# Customizing Distributed Search Among Agents with Heterogeneous Knowledge

Susan E. Lander and Victor R. Lesser
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
Email: lander@cs.umass.edu

## Abstract

*Negotiated search*, a paradigm for cooperative search and conflict resolution among heterogeneous, reusable, expert agents has been implemented in a flexible framework, TEAM. We present negotiated search and discuss the use of customized *negotiated-search strategies* that take advantage of specific capabilities and relationships that exist in an agent set. Strategies are dynamically selected based on the individual agents' views of the problem-solving situation and on communicated knowledge about the characteristics of agents in the agent set. We present experiments that show strategies can reduce the amount of search required to find mutually-acceptable solutions and can improve the quality of those solutions. The use of customized negotiated-search strategies has far-reaching implications for the design of agents that are intended to be reusable and for the assembly of agents into application systems.

## 1 Introduction

We present *negotiated-search*, a distributed-search paradigm for cooperation among heterogeneous expert agents. Agents can be both *logically heterogeneous* and *implementationally heterogeneous*. Logically heterogeneous agents may have different long-term knowledge (expertise), goals, views or perspectives on objects and relationships in the domain, constraints or preferences, or criteria for evaluating solutions. Implementationally heterogeneous agents may have different knowledge representations, languages, architectures, inference engines, software tools, or specialized processors. Integrating agents with these kinds of differences into a cooperative set requires either: 1) an intensive knowledge engineering/design effort at system-development time; or 2) a sophisticated framework for supporting information exchange and independent agent-level capabilities at run-time.

Run-time support for agent integration further enables agent *reusability*: an agent can be built generically with the intention of using that agent in multiple problem sets rather than a single problem set [16]. In this paper, we describe a framework that supports the integration of heterogeneous, reusable agents, and a generic algorithm for cooperative distributed search by agents within the framework. We then discuss how distributed search can be customized to take advantage of characteristics of both the individual agents within an agent set, and inter-agent characteristics of the agent set itself. Customized search strategies may be more efficient and/or result in better solutions than a generic distributed search algorithm, but cannot be uniformly applied across heterogeneous agents. In order to apply a particular strategy, it must be known to all participating agents, and some agent must be able to perform each required task. It may be the case that agents must meet strict criteria to participate in a strategy, such as using a particular knowledge representation. We describe the information flow required among agents to enable them to select and apply the most effective strategy possible within the current agent set.

Agent sets are dynamically formed by grouping agents with the specific expertise required for the problem. Agents do not have *a priori* knowledge of what other agents will be included in the set and what their capabilities will be. The agent sets we are considering are cooperative, i.e., agents are not hostile and will not intentionally mislead or otherwise try to sabotage another agent's reasoning. Agents are willing to contribute both knowledge and solutions to other agents as appropriate and to accept solutions that are not locally

This research was supported by ONR Contract #N00014-89-J-1877, DARPA Contract #N00014-92-J-1698, and by a University Research Initiative Grant, Contract #N00014-86-K-0764. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

optimal in order to find a mutually-acceptable solution. However, even in this cooperative environment, conflicts are inherent in the agent set due to inconsistent knowledge among agents, incomplete knowledge and/or incorrect assumptions, different problem-solving techniques, and different criteria for evaluating solutions. It is not possible to anticipate and engineer out all potential conflicts at agent-development time since it is not known what knowledge will be contained in the complete system [7]. Thus, conflict resolution is an integral part of problem solving among logically heterogeneous agents and is an important aspect of negotiated search.

There are two basic ways to characterize approaches to conflict resolution in negotiated search: *extended search* and *relaxation*. The first, *extended search*, is applied by an agent when it recognizes a conflict with another agent in an existing solution. The agent sidesteps the conflict by extending its local search until a solution is found that does not conflict. Extended-search methods are used when an agent believes that a mutually-acceptable solution can be developed if it continues to examine its local solution space for additional solutions.

The second negotiated-search method, *relaxation*, occurs when an agent relaxes some requirement on a solution, thereby expanding its local search space. Relaxation may lead directly to a solution: if a history of existing, but unacceptable, solutions is kept, one of these solutions may become immediately acceptable. If not, extended search can now be applied in the expanded space to increase the chance of finding a solution. Relaxation methods are utilized when the solution space is believed to be overconstrained or when the expense of further local search is unjustified.

Both extended search and relaxation are realized as *negotiated-search operators* within an agent. A negotiated-search operator is defined as an agent-independent entity with specific inputs, outputs, and functionality. It is instantiated in a domain-dependent way by a particular agent within that agent's architectural and representational restrictions. General methods that can be implemented as negotiated-search operators for extended search include heuristic search, searching for alternate goal expansions, or case-based search. General methods that can be implemented as relaxation operators include relaxing or relinquishing constraints, relaxing or relinquishing goals, manipulating constraints (e.g., unlinking, bridging [17]), or manipulating evaluation criteria [22].

