

A Distributed Planning Approach Using Multiagent Goal Transformations

Michael T. Cox, Mohammad M. Elahi, and Kevin M. Cleereman

Department of Computer Science and Engineering

College of Engineering & Computer Science

Wright State University

Dayton, OH 45435-0001

{mcox, melahi, kcleerem}@cs.wright.edu

Abstract

A multiagent goal transformation is defined as a directed alteration of goals by the agents in a multiagent system in order to adjust for lack of resources and/or lack of capabilities. Multiagent goal transformation can be used as a uniform coordination mechanism. We describe preliminary research that implements a distributed planning approach incorporating several co-operative planning agents and that uses a multiagent goal transformation as a coordination mechanism. Each agent in the system represents a domain with capabilities differing from the domains represented by the other agents. The distributed coordination mechanism allows distribution of tasks among agents and concatenation of resultant sub-plans from agents without a central coordination process. Examples are given to illustrate how the system works in the logistics transportation domain.

1 Introduction

The planning tasks in multiagent planning are distributed among several autonomous agents. The distribution process may depend on what is being distributed among the agents: the planning process, the execution of the plan, or the planning problem itself. In a cooperative distributed planning approach, multiple cooperative agents with shared objectives and representations work together to produce an overall plan that can be jointly executed in a coherent and effective manner (Desjardins *et al.*, 1999). In this paper we demonstrate a cooperative distributed planning approach that distributes the planning process among individual agents in a multiagent system.

In building a cooperative distributed planning system, some of the key questions one must address include:

- How is the overall planning problem decomposed and allocated to the agents?
- How are the sub-plans of individual agents concatenated to produce the overall plan that can be executed coherently and effectively?
- How do agents communicate with one another during planning?

We show how a system called COMAS (Edwin and Cox 2001) addresses all of the above issues. We have extended COMAS to utilize a novel approach to create a domain-specific agent topology that best finds a solution for a specific problem in a particular domain. We have used a planning and learning architecture called PRODIGY (Carbonell *et al.*, 1992; Veloso *et al.*, 1995) as the underlying framework, and we have used Prodigy/Agent (Cox *et al.*, 2001) as individual agents. The paper is organized as follows: Section 2 defines the concept of multiagent goal transformation used in our approach. Section 3 describes the generalized approach with different issues involved. Section 4 continues with an example that illustrates the approach. Section 5 discusses the experimental results. Section 6 focuses on some future works still to be done. Finally, section 7 has concluding remarks.

2 Multiagent Goal Transformations

Cox and Veloso (1998) defined a goal transformation as a minimal movement in a goal hyper-space that transforms a given goal into another goal that is similar to the first. The concept is used in dynamic environments where a goal may become obsolete due to the changing circumstances, and the planner needs to alter its goal to another goal that is achievable.

The goal transformation concept also provides a mechanism to coordinate agent behavior in multiagent planning. If an agent cannot achieve a goal by itself, it can transfer the goal into one that is achievable through co-operation. A multiagent goal transformation is defined in (Cox *et al.*, 2001) as follows: to solve a goal G , solve instead a goal G' that generates a sub-solution, and then pass the remainder of the goal (i.e., G minus G') to another agent. To illustrate the utility of this type of transformation, they defined a rule for implementing multiagent goal transformations with a domain-dependent process. This rule only works for a single class of problems in the domain.

To generalize the concept of a multiagent goal transformation in a domain-independent manner, we refine the definition. We distinguish between dynamic preconditions that may be adopted as subgoals because operators exist that modify the state, and applicability conditions (i.e., static preconditions) for which no operator of any agent exists to change the state. For example no operator may exist to change the weather in a particular domain, yet some operators may require good weather for the operator to be applicable.

To achieve a particular goal state G , an agent must have an operator that can achieve the state. Now all open dynamic preconditions (those whose state is not true) are split into two sets. Set one contains states for which the agent has an operator, and set two contains states for which it does not. The agent solves those preconditions in set one, and then it requests other agents to solve each state of set two. For example, a planner of a transportation domain might have two dynamic preconditions for the operator representing the action "Load Airplane P with object O at location L ." One precondition requires that the object O be at the location L and the second precondition requires that the airplane P be at the same location. Suppose an agent A only has an operator to move objects but does not have an operator to fly airplanes. In this example, A will solve the first condition, and ask another agent to solve the second.

