In E. Costa & A. Cardoso (Eds.). (1997). *Progress in Artificial Intelligence: Eighth Portuguese Conference on Artificial Intelligence* (pp. 309-318). Berlin: Springer.

Controlling for Unexpected Goals when Planning in a Mixed-Initiative Setting

Michael T. Cox and Manuela M. Veloso
Computer Science Department. Carnegie Mellon University
Pittsburgh, PA 15213-3891
{mcox;mmv}@cs.cmu.edu

A mixed-initiative setting is where both human and machine are intimately involved in the planning process. We have identified a number of challenges that occur for traditional planning frameworks as humans are allowed more latitude. In this paper we will examine the types of unexpected goals that may be given to the underlying planning system and thereby how humans change the way planning must be performed. Users may want to achieve goals in terms of actions as well as states, they may specify goals that vary along a dimension of abstraction and specificity, and they may mix both top-level goals and subgoals when describing what they want a plan to do. We show how the Prodigy planning system has met these challenges when integrated with a force deployment tool called ForMAT and describe what opportunities this poses for a generative planning framework.

1 Introduction

The objective of mixed-initiative planning is to fully engage the user in automated planning processes. Yet such situations add unexpected challenges to current technologies. Incorporation of the user into an existing automated planner is not accomplished by simply presenting to the user the option of making any decision the system would otherwise make itself, because some decisions a machine might consider may not be appropriate for or understandable to a human user. For instance, the formalism of operator postconditions and preconditions (or a specific hierarchical action decomposition) may not be natural to a user, and the relationship of these conditions to goals and subgoals (or level of refinement) may not be obvious. So it is not realistic to assume that the user will either be familiar with the planning formalisms or willing to learn them. Moreover, the planning system may actually be embedded as an unobtrusive subcomponent of a larger system whose task is only obliquely relevant to planning, so the user's awareness of the planning facility may be marginal. Therefore, a user view must be presented that abstracts the details of the underlying planner. Our belief is that the focus should be upon what the user sees (the interface), what the user wants (the goals), and what the user does (the task). But given such user-centered objectives, the user will inevitably violate the implicit assumptions and expectations of the planner. To compensate, the planner will have to be smart and robust.

As a concrete example of these challenges, we will examine the specification of a user's objectives and their transformation into planning goals. Normally the goals given to a planning system by a knowledgeable and sympathetic user are in a well-structured format. But goals provided by a more realistic and less restrained user present at least three problems to traditional planning systems: (1) *input goals may actually be specified as actions rather than state conditions;* (2) *they may be abstract rather than grounded;* and (3) *they may include subgoals along with top-level goals*. This paper will describe the control exerted by the planner to manage such problems. The second section will introduce the mixed-initiative planning system from which we extrapolate our experience. The subsequent sections describe these three problems and their implementation- and domain-independent solutions. The paper concludes with a brief discussion.

2 Prodigy/Analogy — ForMAT Integration

ForMAT [6, 7] is a case-based planning tool that supports military deployment planning through the acquisition of user-built deployment cases (plans), query-driven browsing of past plans, and functional analysis primitives for evaluating new plans. However, human performance while creating force deployment plans using ForMAT varies as a function of the military experience of the user. This variance appears to be due (in part) to ForMAT's lack of automated support for adapting similar past plans in the context of new planning problems. The more experienced user can accomplish the adaptation task manually, whereas the novice cannot as easily. A technology integration experiment was established between ForMAT and Prodigy/Analogy [9, 11] in order to explore mixed-initiative plan development and adaptation support for force deployment users (see [10, 12] for details).