In negotiated search, two interwoven processes occur: first, *local search* for an optimal solution to a subproblem under the local requirements for solutions; and second, *composition*[1] of local subproblem solutions into an overall solution. Local search is guided by the domain expertise of an agent. Composition relies on the group problem-solving skills of each agent: communication, coordination, evaluation of mutual acceptability, and assimilation of externally provided information. Effective integration of these two processes is captured in the `TEAM` framework described in Section 2.

A difficult problem in any multi-agent system is to decide whether or not a solution is acceptable. We present a two-level view of solution evaluation, taking into account both local agent utility measures and a global objective function. Consider an example: an agent produces its *best* proposal where "best" is defined as minimal cost. When this proposal is used as the basis for a complete solution however, all other agents must produce proposals that are higher cost than they might have been with some other base proposal. The overall effect is that the cost of the solution is higher than it would have been had the first agent produced a higher cost base proposal. Due to its local perspective, the agent could not predict this effect: some global perspective is required to reasonably evaluate solutions.

Although a global objective function provides an agent-independent measure of solution quality, there is reason to respect the individual agents' criteria for evaluating solutions as well. The information needed to calculate and evaluate solution attributes is often embedded in local expertise. For example, in mechanical design a solution often consists of interacting, but fairly autonomous, components. For some components, life expectancy may be an important attribute. Component life expectancy can only be calculated at the agent level, however, because it involves procedures and information that are private to an agent. Global objective functions provide a way to rank alternative solutions on attributes that do not require domain expertise to evaluate (cost, for example) while local criteria are used to evaluate solutions using agent-level expertise. Thus, we attempt to maximize the global evaluation while respecting local utility.

Within the `TEAM` framework, it is possible to find mutually-acceptable solutions with loose coordination between agents because agents are able to work independently and asynchronously. However, more effective behavior is likely to be achieved using a *negotiated-search strategy* that guides the selection of operators to execute at each agent. A negotiated-search strategy is designed to reduce the amount of search required to find mutually-acceptable solutions by taking advan-

[1] Sathi similarly uses the term *composition* as the name of a specific negotiated-search operator that combines local information [18].

tage of specific attributes of both individual agents and agent sets. The use of a particular strategy requires that certain capabilities, characteristics, and relationships exist among some agents in the agent set. This is a problem that is unique to heterogeneous multi-agent systems where the agents do not share an underlying and integrated architecture and where these attributes are not explicitly built into the agents and agent set. We will describe a negotiated-search strategy and its required attributes in Section 6.

Section 2 describes TEAM. In Section 3, we relate our work to that of other researchers. Requirements for knowledge consistency are discussed in Section 4. Section 5 gives a brief description of the negotiated-search paradigm and its realization in the TEAM framework. A more detailed description is available in [11]. Section 6 presents the general concept of negotiated-search strategies, defines a specific strategy called *linear-compromise*, and compares the performance of the system with and without linear-compromise.

## 2 The TEAM Framework

The TEAM framework supports loosely coupled, heterogeneous agents engaged in problem-solving through negotiated search. Agents are distinct and independent entities that communicate through a shared memory.[2] In TEAM, agents can be added to or deleted from an agent set using a standard procedure detailed in [11]. To enable this level of agent autonomy, information in the system is partitioned in layers of abstraction. There are three major classes of information: one that is accessible only to the system, one that is accessible to all agents, and one that is accessible only to an individual agent. The architecture and knowledge partitions of the TEAM framework are shown in Figure 1. By partitioning information in this way, agents can operate within the framework without having detailed knowledge about other agents in the set or about the implementation of data structures within the framework. When an agent wants to make changes to shared memory, it sends a message to the framework controller and the controller makes the actual changes. For example, an agent may ask the framework controller to create a new solution from a base proposal it has generated. The controller will build an empty solution object, copy values from the proposal to the solution, and install the new solution in shared memory.

