Learning Experiments in a Heterogeneous Multi-agent System

M V NagendraPrasad and Victor Lesser
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
{nagendra,lesser}@cs.umass.edu

Susan E. Lander
Blackboard Technology Group
401 Main Street
Amherst, Massachusetts 01002
lander@cs.umass.edu

Abstract

Self-organization for efficient distributed search control has received much attention previously but the work presented in this paper represents one of the few attempts at demonstrating its viability and utility in an agent-based system involving complex interactions within the agent set. We discuss experiments with a heterogeneous multi-agent parametric design system called L-TEAM where machine learning techniques endow the agents with capabilities to learn their organizational roles in negotiated search and to learn meta-level knowledge about the composite search spaces. We tested the system on a steam condenser design domain and empirically demonstrated its usefulness.

1 Introduction

In this paper, we present a heterogeneous multi-agent parametric design system called L-TEAM where machine learning techniques endow the agents with capabilities to learn their organizational roles in negotiated search and to learn meta-level knowledge about the composite search spaces. L-TEAM is an extension of the TEAM framework[Lander, 1994] for cooperative search among a set of heterogeneous reusable agents. A reusable-agent system is an open system assembled by minimal customized integration of a dynamically selected subset from a catalogue of existing agents. Each agent works on a specific part of the overall problem. The agents work toward achieving a set of local solutions to different parts of the problem that are mutually consistent and satisfy, as far as possible, the global considerations related to the overall problem. As a part of this search process, agents augment their local view of the composite search space with meta-level information about search spaces of other agents through negotiation to minimize the likelihood of generating conflicting solutions[Lander, 1994].

TEAM was introduced in the context of parametric design in multi-agent systems. Each of the agents has its own local state information, a local database with static and dynamic constraints on its design components and a local agenda of potential actions. The search is performed over a space of partial designs. It is initiated by placing a problem specification in a centralized shared memory that also acts as a repository for the emerging composite solutions (i.e. partial solutions) and is visible to all the agents. Any design component produced by an agent is placed in the centralized repository. Some of the agents initiate base proposals based on the problem specifications and their own internal constraints and local state. Other agents in turn extend and critique these proposals to form complete designs. An agent may detect conflicts during this process and communicate feedback to the relevant agents; consequently affecting their further search by either pruning or reordering the expansion of certain paths. The evolution of a composite solution in TEAM can be viewed as application of a series of negotiated-search operators. For a composite solution in a given state, an agent can apply a set of negotiated-search operators represented by the set of arcs leaving that state. An agent can be working on several composite solutions concurrently.

Thus an agent is faced with a variety of control issues. We present our experimental studies involving two of the more important ones in this paper:

1. At a given time, an agent is faced with the problem of choosing an operator from a set of allowed operators that can be applied to one of the composite solutions that can potentially be worked upon. Agents can learn this selection process.
2. Distributed search process in TEAM involves an initial "discovery" phase where agents run into a series conflicts followed by generation of feedback to the other agents involved in the conflict. This leads to agents gaining an enhanced view of the composite search space. Learning can potentially substitute such feedback communication by providing the agents with the learned view of the composite search space derived from similar past problem solving experiences.

The rest of the paper is organized as follows. Section 2 discusses organizational roles in TEAM and presents our experimental results on learning organizational knowledge. Section 3 discusses meta-knowledge about the composite search space and presents our learning algorithm and experimental results. We conclude by discussing the implications of this work.

2 TEAM: A Heterogeneous Multi-agent System

2.1 Learning Organizational Roles in TEAM

In multi-agent systems like TEAM, the search spaces are complex due to interactions between the distributed subspaces. In such complex search spaces, there is a need for organizing the search in such a manner as to choose those actions that lead to generation of helpful constraints for the subsequent searches for solving related subproblems. Organizational knowledge can be described as a specification of the way the overall search should be organized in terms of which agents play what roles in the search process and communicate what information, when and to whom.

