Submitted to the Fifteenth National Conference on Artificial Intelligence

Metaplanning for Multiple Agents

Jonathan Gratch

University of Southern California, Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA, 90292
gratch@isi.edu

Abstract

We present an extension to classical planning techniques that facilitates their use in complex multi-agent domains. The approach implements a form of metaplanning that enables a planner to reason about properties of multiple plans in a single plan network. With this approach, a planning agent can simultaneously generate an individual plan, repair a second, and, together with a group, execute a third. This provides some of the key functionality of sophisticated multi-agent reasoning techniques, such as Grosz and Kraus' shared plans, but within the context of better understood classical planning techniques. As such, it helps bridge the gap between planning and multi-agent research.

Introduction

Recent applications have illustrated the power of planning techniques in remarkably complex virtual environments such as information gathering (Knoblock, 1995), intelligent tutoring (Rickel and Johnson, 1997) and military simulations (Hill, 1997). In these domains, planning systems don't simply develop plans but also monitor their execution and replan when things go awry. Many of these domains involve multiple agents, posing greater challenges: a planning agent must distinguish between its own activities and those done in coordination with others; even the process of planning could conceivably be shared, allowing parts of a group plan to be constructed through collaboration.

It is increasingly important for a planning system to represent multiple plans in various states of development or execution. For example, in the above military simulation domain, a commander agent may have to generate a team plan while simultaneously executing another plan that moves it into proximity with the rest of the team. Coordinating the relationships between plans and between plans and other agents requires sophisticated reasoning techniques. Indeed, some multi-agent research has proposed building elaborate representations systems around a classical planner to support this type of inference (e.g., shared plans, Grosz and Kraus 1996).

Reasoning about multiple plans is a form of metaplanning. In this paper, we propose a surprisingly simple extension to classical planning techniques that allows the planner itself to cleanly perform this type of reasoning *using a single plan*. Actually, we suggest a reconceptualization of the term "plan". Traditionally, the term plan is synonymous with the network of operators, henceforth called tasks, and constraints maintained by the planner. Instead, we make a distinction between this network, which we call the *task network*, and a plan, which we now define as some subset of the task network. Thus, the network contains multiple plans and we allow different plans to have different status (some are executable, some associated with groups, etc.). Most importantly, some plans manipulate the properties of other plans, and in this way, the planner engages in metaplanning. The advantage of maintaining multiple plans in the same task network is we take full advantage of a planner's ability to reason about interactions between tasks, and therefore between plans in this new sense. For example, the effects of tasks in one plan could support preconditions in a second, and inconsistencies across plans can be detected automatically by a planner's constraint reasoning.

Metaplanning has been explored in the past but our focus differs from that earlier work. Previously, metaplanning was seen as a form of search control (Wilensky 1980, Stefik 1981). Such meta-planners treat plan modifiers (such as promotion or step addition) as steps in a meta-space who's sequence of application is to be determined by planning. A drawback is that meta-plans can become quite detailed and the approach seems more cumbersome than alternate methods of search control, particularly learning techniques (Minton 1990). In contrast, we don't attempt to control the details of how a plan is generated, and instead focus on the relationships that plans have with each other and with other domain objects. Another difference is that while earlier work maintained separate planning levels (essentially separate task networks), we maintain all plans, meta or otherwise, in the same task network.

As we mentioned, multi-agent research also considers the problem of metaplanning, focusing on representing properties of plans to support group planning (e.g., I have a plan to do X, we have a plan to do Y). Researchers in this area have identified key meta-level concepts (e.g., joint commitments, Cohen and Levesque 1990, and shared plans, Grosz and Kraus 1996) necessary to support group planning and execution. A drawback of this research is it tends to focus almost exclusively on representational rather than algorithmic issues, and the representations tend to be quite complex. This makes it difficult to see the relationship between these methods and classical planning research. Our approach supports some

of the key functionality provided by multi-agent formalisms without adopting the full representational complexity of these approaches.