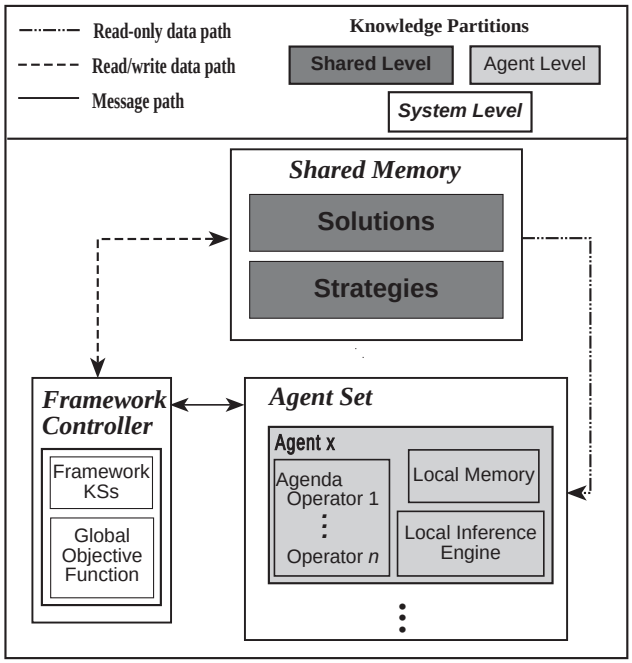


Figure 1: *Architecture of the TEAM Framework*

The shared memory, framework structures, and framework functions are implemented as a blackboard system using a blackboard shell, GBB. Framework functions or, more precisely, framework knowledge sources (KSs), operate on shared memory objects either in direct response to a message from an agent or to perform maintenance and bookkeeping operations on those objects. For example, a framework KS is responsible for linking proposals from different agents into a single solution. This functionality is domain-independent. The domain language, however, is system-specific and must be shared by all agents. Notice though that the agents do not need to use the shared language internally as long as they can use whatever subset of it is required for communication.

During processing, there are two distinct phases: 1) an agent cycle; and 2) a framework cycle. During the agent cycle, each agent is invoked sequentially. The agent uses information in shared memory to choose applicable negotiated-search operators and add them to its agenda. It then invokes its highest-priority operator and returns the result. After all agents have executed, the framework controller is invoked to update the shared memory based on messages from agents and to propagate the effect of changes to shared memory objects.

[2] Though the current shared memory implementation of TEAM is appropriate for many types of search problems, we are beginning to look at applications in domains that would benefit from a fully distributed architecture. The current architecture can be extended without any loss of functionality or philosophy since centralized mechanisms are limited to domain-independent tasks that could be performed by any agent.

## 3 Related Research

Negotiated search has roots in blackboard problem-solving, constraint-directed search, system integration, and in negotiation, both human and computational. From blackboard problem-solving, we take ideas about shared and competing solutions, opportunism, flexibility, knowledge modularity, and incremental extension of partial solutions [4]. Although the blackboard literature provides a great deal of insight into the behavior of semi-independent knowledge modules [14], `TEAM` provides specific modularity, decentralized-control, conflict-resolution, and coordination capabilities that are not present in traditional blackboard systems[13].

Coordination in multi-agent systems has been extensively investigated [1, 3, 20, 23]. The themes of coordination and conflict resolution are closely related: conflict resolution activities often require coordination and coordination activities often require conflict resolution. Negotiated search focuses on the coordination of agents through recognition of and reaction to conflicts and with respect to agent and agent set characteristics.

An active research area is the integration of systems with different architectures (e.g., neural nets vs. production systems) or algorithms (e.g., case-based vs. rule-based search) where each agent implements the most appropriate technology for the class of problems it is designed to handle [19]. Other investigators are looking at integrating systems with different languages, dialects, communication protocols, and model mismatches [16]. Cooperation among these agents is problematic—an obvious and immediate need is seen for a shared language for interacting objects and events and for integrating machinery. An effort is under way to develop techniques to support sharing of knowledge among systems, The Knowledge-Sharing Effort, sponsored by the Air Force Office of Scientific Research, the Defense Advanced Research Projects Agency, the Corporation for National Research Initiatives, and the National Science Foundation. Whereas the focus of this initiative is the development of the technical infrastructure for knowledge sharing, we investigate high-level knowledge sharing issues as part of a complete framework for agent integration and agent-oriented control.

From constraint-directed reasoning, we adapt a simple representation of constraints to bound local and shared problem spaces. Sophisticated constraint-directed reasoning such as that developed by Fox [6] is not directly applicable to heterogeneous-agent problem solving because its relies on developing an understanding of the solution space through analysis of constraints and constraint variables. In heterogeneous agent problem solving, each agent can analyze its local solution space but there is no effective method for analyzing the shared space in a general way since no agent has enough knowledge about the constraints of other agents. This problem has been addressed somewhat by Sathi [18], Sycara [21], and Mammen [15] in their work on constraint-directed negotiation. However, these investigations of constraint-directed reasoning in multi-agent systems have required that the agents share an underlying integrated problem-solving methodology and agent architecture. Although these agents have heterogeneous resource requirements, they are implementationally homogeneous and use that homogeneity implicitly for control.

The negotiation literature provides insight into the underlying mechanisms of conflict resolution and the development and application of specific negotiated-search operators. Klein [9] has focused on the development of a taxonomy of conflict types. Werkman has developed a multi-agent system that relies on an integrated knowledge representation and mediation framework to discriminate among alternative solutions from different perspectives [24]. Sycara describes a negotiation system, PERSUADER, for a non-cooperative labor/management domain [22]. Because of the hostile nature of the agents, a mediator is used to buffer interactions and make objective decisions. Pruitt [17] and Fisher [5] offer insights into human negotiation. Much of the human conflict-resolution process is driven by the need to protect the egos of the participants and is not directly applicable. However, some ideas about search and creativity in computational conflict resolution have evolved from this work.

