribute of a composite solution as shown in Figure 1. If any agent locally evaluates a solution as unacceptable, the solution is considered globally unacceptable. However, as can be seen in that figure, a solution that is unacceptable at some point in time can later become acceptable when the agent or agents that reject it relax their solution requirements.

There are three primary forms of relaxation, *unilateral relaxation, feedback-based relaxation,* and *problem-state relaxation.* Unilateral relaxation occurs when an agent decides to relax a requirement due to its inability to find a solution under the problem specification, i.e., the agent finds that, given the problem specification and its initial solution requirements, it cannot produce a locally acceptable proposal. This situation occurs in the application of the *initiate-solution* operator as described in Section 4.1.

Feedback-based relaxation occurs when an agent relaxes a solution requirement because of some explicit information about the requirements of some other agent(s), i.e, a conflict is found between relaxable local solution requirements and less flexible external solution requirements. This occurs when external information has been received by an agent and is being assimilated as described in Section 4.5.

Problem-state relaxation is a reaction to the lack of overall problem-solving progress. In the current TEAM framework, problem-state relaxation occurs at specific processing-cycle intervals: for example, all agents may relax a solution requirement after 10 processing cycles. Alternatively, the user can specify the relaxation parameter separately for each agent, so that one agent may relax after 10 processing cycles while another will relax after 20 processing cycles. Problem-state relaxation occurs because the problem may be overconstrained by the full agent set. The ability to formulate, communicate, and assimilate constraining information is not guaranteed to be complete and precise across the agent set and the reality is that agents can't always determine whether the composite search space is overconstrained. Therefore, they must have some heuristic method (as well as the deterministic methods above) for deciding when it is

appropriate to relax requirements.[4] Because of problem-state relaxation, we can guarantee that if any initial proposal is generated that can result in a feasible solution, either that solution will eventually become acceptable to all agents, or some other solution will become acceptable to all agents and deadlock will not occur.

4.4 Terminate-Search

The operator *terminate-aearch* is applied by TEAM, rather than by an agent, to change the search phase of the algorithm from *initial* to *closed* when some (user-specified) number of acceptable proposals been found.[5] As seen in Figure 1, when this change occurs, partial and complete *unacceptable* solutions move from intermediate to termination states. Any partial *acceptable* solutions are completed however to ensure that good partial solutions are not abandoned.

4.5 Assimilating Information

There are two operators associated with assimilating information at an agent: *store-received-information* and *retrieve-information. Store-received-information* takes conflict information from other agents, syntactically checks to see if the information already exists in the local knowledge base and, if not, stores it so that it can be retrieved. A received requirement may be indexed by various attributes including the name of the sending agent, the flexibility of the requirement, the names and acceptable values of constrained solution attributes, and, in the case of ordered solution attributes, whether the requirement defines a minimum or maximum boundary on potential values, e.g., $x > 5$.

Retrieve-information is an operator that extends or replaces an agent's default capability to retrieve relevant constraining information from its knowledge base. Because an agent's internal knowledge is expected to be locally consistent, the default retrieval mechanism generally does not handle cases where conflicts may exist in the retrieved requirements. Requirement retrieval occurs during solution initiation, extension, and criticism. The goal of the retrieval process is to find the most restrictive, but non-conflicting, set of solution requirements that constrain a solution for the current local search problem. Different types of requirements require different treatment, but to provide a concrete example of retrieval, we present the algorithm used for selecting boundary constraints on numerical solution attributes in our application systems. Potentially relevant constraints are retrieved and sorted into maximum and minimum boundary groups. The most restrictive maximum constraint (MAX) and the most restrictive minimum constraint (MIN) from each group are selected (where most restrictive means the highest value from the MIN group and the lowest value from the MAX group). Then the algorithm loops through the following sequence until a set of minimum and maximum values is found or until it is determined that no non-conflicting set exists.

LOOP: If the value of MAX is greater than or equal to the value of MIN, return MAX and MIN since a non-conflicting set has been found. Otherwise, if the flexibility of MAX is greater than the flexibility of MIN select the next most restrictive maximum constraint (MAX) and go to LOOP. Otherwise, if the flexibility of MAX is less than the flexibility of MIN, select the next most restrictive minimum constraint (MIN) and go to LOOP. Otherwise, the flexibility of MAX is equal to the flexibility of MIN. Then: if MAX is locally owned, select the next most restrictive minimum constraint (MIN) and go to LOOP. If MAX is not locally owned and MIN is locally owned, select the next most restrictive maximum constraint (MAX) and go to LOOP. If neither MAX nor MIN is locally owned, select the next most restrictive minimum constraint (MIN) and go to LOOP.

In reusable agent sets, operator diversity is expected—not every agent will instantiate every operator including the *store-received-information* and *retrieve-information* operators. Because of this, when an agent formulates and sends conflict information to another agent, there is no guarantee that the receiving agent will use that information appropriately. Therefore, although conflict information is shared willingly and cooperatively in negotiated search, agents do not depend on other agents to react in a fixed way to that information.

4.6 Agent-Level Control of Operator Application

Figure 1 describes domain-independent state preconditions that must be satisfied before an agent can apply one of its operators to a particular solution. However, because there are multiple solution paths, and because some operators are not directly involved in solution generation (e.g., *store-received-information),* an agent may have multiple operators ready to execute at any given time. The order in which an agent schedules local operators is not mandated by either TEAM or by the negotiated-search algorithm. However, because an agent's perception of the world changes over time, the order in which particular operators are executed does affect system performance and the effect of local scheduling on the overall behavior of the system should be considered. Some general policies for local scheduling are useful in most situations, i.e., agents should assimilate any new information received before initiating or critiquing solutions. The degree of sophistication required in local scheduling though is highly dependent on the application and the complexity of required interactions.