Each agent in TEAM plays some organizational role in distributed search. A role is a task or a set of tasks to be performed in the context of a single solution. A pattern of activation of roles in an agent set is a role assignment. All agents need not play all organizational roles; which in turn implies that agents can differ in the kinds of search operators they are allotted. Organizational roles played by the agents are important for the efficiency of a search process and the quality of the final solutions produced.

During each cycle of operator application in TEAM, each agent in turn has to decide on the role it can play next, based on the available partial designs. An agent can choose to initiate a new design or extend an already existing partial design or critique an existing design. The agent needs to decide on the best role to assume next and accordingly construct a design component[1]. It can be an extremely difficult task for a system designer to construct a generic rating function for each agent that takes into account the specifics of the agent set and the complexities of the search controlling the design of a specific artifact. In this paper we propose learning methods that let the agents construct these rating functions based on past problem solving experience.

2.2 Learning Algorithm

The formal basis for learning search strategies adopted in this paper is derived from the UPC formalism for search control (see [Whitehair & Lesser, 1993]) that relies on the calculation and use of the **U**tility, **P**robability and **C**ost (UPC) values associated with each $\langle state, op, final\ state\rangle$ tuple. The Utility component represents the present state's estimate of the final state's expected value or utility if we apply operator *op* in the present state. Probability represents the expected uncertainty associated with the ability to reach the final state from the present state, given that we apply operator *op*. Cost represents the expected computational cost of reaching the final state. Additionally, in complex search spaces, for which the UPC formalism was developed, an application of an operator to a state does more than expand it. The operator application may result in an increase in the problem solver's understanding of the interrelationships among states. In these situations, an operator that looks like a poor choice from the perspective of a local control policy may actually be a good choice from a more global perspective due to some increased information it makes available to the problem solver. This property of an operator is referred to as its *potential* and it needs to be taken into account while rating the operator. An evaluation function defines the objective strategy of the problem solving system based on the UPC components of an operator and its potential. For example, a system may want to reach any final state as quickly as possible with high quality solutions or it may want maximum utility per unit cost. The evaluation function is applied to all the operators applicable to any frontier states of the on-going search and an operator that maximizes the ratings of all the applicable operators is selected.

Starting from this core of UPC formalism, we modify it to suit our purpose of learning organizational roles in negotiated search in multi-agent systems. Our first modification involves classification of all possible states of a search into a pre-enumerated finite class of situations. These classes of situations represent abstractions of the state of a search. Thus, for each agent, there is UPC vector per situation per operator leading to a final state. A situation in L-TEAM is represented by a feature vector whose values determine the class of a state of the search. In L-TEAM, an agent responsible for decision making at the node retrieves the UPC values based on the situation vector for all the roles that are applicable in current state. Depending on the objective function to be maximized, these UPC vectors are used to choose a role to be performed next.

Let $\{S^k_j\}$, $1 \leq \mathrm{j} \leq M_k$, be the set of possible situation vectors for Agent k where each situation vector is a permutation of the possible values for the situation-vector features and let op^k_i, $1 \leq \mathrm{i} \leq N_k$, be the operators Agent k possesses. Agent k has $M_k * N_k$ UPC vectors $\{op^k_i, S^k_j, Agent\ k, U^k_{ij}, P^k_{ij}, C^k_{ij}, Pot^k_{ij}\}$. Given a situation S^k_b, objective function $f(U, P, C, Pot)$ is used to select an operator op^k_a such that

During learning

$$Prob(op^k_a) = \frac{f(U^k_{ab}, P^k_{ab}, C^k_{ab}, Pot^k_{ab})}{\sum_i f(U^k_{ib}, P^k_{ib}, C^k_{ib}, Pot^k_{ib})}$$

After learning

$$op^k_a = {}^k f^{-1}_{op} \max_i f(U^k_{ib}, P^k_{ib}, C^k_{ib}, Pot^k_{ib})$$

where $1 \leq \mathrm{i} \leq \mathrm{N}$, and ${}^k f^{-1}_{op}(rating)$ represents the operator whose UPC values are such that $f(U, P, C, Pot)$ = rating.