## 4 Knowledge Consistency

In this section, we describe how private and shared information is maintained and assimilated by an agent in negotiated search. In [8], Huhns describes a distributed truth-maintenance approach to database management. We do not intend to imply that we use distributed truth-maintenance techniques. However, by examining the definitions of terms presented in that work, we can contrast and justify our approach under the requirements of agent heterogeneity and reusability.

In negotiated search, each agent maintains a local database that contains its domain-level declarative knowledge and information about the current state of problem solving. This information is assumed to be internally consistent. As information is communicated however, consistency is not expected with respect to incoming information.

There are several ways to circumvent the problems associated with inconsistent internal and external knowledge: 1) ignore any externally obtained knowledge that conflicts with internal beliefs; 2) replace internal knowledge with conflicting external knowledge when it is received; 3) allow conflicting internal and external knowledge to co-reside; or 4) always resolve any inconsistency that becomes apparent when information is communicated so that the entire agent set maintains *local-and-shared consistency*. Maintaining local-and-shared-consistency implies that any time an inconsistency is found, some decision will be made as to which information is correct and that decision will be propagated through all agents that have shared the knowledge.

*Global consistency*, the strictest form of consistency, implies that all knowledge is consistent across all knowledge bases, whether or not the agents interact. With reusable, heterogeneous agents, we claim that global consistency is neither feasible nor desirable. From a feasibility standpoint, it is impossible to knowledge-engineer agents that are created independently by different people at different times with different expertise. Furthermore, global consistency requires that different perspectives and beliefs cannot co-exist within an agent set. However, different perspectives can actually be quite beneficial, particularly in creative problem-solving situations [2].

The question then becomes whether or not local-and-shared consistency, as described above, is achievable and desirable. To maintain this level of consistency, it must be the case that every disagreement is fundamentally resolved when it is noticed and the effects of that resolution are propagated throughout the set of participating agents. To do this would be a very large burden, both computationally and philosophically. Computationally, we would want to try to resolve the conflict based on some concrete evidence that one or the other perspective is more valid: perhaps by invoking some deep model of the domain, applying naive physics, maintaining measures of certainty of each agent for a particular piece of information, or collecting collaborating evidence. Although there are certainly situations in which the validity of some agent's belief should be tested, it can be computationally intensive to do so. Furthermore, in many situations, there is no absolute measure of correctness, e.g., agents might have different color preferences for an object. And again, philosophically, it is often not desirable to enforce consistency among agents.

For the reasons given above, we have decided against enforcing any type of consistency across the borders of agents. The decision about how to handle conflicting information is agent- and strategy-specific. However, we have defined a representative operator, *assimilate-information*.

*Assimilate-information* is instantiated at an agent to collect and absorb information that has been communicated by other agents. Information received from other agents is placed in a separate database, the non-local database, rather than in the local database of the agent. Logical connections are established between the two databases that indicate which information should be used during problem solving. When no explicit conflict exists between these two knowledge bases over some shared variable, the most restrictive information available from either is chosen for any required reasoning. For example, if an agent has the local constraint $\{x < 10\}$ and receives the constraint $\{x < 5\}$ from some other agent, it will use the latter in its decision-making since it does not conflict with its own knowledge. However, when there are explicit inconsistencies in the two knowledge bases, an agent must choose what information it will use in making future decisions. Figure 2 shows the process of assimilating conflicting knowledge. The agent can decide to either: 1) override local knowledge and use the conflicting external information; 2) ignore conflicting external information and use the local knowledge; or 3) relax local solution requirements to reduce the inconsistency and reexamine the situation within the new solution boundaries. By allowing conflicting information to co-reside within an agent, decisions about what information to use in problem solving can be made dynamically based on the current problem-solving situation.

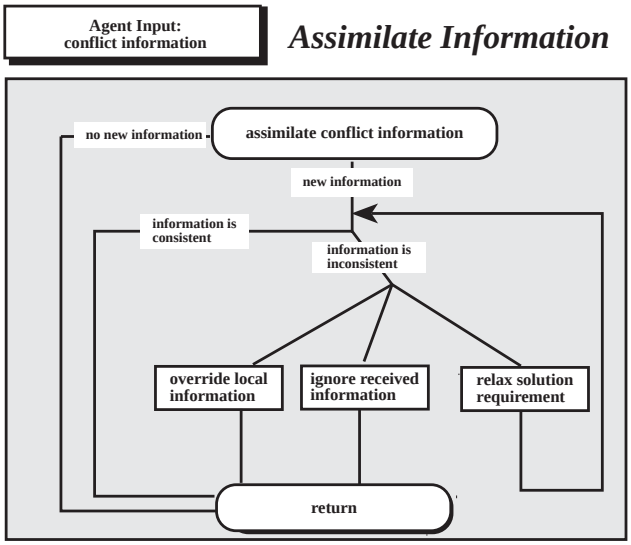


Figure 2: *Assimilating Conflict Information*

A possible selection mechanism (implemented in TEAM) attempts to fairly distribute the burden of relaxation. Each piece of information has an associated flexibility value that measures the degree to which an agent is willing to relax that information. When assimilating conflicting information, an agent will choose to relax its own solution requirements if its own information is more flexible than the received information. If the received information is more flexible or equally flexible, the agent will ignore it. However, if the received information places a hard constraint on solutions (a constraint that applies to all feasible solutions), the agent will override any local information that conflicts.