3 The Approach

Multiagent goal transformations can be used as a uniform coordination mechanism among agents in a distributed multiagent planning system when different agents cooperate by sharing different capabilities. We divide the capabilities by splitting a domain into sub-domains and assigning them to different agents in the system. Then we encode the coordination mechanism in the domain of each agent so as to enable the agents to communicate with each other. In the following subsections, we describe the approach with generalized algorithms and other key tasks involved in distributed planning.

3.1 Problem Decomposition and Task Allocation

Problem decomposition and task allocation are performed by the agents themselves. The only necessary task is the reasoning by the agents to identify the part of the problem that a given agent can independently achieve and the remaining part the given agent cannot independently achieve. This process continues until an agent receives a sub-problem that the agent can solve on its own.

Problem decomposition is achieved by splitting a domain into sub-domains that are assigned to the agents in the system. Task allocation is achieved by encoding the agent communication mechanism in the domain file of each agent. A domain can be split into any number of sub-domains bounded by the total number of operators in the domain. Each sub-domain contains not only those operators assigned to it, but also a set of phantom operators that are not assigned to. A phantom points to the agent(s) to whom the operator is assigned. In this way, communication between agents is encoded in every sub-domain — if an agent does not own an operator, it knows which agent to enlist for that operator by checking its set of phantoms.

3.1.1 Algorithm for splitting a domain

Suppose D is a domain with n operators. To split D into m sub-domains one would employ the following steps:

1. Arbitrarily divide the operators from D into $m \leq n$ groups. Assign each group of operators to a separate sub-domain, $D_{1 \leq i \leq m}$.
2. For each operator owned by a sub-domain, make no changes to the operator. For each remaining operator in the original domain, encode the request to the agent that owns the operator, and place all such paired operators in the subdomain.

3.1.2 Algorithm for agent communication

Suppose a domain D is split into n sub-domains D_1, D_2, \dots, D_m , and assigned to n agents A_1, A_2, \dots, A_m , respectively, in a multiagent system. If any agent A_i with Domain D_i in the system receives a request to achieve goal G , then A_i has two choices:

- If it has the capability to achieve the goal G by itself, then A_i will generate the plan and return the plan to the requesting agent.
- If it cannot achieve the goal G by itself, then it will achieve the part of the goal that it *can* complete and will enlist other agents to achieve the goal remainder.

In the first case, the system does not need multiagent goal transformations. In the second case, the steps are as follows:

1. The agent A_i starts planning using its own operators defined in D_i . If A_i requires an operator not owned by D_i in order to achieve a goal G , A_i must find and request that another agent A_j ($1 \leq j \leq n$ and $i \neq j$) achieve G .
2. After receiving the request from A_i , A_j will follow the same procedure as in step 1. This process will continue until an agent can achieve the goal of the received request by itself. This final agent will then send the generated plan back to the requesting agent.
3. After receiving the sub-plan, the requesting agent will merge the received sub-plan with its own sub-plan, and will send the resultant plan back to the agent from which it has received the request. This process continues until the initiating agent gets the overall plan for achieving goal G .

To illustrate the algorithm, let us assume that there exists a multiagent system with 2 agents for a simple blocksworld domain: one pickup agent that can pickup objects from the table and one putdown agent that can put down objects onto the table. The original domain is divided into two sub-domains: the pickup-domain and the put-down domain. Suppose that the pickup agent has received a request to transfer an object $Obj1$ from $table1$ to $table2$. The pickup agent picks up $Obj1$ from $table1$, and requests the putdown agent to put $Obj1$ on $table2$. The putdown agent achieves the goal, and returns the subplan: (putdown $Obj1$ $table2$). The pickup agent receives the subplan and concatenates it with its own plan: (pickup $Obj1$ $table1$) to produce the complete plan. The resultant plan is as follows.

((pickup $Obj1$ $table1$) (putdown $Obj1$ $table2$))

3.2 Concatenating or Merging of Sub-plans

An agent can receive any number of sub-plans from other agents in the system. The agent stores the received sub-plans from other agents as lists. After completing the planning process, the agent will insert sub-plans into the points where requests to other agents were made.

To illustrate the generalized procedure, assume:

- There are two agents: A and B .
- Agent A may request agent B as many times as necessary to solve sub-problems.
- Agent B does not need to make requests of Agent A .