In this paper, we describe our approach to metaplanning, motivated by the issues raised in multi-agent research. We have successfully applied the method to a large-scale application domain of military combat simulations, and we illustrate the method on an example from this domain.

The Planner

We briefly overview of the basic planner before describing how it can be extended to perform metaplanning. While our extension could be incorporated into a variety of planning architectures, some of the details of our impact the example and discussion below.

Our planner incorporates a number of novel features, but for the purposes of this article it may be considered equivalent to IPEM (Ambros-Ingerson and Steel 1988), which shares many features with the more recent XII planner (Golden *et al*. 1994). Some differences between our approach and IPEM are briefly considered later in the article. IPEM was designed to support planning, execution and replanning for environments where actions have duration and the world can change in surprising ways.

IPEM plans by posting constraints to a task network in the same fashion as other classical planners such as SNLP (McAllester and Rosenblitt 1991). Constraints are added in response to flaws in the current network. For example, if a task has an open precondition, the planner attempts to resolve this flaw by identifying an existing task that establishes the effect (*simple-establishment*) or introduce a new task (*step addition*). Both modifications add constraints to the network. Simple-establishment asserts a *protection constraint* that protect the effect from the moment it is created until it is used by the precondition, and binding constraints that ensure the effect unifies with the open-precondition. Step-addition posts a constraint to include the task in the network in addition to the constraints posted by simple-establishment. Unlike SNLP, tasks have duration: they must be explicitly initiated and terminated. Tasks can also be decomposed hierarchically.

Besides the task network, the planner maintains a declarative representation of the perceived state of the world or *current world description* (CWD). The CWD allows the planner to monitor the execution of task and detect any surprising changes in the environment. The planner may only initiate tasks who's preconditions unify with the CWD (and are not preceded by any uninitiated tasks). Similarly, tasks are terminated when all of their effects appear in the CWD. Task initiation and termination may be interleaved with other planning operations. As the CWD reflects the perceived state of the world, it may change in ways not predicted by the current task network. For example, some external process modifying the environment is detected by changes to the CWD not predicted by the current set of executing tasks. These changes may provide opportunities (as when an unsatisfied preconditions now observed in the world). They may also threaten constraints in the plan network, forcing the planner to modify the task network to handle the resulting flaws.

Metaplanning

We extend the planning algorithm in several ways: (1) allow multiple plans to be represented in a single task network (the notion of a plan will be defined below); (2) allow plans to participate in relationships with other domain objects including other plans; (3) allow plans to modify the relationships in which plans participate.

To support these capabilities, we introduce the notion of a *plan designator*, a symbol that denotes the portion of the task corresponding to a plan. Plan designators can be used just as any other constant when defining a domain theory. They may be used as terms in the preconditions and effects of tasks. For example, if *P* is a plan designator, we may define a predicate *FLAWED* over plans and use *FLAWED(P)* as a precondition to some task in our domain theory. The difference between this and a statement such as *ON(A,B)* is that while *ON(A,B)* is typically interpreted as a relation among objects in an external environment, *FLAWED(P)* is to be interpreted as a relation over portions of the planner's task network (i.e. the subnetwork denoted by *P* participates in some flaw).

Plans and Plan designator semantics

A plan corresponds to a subset of the task network and a plan designator is the symbol that denotes this subset. We have a procedural semantics for determining this denotation that is closely tied to the operations a planner can use to make changes to the task network. A task network can be viewed as a set of constraints (task T1 is constrained to be in the plan, the fact F is protected from T3 to T4, etc.). For a given task network, each of these constraints belongs to exactly one plan (i.e., plans form a partition of the set of constraints comprising the task network).[1] Whenever new constraints are inserted into the task network, they are simultaneously added to a particular plan.