## 5 Negotiated Search

In this section, we give an overview of the realization of the negotiated-search paradigm in TEAM. We begin with a description of how the local problem-solving efforts of the agents are integrated. Next, the mechanisms used in local problem-solving at the agent level are presented.

### 5.1 Integration of Local Search

Agents communicate and coordinate their efforts through a high-level view of problem-solving, maintained in a shared memory, and accessible to all agents. Each agent works on some subproblem(s) and produces *proposals* that represent subproblem solutions. Proposals are integrated into shared solutions in shared memory. Each solution is initiated by the generation of an agent proposal, called the *base proposal*, that partially specifies the parameters of a complete solution.

Solutions are critiqued by other agents. Conflicts are determined locally by the agent critiquing the solution. For example, an existing solution specifies a value of 6 for the parameter $x$ and an agent criticizing that solution has the constraint $\{x > 7 \ \ (5, 3, 1)\}$ where $(5, 3, 1)$ indicates potential relaxations of $x$. In other words, the preferred value for $x$ is greater than 7, but other values are feasible. Each relaxation lowers the utility of the solution to the agent. The agent is able to respond with a feasible proposal given that $x = 6$ but a conflict occurs because the value is not within the preferred range. Therefore, conflict information (the violated constraint $\{x > 7\}$) will be returned by the agent along with a local acceptability value. In the example above, the solution would initially be rated as *unacceptable* to the agent because it violates a preference constraint. However, at some future point, the agent might decide to relax that constraint to the next level, $\{x > 5\}$. If this occurs, the agent would change its evaluation of the solution to *acceptable*.

The acceptability of a solution is measured by two factors. First, all agents must consider the solution to be acceptable. Second, the user can specify a global objective function to be applied to the complete solution that must meet or exceed a threshold value. If a solution is complete (all agents have examined the solution) and acceptable, the system will add it to the set of completed solutions and will stop if it has met a user-defined quota on the number of alternative solutions required. If the solution is not complete, it stays active while waiting for critiques from other agents. Agents locally schedule their own activities and may respond at different times.

Unacceptable solutions are saved along with information about why they were unacceptable. A solution that is unacceptable because of local constraint violation(s), such as the $(x > 7)$ constraint described above, can be considered a potential compromise. If requirements are relaxed at some future point, this compromise may become acceptable and the solution will be reactivated.

### 5.2 Local Search

Local search is performed individually by an agent within its current view of the shared solution space. This view includes solution requirements stored in the local database, the nonlocal database, and requirements imposed by the user-defined problem specification. Each search action is represented by a negotiated-search operator. For the system described in Section 6, we define three representative operators: *initiate-solution*, *critique-solution*, and *relax-solution-requirement*[3] Each of these operators has a defined functionality that is implemented locally by each agent in a style consistent with the agent's architecture, knowledge representation, and inference engine.

*Initiate-solution* is a negotiated-search operator that an agent instantiates to generate a base proposal that satisfies its locally-known set of constraints, constraints imposed by the initial solution specification, and any constraints that have been "learned" through the assimilation of conflict information provided by other agents. This operator is applied at system start-up time by at least one agent, and may be applied again throughout problem-solving when existing solutions have not led to promising solution paths.

[3] Another system, STEAM, has been implemented that uses a more complex generic algorithm and a larger set of operators. STEAM is described in [12, 11].

*Critique-solution* is an operator that takes an externally initiated solution as input and checks to see if any local solution requirements or preferences are violated in that solution. The agent returns its evaluation of the solution, an acceptability value for the solution, and conflict information if a conflict has been found.

*Relax-solution-requirement* changes the acceptable threshold level of some local solution requirement and propagates that change through any existing solutions (possibly making unacceptable solutions acceptable). It can be invoked unilaterally when an agent fails to find a solution under the current problem specification. It can also be invoked due to nonproductive, iterative negotiation efforts: if an agent has generated multiple base proposals and/or has responded to multiple solutions without finding a suitable fit, there may not be a fit under the current set of solution requirements. It may also be applied when there is an explicit conflict between a local requirement and some nonlocal information that an agent is attempting to assimilate (as shown in Figure 2.