Let $\mathcal{T}$ be the search tree where each node is annotated with the triple $\{op^k_i, S^k_j, A_k\}$ representing the application of operator op^k_i in situation S^k_j by Agent k. Let

[1] Organizational role is also referred to as organizational role operator or simply operator in this paper but the reader should note that this is different from the operator used in regular literature. An agent in a particular role can perform certain operations. In TEAM, an initiating agent or an extending agent performs certain set of operations to form a design component.

$\mathcal{F}(\mathcal{T})$ be the set of states on the path to the terminal state T. A terminal state is a state that is not expanded further due to detection of a success or a failure. A final state is a terminal state where the search ends successfully with a mutually acceptable design. When the search enters a terminal state, the performance measures are back-propagated to the relevant agents. We use the *supervised-learning approach* to prediction learning (see [Sutton, 1988]) to learn estimates for the UPC vectors for each of the states.

Let ${}_{(p)}U^k_{ij}$ represent the predicted utility of the final solution achieved by Agent k using an operator i in a state n that can be classified as situation j, accumulated after p problem solving instances and $\mathcal{F}(\mathcal{T})$ be the set of states on the path to a final state F. Let U_F be the utility of the solution and let $0 \leq \alpha \leq 1$ be the learning rate. Then:

$$\begin{aligned} {}_{(p+1)}U^k_{ij} &= {}_{(p)}U^k_{ij} + \alpha\ (U_F - {}_{(p)}U^k_{ij}), \\ & \qquad n \in \mathcal{F}(\mathcal{T}), \quad state \ \ n \ \in situation \ j \end{aligned}$$

Probability value modifications are defined similarly.

$$\begin{aligned} {}_{(p+1)}P^k_{ij} &= (1-\alpha)_{(p)}P^k_{ij} + \alpha O_T, \\ & \qquad n \in \mathcal{F}(\mathcal{T}), \quad state \ \ n \ \in situation \ j \end{aligned}$$

We will not dwell on the details of the Cost component update rule because the evaluation functions used in this work do not involve cost. In a design problem solving system, the computational costs are not a primary consideration. Successfully completing a good design takes precedence over computational costs involved as long as the costs are not widely disparate.

Obtaining measures of potential is a more involved process and requires a certain understanding of the system - at least to the extent of knowing which are the activities that can potentially make positive or negative contribution to progress of the problem solving process. For example, in L-TEAM, earlier on in a problem solving episode, the agents apply operators that lead to infeasible solutions due to conflicts in their requirements. However, this process of running into a conflict leads to certain important consequences like exchange constraints that were violated. The constraints an agent receives from other agents aid that agent's subsequent search in that episode by letting it relate its local solution requirements to more global requirements. Hence, the operators leading to conflicts followed by information exchange are rewarded by potential. Learning algorithms similar to that for utility can be used for learning the potential of an operator. Let ${}_{(p)}Pot^k_{ij}$ represent the predicted potential of the terminal state achieved by Agent k using an operator i in a state n that can be classified as situation j, accumulated after p problem solving instances. Let $\mathcal{F}(\mathcal{T})$ be the set of states on the path to the terminal state T, $Pot_T \in \{0, 1\}$ be the potential arising from the state T, where $Pot_T = 1$ if there is is a conflict followed by information exchange else $Pot_T = 0$. Let $0 \leq \alpha \leq 1$ be the learning rate. Then:

$$\begin{aligned} {}_{(p+1)}Pot^k_{ij} &= {}_{(p)}Pot^k_{ij} + \alpha\ (Pot_T - {}_{(p)}Pot^k_{ij}), \\ & \qquad n \in \mathcal{F}(\mathcal{T}), \quad state \ \ n \ \in situation \ j \end{aligned}$$

2.3 Experiments

To demonstrate the effectiveness of the mechanisms in L-TEAM and compare them to those in TEAM, we used the same domain as in [Lander, 1994] — parametric design of steam condensers. The prototype multi-agent system for this domain, built on top of the TEAM framework, consists of seven agents: pump-agent, heat-exchanger-agent, motor-agent, vbelt-agent, shaft-agent, platform-agent, and frequency-critic. The problem solving process starts by placing a problem specification on a central blackboard (BB). Problem specification consists of three parameters — required capacity, platform side length, and maximum platform deflection. During each cycle, each of the agents in L-TEAM can decide either to *initiate* a design based on the problem specification or *extend* a partial design on BB or to *critique* a partial design on BB. During the process of extending or critiquing a design, an agent can detect conflicts and communicate the cause of the conflict to other agents. The receiving agents *assimilate* the information and use it to constrain future searches.