Deciding the goals a plan is to solve is necessarily domain-specific. Therefore, initial partial plans are created (along with the corresponding designation) as result of executing domain-specific procedures that can be associated with tasks. This "seed" plan will consists of constraints needed to represent a *goal* task and possibly a partial plan for achieving its preconditions. For example, in the military simulation domain, an agent can be commanded to achieve a goal. The radio message containing the command is translated into a partial plan using one of these domain-specific procedures (see below).

New constraints are added to this "seed" as it is expanded by the planner. Step addition inserts a task into

[1] In our current work we are generalizing the notion of a plan designator to allow a given task to belong to multiple plans (see below).

the network in order to establish some open precondition of another task (the establishee). Any constraints added as a result of this modification become part of the establishee's plan (more precisely, they becomes part of the plan containing the constraint that asserts the establishee). Similarly, simple establishment introduces protection and binding constraints into the network that becomes part of the establishee's plan. Task decomposition, which breaks an abstract task into a partially ordered sequence of subtasks, generates a set of constraints that become part of the abstract task's plan. The planner has three modifications for resolving threats to protection constraints (promotion, demotion, and separation). Constraints added by these modifications are added to the plan which contains the threatened protection constraint.

Meta-relations, Meta-predicates, and Meta-tasks

Now that we have defined plans, the next step is to describe the relationships in which a plan may participate. A *meta-relation* is some relation over plans (and possibly other domain constructs) represented by the planner. For example, to represent group plans (plans that are to be executed by a group of individuals) the planner must possess a meta-relations that associates plans with groups.

Just as a plan designator is a symbol denoting a plan, *meta-predicates* (predicates involving plan designators) are declarative representations of meta-relations. For example, *PLAN(P)* is a declarative statement that *P* is a plan and it may be used in preconditions and effects of tasks.

Finally, *meta-tasks* are tasks involving plans (i.e., one or more of their preconditions or effects is a meta-predicate). It is through meta-tasks that the planner manipulates meta-relations, and when defining meta-tasks, one must also define how its execution modifies the meta-relations of the plans it involves (as illustrated in the next section). Meta-predicates are used to declare the preconditions and effects of these changes, allowing the planner to introspect on and control the planning process.

We emphasize this distinction between meta-predicates and meta-relations because one may not wish to declaratively represent all the meta-relations maintained by the planner (one reason for this could be efficiency concerns). For example, an ExecutePlan task might make a plan executable upon its initiation and make it unexecutable if the task fails, but we might choose not to explicitly represent the executability meta-relation as a meta-predicate.

To implement meta-predicates and meta-tasks, one must alter the planner to reflect the correspondence between meta-predicates and the underlying meta-relations they reflect. This correspondence is formed through the meaning of predicates on current world description. That is, one must provide a mechanism that assigns truth values to any meta-predicates in the CWD. For normal predicates such as *ON(A,B)*, this truth value is assigned by sensing the environment. With meta-predicates, we are essentially making the plan network "part of the environment" and must similarly provide mechanisms for (1) examining the current meta-relations and assign truth values to corresponding predicates, and (2) receiving the commands of meta-tasks and making the appropriate change in the current meta-relations.

As there is a close tie between meta-relations and the planning architecture, we describe in detail a core set of "planner-specific" meta-relations that we have incorporated into our planner. We make no claims to the suitability of this set, but we have found them useful for implementing multi-agent domains, and they serve to illustrate the type of reasoning we wish to support.

Modifiability: Normally, the planner automatically modifies the plan in response to any flaws that may arise. Under some circumstances, one may wish to override this automatic behavior. For example, Cohen and Levesque introduce the notion of joint commitment to lock in a course of action until certain criteria are satisfied. The modifiability meta-relation allows the planner to engage in this deliberate committing and uncommitting to a course of action.

A plan is considered modifiable by default. If a plan is made unmodifiable then the planner is prevented from adding new constraints to the plan or removing existing constraints. Thus, if a task belonging to an unmodifiable plan has an open precondition flaw, the planner is prevented from resolving it (via simple establishment or step addition) as would normally occur. In contrast, if a task in an unmodifiable plan clobbers a task in another modifiable plan, the planner may still add constraints to the modifiable plan to resolve the conflict.