To merge the received sub-plans from agent B into its own plan, agent A does the following steps:

1. Each time agent A requests a sub-plan from agent B , it annotates the current node in the search tree with the received result.

2. After generating its own plan, agent A parses the plan for each step. The sub-plans stored in the search nodes in step 1 correspond to steps in the plan.
3. Agent A checks whether the operator node information in the current plan step contains an appended sub-plan. If it does, then A inserts the sub-plan into the current position of the overall plan. Otherwise, A inserts the current plan step into the overall plan.
4. Steps 2 - 3 are repeated until agent A 's entire plan is parsed.

3.3 The Coordination Mechanism

Coordination can be of two types:

- Communication of the information of state changes occurred by actions of other agents.
- Coordination of sub-plans among different agents.

The first type of coordination ensures that agents are aware of state changes caused by other agents during the planning process. An earlier version of COMAS implemented this type of coordination. Whenever an agent does some action, the change of state is broadcast to all other agents in the system. Other agents receive the state changes when they sense the world. This interaction between agents (called goal interaction coordination in COMAS) can be of two types: positive goal interaction, where action by one agent helps another agent to achieve a desired state; and negative goal interaction, where action by one agent adversely affect the planning of another agent.

The second type of coordination is accomplished by multiagent goal transformations (section 2) that we have used in the system. Our approach employs a master-slave coordination process. Each requesting agent acts as a master and treats all requested agents as slaves.

4 The Logistics Domain – An Example

The logistics transportation domain (Veloso 1994) is a well-known example domain in multiagent planning research. The basic elements of the domain are the infrastructure, the transportation units, and the packages. The infrastructure consists of a set of cities, and each city consists of a number of locations (e.g., post offices and airports). Trucks and airplanes are the transportation units: trucks can move freely within a city, but transport between cities can only be achieved by airplanes. The domain involves the movement of packages within cities by truck and between cities by airplane. For our example, goals consist of having packages at destination locations.

The logistics domain can be split into several sub-domains to fit in our distributed planning approach. There are six

operators in the logistics domain: LOAD-TRUCK, UNLOAD-TRUCK, LOAD-AIRPLANE, UNLOAD-AIRPLANE, DRIVE-TRUCK, and FLY-AIRPLANE. To split the domain the operators are assigned arbitrarily to the agents in COMAS. If a goal requires operators from different subdomains, multiple planning agents must cooperate. Suppose an agent handles the moving of packages by truck within a city, and another agent handles the moving of packages by airplane between cities. If the goal is to transfer a package from the post-office in city1 to the airport in city2, then neither agent can achieve the goal on its own. The first agent must move the package to the airport in city1 and find a second agent to move it from there to the airport in city2. This intuitive explanation is implemented as a multiagent goal transformation.

One of the factors that need to be considered when splitting a domain is a goal loop. A goal loop occurs in a planning problem when a precondition of a goal produces a subgoal that is already pending. Given a goal G that can be achieved by an operator OP1 with a precondition P1, if P1 produces a subgoal G' that requires an operator OP2 having the precondition P2 = G, then the resulting situation is a goal loop, because the achievement of G is still pending. PRODIGY can detect the goal loop if it exists. Goal loops in COMAS occur when any two operators placed in two different domains may conflict. In the logistics domain, the unload-load operator pairs can produce goal loops — if they are placed in different subdomains when requests are made, goal loops may be produced and plans may fail. Goal loops in COMAS can be avoided by placing these operator pairs in the same domain file.

To request another agent to achieve a goal, an agent needs to define operators that do not belong to the agent's own domain in a special format as discussed in section 3.1 to implement the transformation. Suppose Agent-1 owns the LOAD-AIRPLANE operator. Agent-2 knows that Agent-1 is the owner of the LOAD-AIRPLANE operator. Figure 1 shows the operator LOAD-AIRPLANE that loads a package inside an airplane and is defined in the domain file of Agent-1.

```
OPERATOR LOAD-AIRPLANE
<obj>:      type OBJECT
<airplane>: type AIRPLANE
<loc>: (and type AIRPORT
          (in-obj-city-p <obj> <loc>))
Pre: (and (at-obj <obj> <loc>)
          (at-airplane <airplane> <loc>))
Del: (at-obj <obj> <loc>)
Add: (inside-airplane <obj> <airplane>)
```

Figure 1: LOAD-AIRPLANE operator

Agent-2 has the same definition in its domain file for the above operator except that it does not perform deletions does not have the constraints on <loc>. Instead, the definition contains a request to Agent-1 encoded as a meta-predicate in the operator variable list (i.e., (<var-1>, <var-2>, .. , <var-k>)). The meta-predicate function knows which agent to request for achieving the goal. The definition states that we can infer the goal to have the package bound to the variable <obj> inside the airplane bound to <airplane> if the request to Agent-1 is successful.