Prodigy/Analogy is a fully-automated planner that combines generative and case-based planning. The current Prodigy system employs a state-space nonlinear planner and follows a means-ends analysis backward-chaining search procedure that reasons about both multiple goals and multiple alternative operators from its domain theory appropriate for achieving such goals. A domain theory is composed of a hierarchy of object classes and a suite of operators and inference rules that change the state of the objects. A planning problem is represented by an initial state (objects and propositions about the objects) and a set of goal expressions to achieve. Planning decisions consist of choosing a goal from a set of pending goals, choosing an operator to achieve a particular goal, choosing a binding for a given operator, and deciding whether to commit to a possible plan ordering and to get a new planning state or to continue subgoaling for unachieved goals. Different choices give rise to different ways of exploring the search space. These choices can be guided by either control rules, by past problem-solving episodes (cases), or by domain-independent heuristics.

Under a case-based replay mode, Prodigy/Analogy creates plans, interprets and stores planning episodes, and retrieves and reuses multiple past plans that are found similar to new problems. Stored plans are annotated with plan rationale, and reuse involves adaptation driven by the plan rationale. Research to integrate Prodigy/Analogy and ForMAT has investigated sophisticated methods for providing plan modification guidance to the ForMAT user. Guidance from Prodigy suggests to the user how to modify the elements of a past plan to fit the current situation. The sequence of events is as follows.

A ForMAT user receives the description of a new mission. Selecting attributes from the mission description to serve as probes into memory, the user queries ForMAT's database of past plans in search of relevant exemplars with which to build a plan. While browsing, the user refines the mission statement in terms of specific objectives (i.e., goals to be achieved) utilizing a domain-specific goal language. Using past plans as a template, the user edits the old case, substituting new features and values for old, deleting irrelevant old plan steps, and adding necessary new ones. As plan construction proceeds, the user can perform consistency checks on specific aspects of the plan to ensure plan integrity. During these actions, ForMAT sends messages to Prodigy/Analogy, capturing the history of the user actions.

When the mission goals are entered by the user, ForMAT reports this information to Prodigy/Analogy. Given the mission goals, Prodigy retrieves similar past solutions from its own database of plans (a mirror of the ForMAT database in a state-space representation) or creates a new plan generatively given an empty retrieval. It then identifies useful modifications for the past plans as a function of the new and past missions' rationale. Suggestions are sent to the ForMAT user that specify relevant past plans, additional retrieval probes, and potential modifications that the user should perform when building the plan.

3 Goal Specification

One of the obstacles to integrating ForMAT and Prodigy was that ForMAT's user goals were implicit in textual mission statements provided by commanders. State-space planners, such as Prodigy however, need well-defined goals and an initial state description from which to create plans. The goals describe the desired world in an unambiguous manner, specifying the states of the world that must be true after an agent executes the steps of the plan starting with those conditions present in the initial state. Plan creation is accomplished by some combination of forward chaining from the initial state and/or backward chaining from the goal state using operators and inference rules present in the domain theory. To provide Prodigy with goal input we required that ForMAT be modified to explicitly represent the user goals and that these goals be passed to Prodigy. In response, a goal editor was added to ForMAT. Nonetheless, unrestrained users will still specify goals in surprising ways. A user is given a planning problem in the form of a commander's mission statement. The following statement is an example description of a military objective along with guidance towards its achievement.

> *Need a Hawk unit and the 21st Division Ready Brigade to send to Pacifica to secure an airport. Also want to provide security police to keep the airbase secure so that a squadron of A-10As can be forward deployed there.*

The ForMAT user represents these statements in the system's goal editor. The goals specify the military planning objectives and may or may not bear close resemblance to the kind of goals Prodigy expects. When the user saves the goals from the editor, the goals are automatically sent to Prodigy in the representation shown by Figure 1.

```
(:GOALS
(g-146 :SEND-SECURITY-POLICE (GEOGRAPHIC-LOCATION PACIFICA))   )
(g-145 :SEND-BRIGADE ((FORCE 21ST-DIVISIONREADYBRIGADE)
                       (GEOGRAPHIC-LOCATION PACIFICA))   )
(g-144 :SEND-HAWK     ((FORCE HAWK-BATTALION)
                       (GEOGRAPHIC-LOCATION PACIFICA))   )
(g-143 :DEPLOY-A10A   ((GEOGRAPHIC-LOCATION PACIFICA)
                       (AIRCRAFT-TYPE A-10A)) )   )
```