Flaws: A key property of plans is whether they contain any flaws. For example, one may be reluctant to execute a plan that has inconsistencies or missing steps. The flaw meta-relation and corresponding meta-predicate facilitates this type of reasoning. A plan is considered flawed if any of its constraints participate in a flaw.

Conjectures: Often, one would like to consider alternative plans to accomplish a goal before actually committing to a particular course of action. We use the conjecture meta-relation to support this type of reasoning. In particular, if we are considering two conjectured plans for achieving the same goal, we shouldn't allow constraints in one plan to clobber constraints in the other, as could happen since both plans are represented in the same network. If a plan is conjectured, the planner only checks for flaws between constraints within the plan or with constraints of other non-conjectured plans.

Executability: Simply because a planner has created a plan doesn't mean it should execute it immediately, especially in a multi-agent setting where activities may require coordination. Using the executability meta-relation we can bring the execution of plans under deliberate control. If a plan is unexecutable, the planner is prevented from initiating the executing of any tasks in the plan. Tasks

```
GeneratePlan (?group ?order ?plan)
  pre:   order(?group ?order)
         my-group(?group)
  add:   plan-for(?group ?order ?plan)
  del:   flawed(?plan)
  body: :at-start ?plan = make-plan(?order)
         :at-end   disable-modification(?plan)

ExecutePlan (?group ?plan ?order)
  pre:   plan-for(?group ?order ?plan)
         commonKn(?plan ?group) :type maintenance
         -flawed(?plan) :type maintenance
  add:   achieved(?group ?order)
  body:  :at-start  enable-execution(?group ?plan)
         :at-failure transmit-abort(?group ?plan)
         :at-failure disable-execution(?plan)

TransmitPlan (?group ?order ?plan)
  pre:   plan-for(?group ?order ?plan)
         -flawed(?plan)
  add:   commonKn(?plan ?group)
  body:  :at-start transmit-plan(?plan ?group)
         :at-start disable-conjecture(?plan)

RepairPlan (?group ?order ?plan)
  pre:   plan-for(?group ?order ?plan)
         flawed(?plan)
  del:   commonKn(?plan ?group)
         flawed(?plan)
  body:  :at-start  enable-modification(?plan)
         :at-end    disable-modification(?plan)
```

Figure 1: Meta-level domain theory

which have already been initiated will still be terminated if their effects are observed, but no new task may begin

Common Knowledge: Multi-agent planning requires the ability to communicate plans to other agents. The common knowledge meta-relation keeps track of to whom the plan has been communicated to.

We also allow the definition of domain-specific meta-predicates that do not relate to how the planner functions but may be important for a specific domain.

Example

We illustrate the approach by working through an example of how metaplanning works in an application domain. The approach has been tested in the context of large-scale military simulations (Hill *et. al.* 1997) and we use this domain as a basis for discussion. Our system participated in a simulated military exercises known as Synthetic Theater of War '97 (Stow-97). This exercise involved about 5000 independently controlled simulated vehicles (planes, ships, tanks, soldiers, etc.) operating in a 700 by 500 kilometer virtual world and run across a distributed network of several hundred computers. The goal of this exercise was twofold: to support training for the United States Atlantic Command, and to demonstrate advanced simulation technology. This latter goal required generating appropriate and believable behavior, as judged by a group of "subject matter experts" (retired military personnel). Our contribution to this exercise was software for controlling several companies of helicopter entities that engaged in multiple battles against simulated ground vehicles during the course of the 48 hour exercise. Our participation was deemed a success by the standards imposed by the Stow-97 exercise.