Each agent has an assigned organizational role in any single design. In this paper, we confine ourselves to learning the appropriate application of two operators in the agents - initiate-design and extend-design. Four of the seven agents — pump-agent, motor-agent, heat-exchanger-agent, and vbelt-agent — are learning either to initiate a design or to extend an existing partial design in each situation. The other three agents have fixed organizational roles — platform and shaft agents always extend and frequency-critic always critiques.

In the experiments reported below, the situation vector for each agent had three components. The first component represented changes in the global views of any of the agents in the system. If any of the agents receives any new external constraints from other agents in the past m time units (m is set to 4 in the experiments), this component is '1' for all agents. Otherwise it is '0'. If any of the agents has relaxed its local quality requirements in the past n time units (n = 2) then the second component is '1' for all agents. Otherwise it is '0'. Typically, a problem solving episode in L-TEAM starts with an initial phase of exchange of all the communicable information involved in conflicts and then enters a phase where the search is more informed and all the information that leads to conflicts and can be communicated has already been exchanged. During the initial phase of conflict detection and exchange of information, the third component is '0'. In the latter phase, it is '1'. We used the following objective evaluation function:

$$f(U, P, C, potential) \quad = \quad U * P + potential$$

We trained L-TEAM on 150 randomly generated design requirements and then tested L-TEAM and TEAM pairwise on 100 randomly generated design requirements different from those used for training. TEAM was setup so that heat-exchanger and pump agents could either initiate a design or extend a design whereas v-belt, shaft and platform agents could only extend a design. In TEAM, an agent initiates a design only if there are no partial designs on the blackboard that it can extend. We looked at two parameters of system performance. The primary parameter was the cost of the best design produced (lowest cost). The other parameter was the number of cycles the system went through to produce the best cost design. In TEAM (and L-TEAM) each agent in turn, gets a chance to perform an operation during a cycle. The number of cycles represents a good approximation to the amount of search performed by the entire system.

Average cost of a design produced by L-TEAM was 5587.6 and by TEAM was 5770.6. In the design domain, this difference of 3.2 % is considered a big win, especially because these designs may be mass-produced. Wilcoxon matched-pair signed-ranks test revealed that the cost of designs produced by L-TEAM was lower than those produced by TEAM at significance level 0.05 (p value was 0.00000003). The average number of cycles needed for L-TEAM to produce a design was 13.89, while TEAM needed 13.01 cycles.

3 Learning Meta-knowledge about Composite Search Spaces

3.1 Composite Search Spaces

The search space in a multi-agent system like TEAM can be viewed as consisting of two components: the *local* space of each individual agent and the *composite* space of the system. A local space is private to an agent whereas the composite space is shared by all agents. An agent defines a local solution space by assigning values to the parameters in its local solutions. The local search space is defined by the parameters the agent uses to constrain its local search. Problem solving in TEAM starts with agents possessing only local views of the search and solution spaces. This is highly unlikely to lead to mutually acceptable solutions lying in the composite space. Agents engage in a failure-driven exchange of feedback on non-local requirements to develop the closest approximation possible to the actual composite search space. With their improved view of the global situation, the agents are more effective at developing globally acceptable solutions.