5 Conclusions

Negotiated search is a flexible and widely applicable distributed-search algorithm. It specifically addresses issues that arise in multi-agent systems comprised of reusable and heterogeneous agents. The algorithm acknowledges the inevitability of conflict among the agents, and exploits that conflict to drive agent interaction and guide local search.

[4]Using the number of processing cycles as a heuristic is a simplistic approach. More sophisticated mechanisms for applying problem-state relaxation based on characteristics of problem-solving situation, rather than on time, are discussed in [Lander, 1993].

[5]This is a simplified version of the TEAM termination policy that integrates agent acceptability and, optionally, a domain-dependent global evaluation of solutions.

Negotiated search has been implemented in TEAM, a generic framework for the integration of reusable agents, and consequently, in two application systems built on top of TEAM: STEAM (a seven-agent system for the mechanical design of steam condensers); and AGREE (a two-agent system for buy/sell contract negotiation). Testing and analysis of the algorithm within the context of the application systems is described in other work [Lander, 1993; Lander and Lesser, 1992a; Lander and Lesser, 1992b]. Results from experiments conducted with negotiated search show that the algorithm can produce high-quality solutions. They also support the claim that the algorithm is flexible enough to work in reusable-agent systems where the search operators are randomly distributed across the agent set. We see negotiated search as a default algorithm—one that will provide reasonable solutions in a reasonable amount of time without problem-specific customization. As a complementary approach to developing this general algorithm, we are developing customized algorithms that require specific agent characteristics or inter-agent relationships to exist. By taking advantage of these characteristics, it is often possible to improve solution quality and/or processing-time performance. TEAM supports the dynamic selection of a search algorithm, thereby enabling an agent set to switch to a customized algorithm if the requirements for application of the algorithm are met. This work is described in [Lander, 1993].

Acknowledgements

We thank Margaret Connell and Kevin Gallagher for their support in this project. GBB, a system integration tool from Blackboard Technology Group, was used as the basic platform on which our framework was built.

References

[Adler et al, 1989] Mark R. Adler, Alvah B. Davis, Robert Weihmayer, and Ralph W. Worrest. Conflict-resolution strategies for non-hierarchical distributed agents. In Michael N. Huhns, editor, Distributed Artificial Intelligence, Volume 2, Research Notes in Artificial Intelligence. Pitman, 1989.

[Cammarata et al, 1983] S. Cammarata, D. McArthur, and R. Steeb. Strategies of cooperation in distributed problem solving. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pages 767-770, Karlsruhe, Federal Republic of Germany, August 1983.

[Conry et a/., 1992] S.E. Conry, K. Kuwabara, V.R. Lesser, and R.A. Meyer. Multistage negotiation in distributed constraint satisfaction. IEEE Transactions on Systems, Man and Cybernetics—Special Issue on Distributed Artificial Intelligence, January 1992.

[Davis and Smith, 1983] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. Artificial Intelligence, 20:63-109, 1983.

[Durfee and Montgomery, 1990] Edmund H. Durfee and Thomas A. Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In Proceedings of the Eighth National Conference on Artificial Intelligence, pages 86-93, Boston, Massachusetts, August 1990.

[Klein, 1991] Mark Klein. Supporting conflict resolution in cooperative design systems. IEEE Transactions on Systems, Man, and Cybernetics, 21(6):1379-1390, November/December 1991.

[Laasri et al., 1992] B. Laasri, H. Laasri, S. Lander, and V. Lesser. Toward a general model of intelligent negotiating agents. The International Journal on Intelligent Cooperative Information Systems, 1992.

[Lander and Lesser, 1992a] Susan E. Lander and Victor R. Lesser. Customizing distributed search among agents with heterogeneous knowledge. In Proceedings of the First International Conference on Information and Knowledge Management, pages 335-344, Baltimore, Maryland, November 1992.

[Lander and Lesser, 1992b] Susan E. Lander and Victor R. Lesser. Negotiated search: Organizing cooperative search among heterogeneous expert agents. In Proceedings of the Fifth International Symposium on Artificial Intelligence, Applications in Manufacturing and Robotics, pages 351-358, Cancun, Mexico, December 1992.

[Lander, 1993] Susan E. Lander. Distributed Search in Heterogeneous and Reusable Multi-Agent Systems. PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1993. In preparation.

[Lesser and Erman, 1980] Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and experiment. IEEE Transactions on Computers, C-29(12):1144-1163, December 1980.

[Neches et al, 1991] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Enabling technology for knowledge sharing. AI Magazine, 12(3):36-56, Fall 1991.

[Sathi and Fox, 1989] Arvind Sathi and Mark S. Fox. Constraint-directed negotiation of resource reallocations. In Les Gasser and Michael Huhns, editors, Distributed Artificial Intelligence, Volume 2, pages 163-193. Pitman, Morgan Kaufmann Publishers, 1989.

[Sycara et al, 1991] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. IEEE Transactions on Systems, Man and Cybernetics, Fall 1991.

[Sycara, 1985] Katia Sycara. Arguments of persuasion in labour mediation. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, pages 294-296, Los Angeles, California, 1985.

[von Martial, 1992] Frank von Martial. Coordinating Plans of Autonomous Agents. Lecture Notes in Artificial Intelligence, Springer-Verlag, 1992.

[Werkman, 1992] Keith J. Werkman. Multiple agent cooperative design evaluation using negotiation. In Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh, PA, June 1992.

[Yokoo et al, 1992] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In Proceedings of the Twelfth International Conference on Distributed Computing Systems, Yokohama, Japan, June 1992.

[Zlotkin and Rosenschein, 1990] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and conflict resolution in non-cooperative domains. In Proceedings of the Eighth National Conference on Artificial Intelligence, Boston, Massachusetts, July 1990.