5 Experimental Results

An experiment was conducted to illustrate the difference in planning performance within systems with different agent sizes. We have split the logistics domain into a number of sub-domains based on the number of agents in COMAS. For the experiment, we have used 1, 2, and 3-agents in COMAS. Operators are split arbitrarily into different sub-domains assigned to the agents. For the two-agent system, each agent owns 3 operators. Operator assignments are done with and without the inclusion of goal loops.

In this experiment, we manipulate the number of agents in the COMAS and the depth bound during planning, using a total of 25 random planning problems. We evaluated planning performance with and without the inclusion of goal loops and with different depth bound.

The coordination mechanism employs multiprocessing to enable an agent to take any number of requests simultaneously. Also, implementing goal interaction coordination enables agents to coordinate actions so that other agents can update their world state according to the change. We have used Knowledge Query Manipulation Language (KQML) (Finin, McKay, & Fritzson 1992) as the coordination protocol between agents.

For simplicity, we have used a blocking mechanism for communication between agents in our system instead of asynchronous communication. An agent blocks after sending a request to another agent until it gets a subplan. Although we did not use true parallelism in our experiment, it is sufficient to show the efficiency of the approach.

We calculated the average number of nodes expanded per agent as the performance measure for each system. This was calculated only for the problems for which the system generated a solution. The smaller the number of nodes expanded, the better the performance.

Figure 2 shows the planning performance of different systems in terms of the average number of nodes expanded per agent without goal loops. All of the agents successfully solved all 25 problems. It is obvious from the graph that increasing the number of agents in the system resulted in

improved performance. This is because splitting the original domain into several sub-domains produces a much smaller search space for the underlying PRODIGY planner.

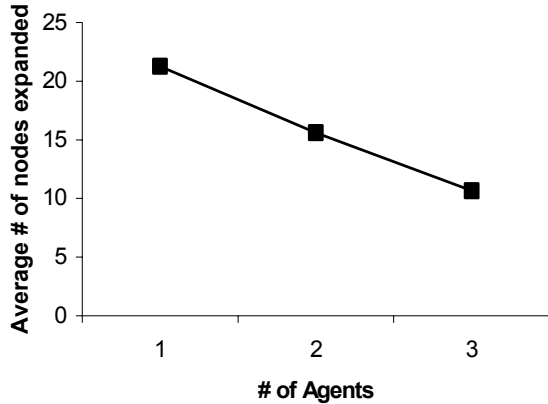


Figure 2: Performance without goal loop as a function of number of agents in the system

Figure 3 shows the same performance measure with the inclusion of goal loops. The overall number of nodes expanded per agent is lower than that of Figure 2. However the number of problems solvable decrease. When a goal loop occurs the problem cannot be solved because only one agent will possess a given operator, and because the agents synchronize with blocking. That is, a deadlock will occur because some agent will wait due to a block while another agent will require the first. In Figure 3, the 1-agent system solved all 25 problems, whereas the 2- and 3-agent systems solved 20 problems each.

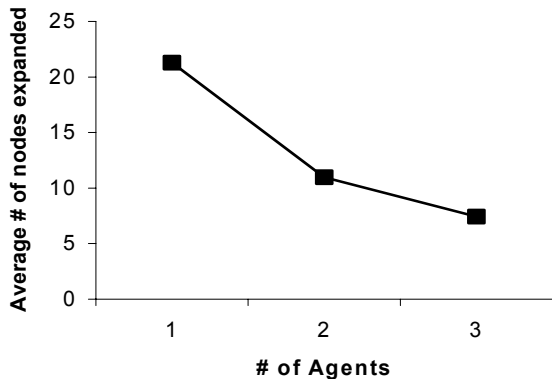


Figure 3: Average number of nodes expanded per agent with goal loop

One of the advantages of splitting a domain in COMAS is the capability of solving a problem in a much lower depth-bound. Figure 4 illustrates the effect of depth-bound in our approach. As the graph shows, increasing the number of agents in COMAS results in a higher number of solvable problems with the same depth bound value. If the number of agents in COMAS increases, the number of solvable problems increases with a fixed depth-bound value.