The parameters defining a local solution space may be constrained by either *implicit* constraints or *explicit* constraints. Implicit constraints represent procedurally captured or embedded requirements. Explicit constraints are declaratively represented requirements that can be shared by agents to enhance the effectiveness of the search for mutually acceptable solutions. In TEAM, explicit constraints are limited to simple boundary constraints of the form $(x < n)$, $(x \leq n)$, $(x > n)$, or $(x \geq n)$. If x is a shared parameter, then an explicit constraint on x can be shared with other agents. An agent that includes the parameter x in the definition of its local search space can assimilate a constraint on x that it receives as feedback from another agent. This process is called information assimilation and it enhances an agent's view of the composite search space. For example, Figure 1 shows a two agent system with their local views and the composite search space. Initially agents start out with just local views of their search spaces. Proposal of infeasible solutions leads to conflicts and subsequent exchange of constraints if those conflicts are due to violations of explicit constraints. In the simple two agent, two shared parameter system in our example, after all the explicit constraints are exchanged through iterative process of conflict detection and explicit constraint feedback, each agent's approximation of the other agent's solution space on the shared parameters is shown in the lower part of Figure 1.

The search in TEAM is cast as a constraint-optimization problem meaning that not all constraints need to be satisfied in a solution. As many constraints are satisfied as is possible. Constraints have differing amounts of flexibility. Some may be hard, meaning that they must be satisfied in any legal solution. Some others may be soft constraints that may be relaxed as and when needed. Softness of a constraint represents its degree of flexibility with some constraints being softer than others. The composite solution space lies within the intersection of the local parameter spaces under hard constraints but not necessarily under soft constraints. Order of constraint relaxation is an issue out of the scope of this paper. Lander[Lander, 1994] presents an algorithm for this.

Given a problem specification, each agent in TEAM initiates a search with completely local views. Whenever conflicts on explicit constraints are detected, a feedback process conveys the conflict information and the corresponding constraints to the agents involved. Agents now do their subsequent searches with an enhanced view of the composite solution space. Thus, conceptually, search in TEAM can be divided into two phases: 1) an initial phase where conflict detection and exchange of explicit conflicts occurs and 2) a latter phase phase where agents have assimilated feedback information, formed approximations of the composite search space, and no further useful interaction is likely to occur[2].

3.2 Learning Algorithm

We now present our experiments in endowing the agents with capabilities to learn the characteristics of the composite search space. The algorithm we use here is the Instance-Based Learning(IBL) algorithm[Aha, Kibler, & Albert, 1991]. During the learning phase, the agents perform their search with information assimilation as discussed above. At the end of each search, an agent stores the problem requirements and the non-local constraints it received as feedback from the other agents as an approximation of the non-local requirements on the composite solution space. After the agents learn over a sufficiently large training set, they can replace the information assimilation process with the learned knowledge. When a new problem instance is presented to the agent set, it chooses the set of non-local constraints that are stored under the problem specification that is closest to

[2] An agent's view of the composite search space is approximate even after the exchange of all the conflicting explicit constraints because of the existence of implicit constraints on shared parameters. These types of constraints cannot be communicated by an agent to another agent.

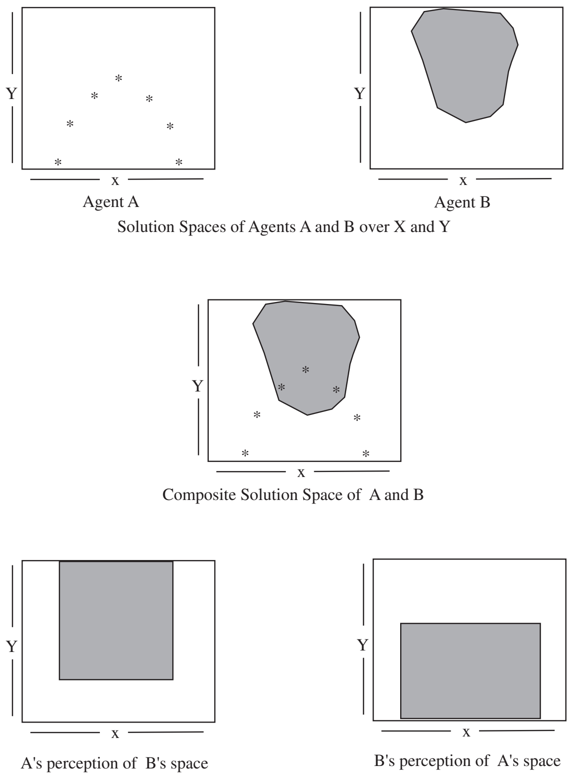

Figure 1: Local and Composite Search Spaces

the present problem specification and adds them to the set of local requirements at the start of the search.