Fig. 1. ForMAT output to Prodigy

For a generative planner, this input presents a number of problems. First, the goals are represented as actions to accomplish, rather than as states to achieve. For example, the input includes goals to send units to Pacifica, rather than to achieve the state of such units' locations being in Pacifica. Second, some goals pertain to particular unit instances (e.g., the 21st Division Ready Brigade); whereas, others pertain to unspecified units of a particular combat type (e.g., a Hawk anti-aircraft unit). Third, both top-level goals and subgoals are sent simultaneously without discretion. In Figure 1, deploying the squadron of A-10A aircraft is the top most goal, while all other goals are in support of this objective. The underlying subgoal structure to a resulting plan in Prodigy is partially shown in Figure 2.

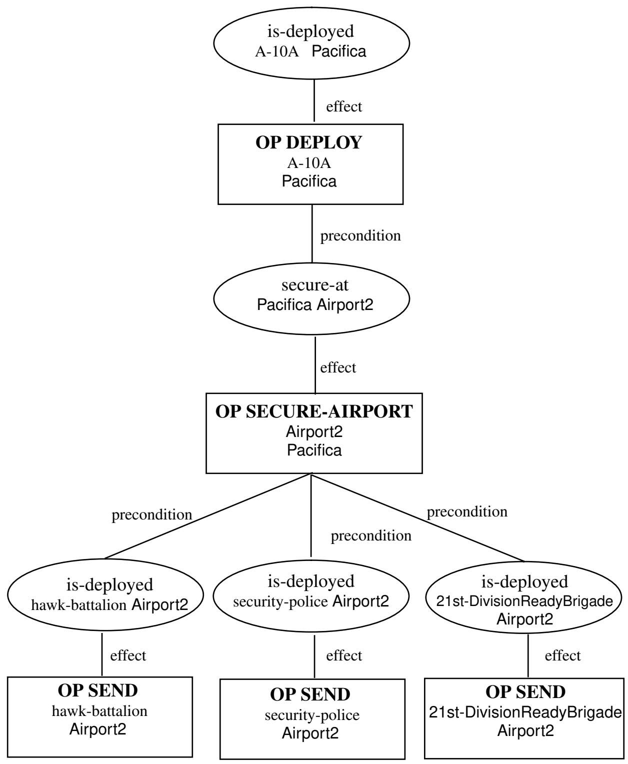

Fig. 2. Prodigy goal tree

4 Goals as Actions

Initially, the impulse was to convince human planners that goals should be represented as desired states of the world, and attempted to provide examples. We claimed that "to send" was an action corresponding to a planning operator that resulted in effects on the world. An operator effect constituted a proper goal. It became apparent, however, that such change would be resisted by users accustomed to thinking in terms of actions and that insisting on such changes might jeopardize the willingness of users to use the system. So the solution was to bypass such a conflict altogether and to build a preprocessor instead that made the transformation to a state representation automatically and in the background.

The preprocessor simply parses each ForMAT goal and obtains the corresponding Prodigy operator from a table. The primary effect of such operators then becomes the translated goal. To some extent, this solution finesses the more general problem of understanding user intent and desire [8]. For example, the translation heuristic assumes that the user does not wish the action to be accomplished due to a side-effect the operator can produce. However, for the purposes of the integration experiment and within the limited domain for which it is used, this solution proves sufficient.

A larger open question exists as to whether humans plan better in terms of actions or states (cf., [4]). Some hierarchical planners support task level specification of goals. See, for example, [13] for an application of hierarchical planning in a mixed-initiative setting for military operations planning. Nevertheless, in the military domain, the key notion of *objective* is important in high level planning, and this concept is often cast in terms of state. In either case, an automated planning component should allow humans to express goals in natural and familiar terms consistent with their language of manual planning.