In this domain, each company of helicopters consists of five vehicles that coordinate their activities as a group, and a commander agent that performs all high-level planning, replanning, and execution monitoring. The individual vehicle agents execute the commander's high-level plans, making appropriate reactions based on the current state of the environment (Tambe 1997).[2] The commander agent is based on the planning architecture described above, whereas the vehicle agents can be viewed as RAP-like execution systems (Firby 1987) supplemented with Tambe's teamwork and agent tracking techniques.

We illustrate the capabilities of metareasoning by examining the planning performed by the commander agent in the course of a typical exercise. During such an exercise, the command agent receives orders from its commanding unit (a battalion command agent). This consists of the goal of the company's mission, and a partial plan for achieving it. The command agent generates a complete plan to achieve the goal, communicates this plan to the vehicles in its company, and monitors the plan's execution. From the perspective of the vehicle agents, the plan is an abstract specification of their mission and they have significant latitude in executing it with regard to the current environment and any unanticipated circumstances. However, some circumstances may violate the constraints of the commander's plan. In these cases, the commander agent must detect the flaw, repair the plan, and communicate the change to the vehicles in its company.

The commander's planning is represented using two plans in a single task network: a "base-level" plan corresponding to the company's mission, and a "meta-level" plan consisting of tasks that manipulate the base-level plan. We only describe the meta-level in detail. Figure 1 lists the set of meta-level tasks. Currently, planning at the meta-level is quite simple, involving only four tasks. At a high-level, given some new orders, GeneratePlan results in a base-level plan that achieves the orders. TransmitPlan makes this plan common knowledge to all the vehicles in a group (in simulation this corresponds to sending the plan out over simulated radio). If a plan has no flaws and is common knowledge to a group, then that group can execute the plan. Finally, flawed plans can be repaired by

[2] The commander and vehicle agents are all implemented within the Soar agent architecture (Newell 1990).