3.3 Experiments

L-TEAM is the TEAM system described in the previous experiments but augmented with the IBL capabilities[3]. We trained the system with 100 randomly chosen instances and then tested the system on 100 instances different from the training instances. During the testing phase, L-TEAM's information-assimilation mechanism was disabled. Thus the agents had to rely on their learned knowledge of the approximations of the composite solution space to produce good solutions. However, if the similarity of the present problem specification to the most similar past problem solving instance is not below a threshold then using the constraints of this past problem solving instance may be misleading and hence the present problem instance does not add these constraints to its set of local requirements. In such a situation, L-TEAM behaves like TEAM with no information assimilation. For comparison purposes we also ran the test cases against TEAM with no information assimilation capabilities and TEAM with information assimilation.

Average cost of a design produced by TEAM with information assimilation was 6598.0, by TEAM without information assimilation was 7227.2 and by L-TEAM after learning was 6961.26. Average cycles per design taken by TEAM with information assimilation was 12.98, by TEAM without information assimilation was 15.54 and by L-TEAM was 13.9. Wilcoxon matched-pair signed-ranks test revealed that the cost of designs produced by L-TEAM after learning was lower than those produced by TEAM with no information assimilation at significance level 0.05 (p value was 0.000056). However, Wilcoxon matched-pair signed-ranks test also revealed that the cost of designs produced by TEAM with information assimilation was lower than those produced by L-TEAM at significance level 0.05 (p value was 0.00014).

We next ran an experiment where L-TEAM used learned knowledge when it found a past problem solving instance similar enough to the present problem specification. If it failed to find such an instance, it used negotiated search with information assimilation. The same 100 test instances as in the previous experiments were used.

Average cost of design produced by L-TEAM with information assimilation was 6680.02 and it took 12.94 cycles on an average. Wilcoxon matched-pair signed-ranks test revealed no significant difference in cost of design produced by L-TEAM with information assimilation and TEAM with information assimilation. However, L-TEAM with information assimilation produced designs that on an average cost 82.00 units more than those produced by TEAM with information assimilation. L-TEAM took 0.04 cycles less than TEAM on an average.

These experiments suggest that learning could substitute for negotiation in situations where negotiation becomes very expensive due to communication costs

[3]Note that the L-TEAM system has either role learning or IBL active but not both. In future, we intend to turn on both the mechanisms and look for interactions between them.

(however, negotiation is still needed during the learning phase). The results achieved by using learned knowledge of the composite solution space are better than having no negotiation at all. When negotiation is permitted, it still outperforms using learned knowledge. Note however that, given the limited number of experiments conducted till now, we are not claiming this to be true across all multi-agent negotiated-search systems. In fact, we believe that the power of well-tailored learning mechanisms can endow a multi-agent system with capabilities that far transcend those produced in the work presented here.

4 Implications and Conclusion

Previous work in self-organization for efficient distributed search control has, for the most part, involved simple agents with simple interaction patterns and concentrated primarily on toy domains. The work presented in this paper represents one of the few attempts at demonstrating the viability and utility of self-organization in an agent-based system involving complex interactions within the agent set.

Acknowledgments

This work is supported in part by the National Science Foundation Cooperative Agreement Grant No. EEC 9209623. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

[Aha, Kibler, & Albert, 1991] Aha, D. W.; Kibler, D.; and Albert, M. K. 1991. Instance-based learning algorithms. *Machine Learning* 6:37–66.

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

[Sutton, 1988] Sutton, R. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–44.

[Whitehair & Lesser, 1993] Whitehair, R., and Lesser, V. R. 1993. A framework for the analysis of sophisticated control in interpretation systems. Computer Science Technical Report 93–53, University of Massachusetts.