*Relax-solution-requirement* can be used by any agent that has an ordered set of possible local utility values. This ordered set is divided into classes (e.g., excellent, good, fair, poor), assigning a class to each possible value or range of values. For each proposal generated during a particular agent cycle, a constraint is placed on the class of the utility value for that proposal. The class must must be greater than or equal to a threshold. For example at the start of processing, any proposal used to initiate a solution should have a utility value in the *excellent* class. The first time *relax-solution-requirement* is applied, the threshold utility class drops to *good*, and this process will continue through the set of utility classes.

### 5.3 General Negotiated Search

The default negotiated-search strategy used to guide agent interactions is called *general negotiated search* (`GNS`). `GNS` is an opportunistic search augmented by the communication and assimilation of conflict information. One or more agents produce base proposals and other agents critique the partial solutions created from those proposals. There may be multiple partial solutions available at any given time and the selection of one to critique is made locally. If a conflict is detected, any available constraining information is communicated to other agents. Agents that receive conflict information attempt to assimilate that information. If an agent has successfully assimilated conflict information from another agent and later attempts to generate a proposal, the new proposal will avoid that conflict.

An agent may both generate its own base proposals and critique other agents' solutions, depending on which negotiated-search operators it instantiates. The order of operator application is locally specified by the agent designer. If the required number of solutions is not found within a specified number of agent cycles, called the *relaxation threshold*, each agent will apply the negotiated-search operator, *relax-solution-requirement*, to expand the solution space. Existing solutions are reevaluated under the new acceptability standards. If the required number of acceptable solutions still aren't found, agents will continue to initiate and critique solutions.

Agents continue in a cycle of search and relaxation until an acceptable solution is found, or until further relaxation is impossible (at which point a failure is declared). This strategy, `GNS`, is widely applicable and is used in `TEAM` as a default. There are many situations where this level of generality is unnecessary, however, and where much more direct and effective strategies can be applied. We will discuss one such situation in Section 6.

## 6 Strategies

This section examines how a limited, but very efficient, negotiated-search strategy called *linear compromise* can be used in place of the default opportunistic-search strategy described earlier. The linear compromise strategy can be applied when two agents have intersecting linear functions over some variable that describe the utilities of their solutions. The strategy can be extended to include more than two agents as long as each agent calculates its local utility with a linear function over the shared variable. To investigate the effectiveness of this strategy, we have implemented a buyer/seller system in `TEAM`.

In the buyer/seller system, the artifact being contracted for is not specified. A base fair market value for the artifact is randomly generated. Each agent generates its own perceived market value by randomly choosing a value within a specified percentage of the base. This models a system in which market value is determined subjectively, and the agents involved in a transaction may not agree. Each agent then generates either the maximum or minimum price it would accept (depending on whether the agent is minimizing or maximizing price) by randomly generating a value within a specified percentage of its locally-perceived market value. The utility of a contract is based on a normalization of the difference between the contract price and the (fixed) worst acceptable price for that agent.

In the general negotiated-search model described earlier, each agent generates proposals at the currently acceptable level of local utility value. If no agreement is reached after a cycle, each agent determines whether to drop its acceptable level of utility value. Eventually, at least some agents will drop their acceptability levels and generate proposals at the new level. This algorithm repeats until either a solution is found or until the acceptable level of utility value drops below some threshold for each of the agents, at which point negotiated search fails, and no contract is made.

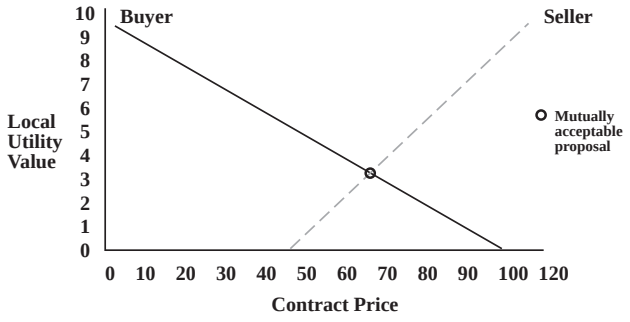


Figure 3: *Intersecting Linear Functions*

In contrast, with intersecting linear utility functions over the value of price, the intersection point of the two functions can be algebraically calculated (Figure 3). This point is the "fairest" value that can be assigned for price since the utility of that solution is equal for each of the agents.[4] The intersection point can be calculated by an agent in one of two ways: 1) the other agent can explicitly communicate information about its utility function; or 2) the required information can be extrapolated from points in the other agent's local solution space. Either or both of these methods could be implemented but we used the latter method which does not require any specialized communication skills on the part of the agents.

When a strategy is applied, each agent has a particular role to play in the development of a solution. In the linear compromise strategy, one agent calculates a solution (*solution-calculator*) based on points generated by the other agent (*point-generator*). It may be that either agent could play either role or one of the agents may only be able to play one of the roles. The requirement for successful application of the strategy is that at least one agent must be capable of taking responsibility for each of the roles.