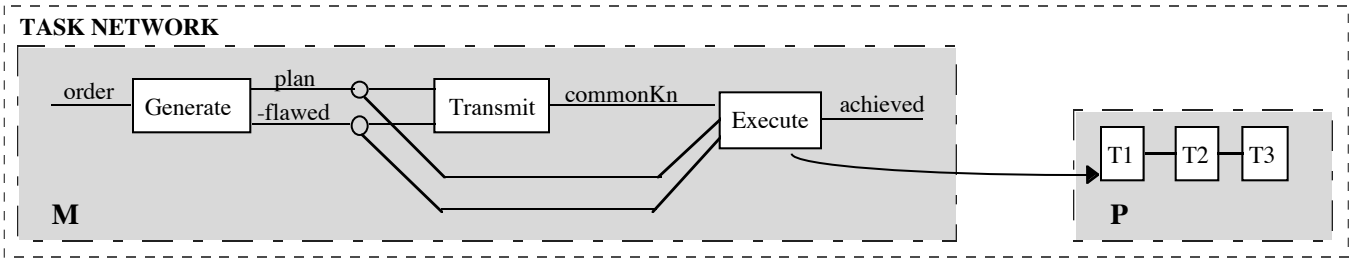

Figure 2: Initial complete meta-level and base-level plans

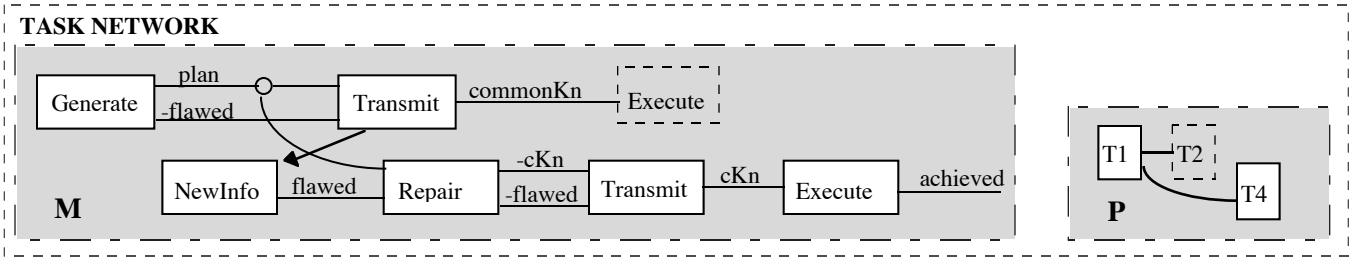

Figure 3: Final meta-level and base-level plans after repair

RepairPlan, but then they are no longer common knowledge. There are two differences between the tasks illustrated in Figure 1 and standard STRIPS operators. The first is the *body* field which names the procedures actually called to implement the task (note that these procedures can return variable bindings). The second is that the preconditions and effects can have types that change what constraints they impose on the plan. For example, the *Not(Flawed(P))* precondition of ExecutePlan is a maintenance condition that must be protected throughout the execution of this task. If a maintenance constraint is violated, any executing task associated with it fails.

Initially the command agent begins with a single meta-level plan, designated by *M*, which consists of two tasks (not illustrated): the *init* task who's effects correspond to the CWD, and an *goal* task who's preconditions are initially empty. The meta-level plan is by default modifiable and non-conjectured.

When new orders are received they are represented by an *ORDER* predicate on the current world description. Simultaneously, via a domain-specific rule, the goal of achieving the order is added to the precondition of the *goal* task. This creates an open precondition flaw in the meta-level plan, which is represented declaratively on the CWD by *FLAWED(M)*. As *M* is modifiable, the planner attempts to eliminate the flaw, eventually adding GeneratePlan, TransmitPlan, and ExecutePlan via step addition and establishing the remaining preconditions via simple establishment. The resulting plan is displayed within the box labeled *M* in Figure 2.

When GeneratePlan is initiated, the make-plan command in its body results in the creation of a new base-level plan, designated by *P*, which is populated initially with a *goal* task who's preconditions represent the goals of the mission, and possibly some abstract tasks that are to be used to accomplish the goal. Typically, the orders will be incomplete (open preconditions, abstract tasks, etc.) which is represented by the predicate *FLAWED(P)* on the CWD. *P* is initially modifiable and conjectured. Since it is modifiable, the planner attempts to resolve the flaws until a viable plan is generated. This results in *FLAWED(P)* being removed from the CWD, which signals the termination of GeneratePlan. As *P* must be communicated and executed with other entities, the planner should avoid changing it without a deliberate decision. Thus, disable-modification is placed in the body of GeneratePlan and its execution makes *P* unmodifiable. Figure 2 illustrates the state of the task network at this point. If all goes as predicted, TransmitPlan and ExecutePlan will be initiated and terminated in turn, completing the mission.

Sometimes things do not go as predicted. For example, perhaps after the group begins executing the plan it receives new information about enemy activity that violates some protection constraint in the base-level plan (say a location that was assumed to be safe to land is now threatened). This threat is represented on the CWD by the reappearance of the *FLAWED(P)* predicate, which in turn violates a maintenance constraint of ExecutePlan, causing this task to fail, and creating an open-precondition flaw at the meta-level, as the achieved predicate is no longer established by the failed task.

Without meta-reasoning, the planner would immediately try to resolve the flaw in the base-level plan *P*, ignoring the other members currently executing this plan, (who are unaware of the flaw). In contrast, as *P* participates in the unmodifiable meta-relation, the planner is prevented from immediately resolving the flaw. It can only modify the plan by deliberately enabling modifications at the meta-level. This deliberation arises in the context of resolving the open-precondition *ACHIEVED* in *M*, resulting in the plan in Figure 3. The planner repairs

the flawed plan, transmits it (as it is no longer common knowledge) and begins execution of the modified plan. (We use search control knowledge to prefer repairing the plan over generating a new plan from scratch.)