5 Goal and Operator Hierarchies

In a traditional state-space planner, goals are represented as literals that have a predicate and an arbitrary number of arguments. Thus goal `g-145` from Figure 1 can be represented as the literal (is-deployed 21st-DivisionReadyBrigade Pacifica). However, the FORCE argument of `g-144` is a type identifier rather than an instance. In the Prodigy system, this is easily represented by specifying an existentially quantified goal such as (exists ((<hwk> hawk-battalion)) (is-deployed <hwk> Pacifica)). The goal is solved if some unit that is an element of the class hawk-battalion is deployed in Pacifica.

However given a hierarchy of types, a difficulty is that different operators may apply to goals up and down the abstraction hierarchy. For instance, the operator SECURE-AIRPORT is used to make secure a specific airport (see Figure 3); whereas a more general SECURE operator is used to make secure objects such as hills for which no specialized operators exist. The difference between the two is that the second operator does not have an <air-defense> variable (nor the precondition associated with it) and the variable <obj> is of type OBJECT, rather than a more specific type AIRPORT (and thus the effect and preconditions are more abstract).[1] The first operator can be used to achieve (secure-at Pacifica Airport2), while the second is appropriate when achieving either a literal such as (secure-at Pacifica Hill21) or an existentially quantified goal such as (exists ((<obj> OBJECT)) (secure-at Pacifica <obj>)).

The advantage of reasoning about such an operator hierarchy is that when achieving novel goals, the more general operator can be applied when no specialized operator is available.[2] However, when the user wants to secure Airport2, both operators are now licensed because Airport2 is a member of the class AIRPORT, but is also a member by transitivity

1. In general an operator can be made be more widely applicable by dropping preconditions and by abstracting its variables.

2. They can also be applied when preconditions for the more specific operator is unsatisfied. If additional information arrives at planning time or during execution, then such general operators can be specialized dynamically.

secure-airport (<obj>, <loc>)
<loc>: type LOCATION
<obj>: type AIRPORT
<internal-security>: type POLICE-FORCE-MODULE
<external-security>: type TROOPS
<air-defense>: type ANTI-AIR-MODULE
Pre: (loc-at <obj> <loc>)
(is-deployed <internal-security> <obj>)
(is-deployed <air-defense> <obj>)
(is-deployed <external-security> <obj>)))
Add: (secure-at <loc> <obj>)))))

Fig. 3. Secure operator (representation adapted from [11])

of the class OBJECT. Thus, the effect of either operator will unify with the goal and so both are applicable. But clearly the planner should choose the more specific operator.

Thus it is useful to think of an existing hierarchy of operators that depends on the semantics of their effects. To formalize this notion, consider that one goal may be an ancestor of another goal. We have already mentioned that both literal and quantified goals exist in the Prodigy framework. But in more general terms we want to argue for the notion that the goal (is-deployed HAWK-BATTALION Pacifica) is more specific than (i.e., is a descendent of) the goal (is-deployed ANTI-AIR-MODULE COUNTRY), given that the first goal is a short-hand notation for the existential goal introduced earlier.

Abbreviation 1: Given G = (predicate arg1 .. argn) *such that* predicate $\in$ Existing-Predicates *and given type hierarchy graph* $H = (V, E)$ *where* V *is the set of class vertices and* E *is the set of incident edges* $\langle x\ y \rangle$ *such that* x *is a member of the class set* y. *Then* G *above is an abbreviation for the expression*

$$\forall\, arg_{i=1..n} \mid arg_i \in V, \exists\, x_i \mid x_i \in arg_i, (predicate\ x_1\ x_2\ ..\ x_n).$$