A role is realized at an agent as a negotiated-search operator. For example, in order to take the *point-generator* role in the *linear-compromise* strategy, an agent must instantiate the *generate-points* operator. This operator implements the functionality defined for that role: that the agent produce two points in its solution space and store those points in a `STRATEGY` object in shared memory. No further restriction is placed on the operator. An agent may provide previously existing points or it may generate new points. Different agents may implement the same operator in different languages, different architectures, or with different search mechanisms for generating points.

Using a computationally simple method with a well-defined agent protocol for interaction to find a compromise is extremely effective. The problem is that in a heterogeneous agent set, the agents must recognize that they are involved in a situation where their shared solution spaces have the appropriate characteristics and where the agents have the capabilities required to develop the compromise solution. The search for a negotiated-search strategy begins when a conflict is detected. Because the agents cannot make assumptions about the available strategies and operators of other agents, there are two distinct phases involved in recognizing that an appropriate strategy exists. First, each agent must individually determine if it is aware of any strategies that fit the locally-perceived situation, and second, some agent(s) must determine if any strategy is applicable over the complete agent set. Consider the linear compromise strategy described above: first, each agent must recognize that it has a linear function that describes utility value for a solution over some variable. Then some agent must recognize that all agents can participate in linear compromise and that all roles can be filled. Once this is established, all agents must be informed that a strategy has been chosen and the agents must be assigned roles. In the buyer/seller system, the agent that initiates a solution becomes the manager for any conflicts that occur in response to the solution[5]. It gathers information from other agents about their potential strategies and roles, selects a strategy that all relevant agents can participate in and assigns roles appropriately. If no strategy is found, the default general negotiated-search strategy described earlier is used. Selection and assignment algorithms and a communication protocol have been implemented for `TEAM` and are described in [11].

Once a strategy has been chosen and roles are as-

[4] With more than two agents, a fair value for price can be calculated with other algebraic methods.

[5] Note that negotiating over possible strategies and role assignments can itself be treated as a conflict situation with the same type of iterative search/relaxation methodology that applies to domain problem-solving. We have adopted a fixed protocol for resolving these conflicts rather than allowing the full range of conflict resolution activities. However, this has been and remains an open research issue[20, 10]

| Run | Buyer's Maximum Price | Seller's Minimum Price | Buyer's Utility (GNS) | Seller's Utility (GNS) | Buyer/Seller Utility (LC) | Contract Price (GNS) | Contract Price (LC) | # Agent Cycles |
|---|---|---|---|---|---|---|---|---|
| 1 | 968982 | 752096 | 4.55 | 4.49 | 4.53 | 909152 | 908615 | 30 |
| 2 | 629300 | 484763 | 4.59 | 5.58 | 4.96 | 568264 | 575015 | 30 |
| 3 | 230783 | 141469 | 7.87 | 7.68 | 7.79 | 196178 | 195638 | 15 |
| 4 | 572992 | 426629 | 7.06 | 7.02 | 7.04 | 488249 | 488087 | 15 |
| 5 | 686769 | 477945 | 3.89 | 4.30 | 4.07 | 588534 | 593710 | 35 |
| 6 | 495098 | 322499 | 7.08 | 7.57 | 7.22 | 443309 | 445691 | 15 |
| 7 | 755135 | 609051 | 5.05 | 5.07 | 5.05 | 738834 | 738883 | 25 |
| 8 | 408309 | 432317 | NA | NA | NA | no solution | no solution | 50 |
| 9 | 536765 | 398897 | 4.99 | 4.64 | 4.86 | 488217 | 485873 | 30 |
| 10 | 545477 | 456101 | 5.06 | 5.04 | 5.04 | 474928 | 474882 | 25 |

Table 1: *Comparison of General Negotiated Search and Linear Compromise*

signed, the strategy is instantiated. The number of further agent cycles that are required to execute the strategy is dependent on the strategy. For *linear-compromise*, there is a constant agent-cycle cost of two. On the first cycle, the point generator communicates the point set and on the second cycle, the solution calculator receives the information and calculates the intersection. In contrast, the agent-cycle cost of the general negotiated-search strategy is on the order of $nr$ where $n$ is the cardinality of the set of possible utility values (e.g., $n = 4$ when possible utility values are *excellent*, *good*, *fair*, and *poor*) and $r$ is the relaxation threshold (as described in Section 5.2). Experiments have been run with the buyer/seller system in which 1) the agent set recognizes and applies the linear compromise strategy, 2) some agent does not recognize the strategy, and 3) appropriate role assignments cannot be made. In Cases 2 and 3, the default negotiated search strategy, `GNS`, executes and eventually finds a solution that reasonably approximates the optimal one achieved by linear compromise.