Issues and Conclusion

As the example illustrates, plan designators and meta-relations provides the planner an ability to deliberate over the process of planning in a way that facilitates multi-agent reasoning, as well as more complex forms of single-agent reasoning. We can represent multiple plans, assign them properties such as modifiability and executibility, and relate them to other domain constructs. We can associate plans with groups of agents and represent the notion of common knowledge, relationships that are essential for coordinated behavior. And we can do this within the context of well understood classical planning techniques.

This work is still in its early stages and there are many issues to be resolved. One key limitation for multi-agent domains is that tasks may only participate in a single plan. This prevents us from representing the notion that a plan is composed of several subplans. This makes it difficult to reason about "contracting out" parts of a plan to other planning agents. Similarly, we cannot repair parts of a group plan while continuing to execute other parts - even though the planner itself supports an interleaving of planning and execution, as there is no way to communicate that a portion of the plan is to be repaired. Representing these more subtle properties of plans is an active area of current research.

While this work draws connections between planning and multi-agent research, the relationship needs further elaboration. For example, which of the meta-level constructs suggested in the multi-agent literature can be represented using our technique?

Finally, although our metaplanning approach could be incorporated into a variety of classical planners, it may be that different planning techniques are more or less conducive to its successful implementation. One key issue is how the planner searches the space of plans. Maintaining multiple (roughly independent) plans in the same network may favor different search methods than those commonly used. A novel feature of our planner is that it generates plans via local iterative repair search (Kautz and Selman 1996), rather than conventional systematic backtracking search of IPEM and other classical planners. We believe that iterative repair search is more appropriate for planners that perform replanning and it also interacts well with the notion of multiple plans, but this conjecture needs further evaluation.

In conclusion, we have illustrated how a simple form of metaplanning can extend the capabilities of a classical planning system. With plan designators and meta-predicates we can reason about multiple plans and their properties. Besides supporting more complex single-agent reasoning this also provides the framework for representing multi-agent information such as group plans. Finally, our hope is that by drawing a clearer connection between classical planning and multi-agent research, it will support more interchange between these two prolific communities.

References

Ambros-Ingerson, J. A. and Steel, S. 1988. "Integrating Planning, Execution and Monitoring," in AAAI-88.

Cohen, P. and Levesque, H., 1990. "Intention is choice with commitment," *Artificial Intelligence, 42(3).*

Firby, J. 1987. "An investigation into reactive planning in complex domains," In AAAI-87.

Golden, K., Etzioni, O., and Weld, D. 1994. "Omnipotence without Omniscience: Efficient Sensor Management for Planning." in AAAI-94, Seattle, WA.

Grosz, B., and Kraus, S. 1996. "Collaborative Plans for Complex Group Action," Artificial Intelligence, 86(2).

Hill, R., Chen, J., Gratch, G., Rosenbloom, P., and Tambe, M. 1997. "Intelligent Agents for the Synthetic Battlefield," in AAAI-97/IAAI-97, pp. 1006-1012.

Kautz, H. and Selman, B. 1996. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search," in AAAI-96, Portland, OR, pp. 1194-1201.

Knoblock, C. 1995. "Planning, executing, sensing, and replanning for information gathering," in *Proceedings of IJCAI-95*, Montreal, pp. 1686-1693.

McAllester, D. and Rosenblitt, D. 1991. "Systematic Nonlinear Planning," in AAAI-91.

Minton, S., 1990. "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Artificial Intelligence,* 42.

Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Rickel, J. and Johnson, L. 1997. Intelligent tutoring in virtual reality," in *Proceedings of Eighth World Conference on AI in Education,* pp. 294-301.

Stefik, M. 1981. "Planning and Metaplanning," in *Readings in Artificial Intelligence,* Nilsson and Webber, eds., Tioga Publishing, Palo Alto, CA, pp. 272-286.

Tambe, M. 1997. "Agent Architectures for Flexible, Practical Teamwork," in AAAI-97, pp. 22-28.

Wilensky, R. 1980. "Meta-Planning," Proceedings AAAI-80, Stanford, CA, pp. 334-336.