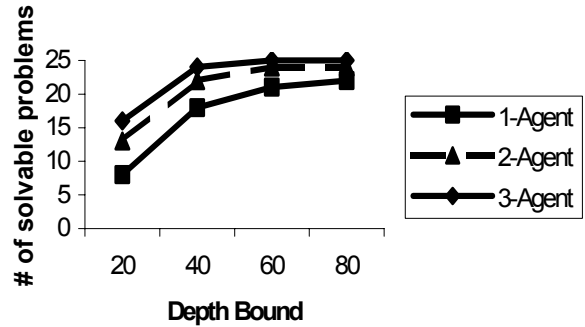


Figure 4: Plan performance as a function of depth bound and number of agents in COMAS

6 Future Works

The experimental results favor this approach, but nevertheless some more work needs to be done on this problem. The approach should engage a generalized procedure for splitting larger domains. This may not be trivial for all domains — there may be some hidden assumptions that have to be considered when splitting the domain. For example, in the blocksworld domain, the assumption of having a single robot arm will not work. Instead, each agent requires a separate robot arm to PICKUP an object.

We are currently working with a static network configuration in our multiagent framework. What if there is no agent to take the request to solve a particular sub-problem? The system should behave robustly in this case. The configuration of the distributed system should be dynamic. Also, we have used a blocking mechanism where an agent blocks after requesting another agent until it gets the subplan back. This does not reflect the parallelism needed for the efficiency of the overall approach. An asynchronous mechanism where agents can still receive requests or perform certain actions after enlisting another agent will increase the efficiency.

We used up to 3 agents in the system for the experiment. Since there are six operators in the logistics domain, there

can be six agents with one operator each. Using a system with up to six agents will be more useful to demonstrating the trends in the resulting data, though it will be difficult to excise goal loops from such a system.

The results we have shown in this paper are for the logistics domain only. We still need to work on some larger domains, e.g. Extended STRIPS.

7 Conclusions

Multiagent goal transformations can be used as a uniform coordination mechanism to implement distributed multiagent planning. In this paper, we have shown how this approach can be used: first we defined the concept of multiagent goal transformation, next we described the generalized approach of how to use the concept, then we have illustrated the approach with an example in the logistics domain, and finally we have shown our process evaluation in order to illustrate the efficiency and viability of this approach.

Acknowledgements

This paper is supported by the Air Force Office of Scientific Research (AFOSR) under grant number F49620-99-1-0244 and by a grant from the Information Technology Research Institute (ITRI) at Wright State University. We also acknowledge Boris Kerkez for his random problem generator used in our experiment to generate batches of problems.

References

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Perez, A.; Reilly, S.; Veloso, M. M.; and Wang, X. *PRODIGY4.0: The Manual and Tutorial*, Technical Report, Computer Science Department, Carnegie Mellon University, 1992.

Cox, M. T.; Edwin, G.; Balasubramanian, K.; and Elahi, M. M. 2001. Multiagent Goal Transformation and Mixed-Initiative Planning Using Prodigy/Agent. In *Proceedings of the Fifth International Conference on Information, Systems, and Cybernetics*, Florida.

Cox, M. T.; and Veloso, M. M. 1998. Goal Transformations in Continuous Planning. In M. desJardins (Ed.), *Proceedings of the 1998 AAI Fall Symposium on Distributed Continual Planning*, 23-30. Menlo Park, Calif.: AAI Press / The MIT Press.

Desjardins, M. E.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. J. 1999. A survey of research in distributed, continual planning. *AI Magazine*, 4, 13-22.

Edwin, G.; and Cox, M. T. 2001. COMAS: Coordination in MultiAgent Systems. In *Proceedings of the Fifth International Conference on Information, Systems, and Cybernetics*, Florida.

Finin, T.; McKay, D.; and Fritzson, R. 1992. *An Overview of KQML: A Knowledge Query and Manipulation Language*. Univ. of Maryland, CS Dept.

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. Berlin: Springer.

Veloso, M. M., Carbonell, J. Perez, A., Borrajo, D., Fink, E., and Blythe, J. 1995. Integrating planning and learning: The PRODIGY Architecture. *Journal of Theoretical and Experimental Artificial Intelligence* 7.