A comparison of the results obtained by the two different strategies is presented in Table 1. In the `GNS` experiments, the cardinality of the rating set ($n$) is 10, and the relaxation threshold ($r$) is 5, for a maximum agent cycle cost of 50. The agent cycle cost for LC is not shown in the table as it is constant for all runs: agent cycle cost in LC is always 2. The LC contract price is optimal where optimality is defined as having equal seller and buyer utilities.

*Linear compromise* can be thought of as a shortcut to iterative applications of search and relaxation operators. By taking advantage of the situational characteristics, it is possible to algebraically calculate a solution and skip multiple search/relaxation cycles. However, the applicability of the linear compromise strategy depends on the existence of the intra-agent utility/parameter value relationship, the inter-agent linear intersection relationship, and the existence of required capabilities at agents in the agent set.

## 7 Conclusions

In this paper, we introduce the concept of negotiated search to integrate a wide variety of approaches to local-agent search and inter-agent cooperation and conflict resolution. We present `TEAM`, an implemented framework for empirical investigation of negotiated search. We note that a negotiated-search strategy (a coordinated sequence of negotiated-search operators across agents) can take the place of loosely coordinated iterative cycles of search and relaxation if a specified set of conditions exist. By taking advantage of these conditions, the amount of search required can be significantly reduced and the quality of solutions can be improved. *Linear compromise*, a customized negotiated-search strategy implemented in `TEAM`, is analyzed and compared to a more widely applicable strategy, `GNS`. Experimental results are presented that show performance improvements using the customized strategy. We will be exploring other strategies and their requirements through future experimentation with `TEAM`.

The use of customized strategies can improve performance and quality, but depends on the ability of heterogeneous agents to recognize that required agent capabilities and characteristics exist, both locally and across the agent set. Agents must also be able to dynamically select a strategy and adjust their problem solving in accordance with the selected strategy. `TEAM` has proven to be an effective tool for investigating the requirements inherent in the integration of heterogeneous and reusable agents. This work provides a foundation for guiding the design of agents for multi-agent sets, for deciding whether or not a particular agent is an appropriate candidate for inclusion in a particular set, and for determining the role an agent should play.

## Acknowledgments

Margaret Connell and Kevin Gallagher provided suggestions and programming support for `TEAM`. Dorothy Mammen provided helpful comments on drafts of this paper.

## References

[1] 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.

[2] Edward de Bono. *Lateral Thinking for Management, A handbook of creativity*. American Management Association, 1971.

[3] Edmund H. Durfee and Victor R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, Sept/Oct 1991.

[4] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[5] R. Fisher and W. Ury. *Getting to Yes: Negotiating Agreement without Giving In*. Houghton Mifflin, 1981.

[6] M.S. Fox, B. Allen, and G. Strohm. Job-shop scheduling: an investigation in constraint-directed reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–158, Pittsburgh, Pennsylvania, August 1982.

[7] Carl Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.

[8] Michael N. Huhns and David M. Bridgeland. Multiagent truth maintenance. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1437–1445, November/December 1991.

[9] Mark Klein. Supporting conflict resolution in cooperative design systems. In *Proceedings of the 10th Workshop on Distributed Artificial Intelligence*, Bandera, Texas, October 1990.

[10] 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.

[11] Susan E. Lander. *Distributed Search and Conflict Management among Heterogeneous Reusable Agents*. Ph.D. thesis, University of Massachusetts, Amherst, Massachusetts, May 1994. Also available as Technical Report 94-32, Computer Science Department, University of Massachusetts, Amherst.

[12] 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*, Cancun, Mexico, December 1992.

[13] Susan E. Lander, Victor R. Lesser, and Margaret E. Connell. Knowledge-based conflict resolution for cooperation among expert agents. In D. Sriram, R. Logher, and S. Fukuda, editors, *Computer-Aided Cooperative Product Development*, pages 253–268. Springer-Verlag, 1991.

[14] Victor Lesser and Daniel Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, 1983. (also in *Blackboard Systems*, R. Engelmore and T. Morgan (eds.), pp 353-386, Addison-Wesley, 1988).

[15] Dorothy L. Mammen and Victor R. Lesser. Using textures to control distributed problem solving. In *Working Notes from the AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems, AAAI-92*, San Jose, California, July 1992.

[16] 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.

[17] Dean G. Pruitt. *Negotiation Behavior*. Academic Press, 1981.

[18] 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.

[19] James M. Skinner. *A Synergistic Approach to Reasoning*. PhD thesis, Dept. of Computer Science, University of New Mexico, Albuquerque, NM, May 1992.

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

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

[22] 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.

[23] Frank vonMartial. *Coordinating Plans of Autonomous Agents via Relationship Resolution and Communication*. PhD thesis, Universität des Saarlandes, 1991.

[24] Keith J. Werkman. *Multiagent Cooperative Problem-Solving through Negotiation and Sharing of Perspectives*. PhD thesis, Lehigh University, May 1990.