Definition 1: *Given that* $Arg(G, i)$ *is the ith argument of goal* G *as abbreviated above*, *we define goal* G_1 *to be an* ***ancestor*** *of* G_2 (G_2 *is a* ***descendant*** *of* G_1) *in hierarchy* H *iff*

(1) $arity(G_1) = arity(G_2) = n$
(2) $predicate(G_1) = predicate(G_2)$
(3) for $i=1..n$, $Arg(G_1, i) \geq Arg(G_2, i)$ *in* H

where the relation "$\geq$" *between arguments is defined as equivalence or class inheritance such that a path exists through* H *from* $Arg(G_2, i)$ *to* $Arg(G_1, i)$. *That is, the edges* $\langle Arg(G_2, i)\ e_1 \rangle\ \langle e_1\ e_2 \rangle \ldots \langle e_k\ Arg(G_1, i) \rangle$ *are all elements of edge list* E.

Knowing this property, the planner can control its choice of operator when solving goals in a hierarchy. Given a goal such that two or more operators apply to the goal, if one operator is an ancestor of another, then Prodigy should prefer the more specific operator. Control rule Prefer-More-Specific-Op in Figure 4 implements this preference. Two operators to be compared are bound to the rule using the standard meta-predicate `candidate-operator` [1]. The function `is-ancestor-op-of-p` is a user defined meta-predicate that returns t *iff* the primary effect [2] of both operators are not equal and the primary effect of the first operator is an ancestor of the primary effect of the second operator (as specified by the "$\geq$" relation of Definition 1).

```
(CONTROL-RULE Prefer-More-Specific-Op
  (if (and (candidate-operator <OP1>)
           (candidate-operator <OP2>)
           (is-ancestor-op-of-p <OP1> <OP2>)))
  (then prefer operator <OP1> <OP2>) )
```

Fig. 4. Given two applicable operators from which to choose, prefer the more specific one

6 Top-level Goals Versus Subgoals

Finally, the goal information ForMAT sends to Prodigy always contains a mix of top-level goals and lower level constraining information. Myers [5] considers such information to be constraint advice, although in the context of state-space planning, we view this advice as simply a subgoal specification. Given that the user provides both subgoals and top-level goals within an agenda, two decisions need to be addressed by a mixed-initiative system. First, for which class of goals should the system plan first? Should it proceed bottom up or top down and why? Second, given that it goes top down, how should the system serendipitously take advantage of the existing information the subgoals provide?

6.1 Order of Planning

Given two goals such as G-143 = (is-deployed A-10A Pacifica) and its subordinate goal G-145 = (is-deployed 21st-Division-Ready-Brigade Pacifica), a planner will first plan for one and then the other. If the subordinate goal is achieved first, thus establishing the brigade in Pacifica, then the precondition of operator SECURE-AIRPORT having <external-security> in <loc> will already be true when planing for the superodinate goal (review Figures 2 and 3).

The problem with this approach, however, is twofold: First, we want to make sure that if more than one operator exists to achieve the subordinate goal, then the plan chosen is consistent with the goals above it in the goal tree so that backtracking is avoided and the plan remains consistent. The top-level goals need to provide guidance to their subgoals.

Second, in this domain, the user should view the planning process and the evolving plan in an understandable, top-down way, rather than in a disjoint fashion as subgoals are randomly assembled. Successive-refinement planning is appropriate in domains that exhibit hierarchical structure, when time is a scarce resource, and when reliable abstract plans exist in the domain [3]. So given a choice of goals to achieve, we want to choose the one highest in the subgoal tree. That is, Prodigy should choose the one that is a supergoal of the other.

Definition 2: G_2, *is a **subgoal** of* G_1, *(or alternatively,* G_1 *is a **supergoal** of* G_2*) iff*

(1) *a backward operators chain exists* $OP_1 .. OP_n$ *from* G_1 *to* G_2 *such that* $\forall OP_{i=1..n-1}$ $\exists p,e \mid p \in pre(OP_i), e \in eff(OP_{i+1}), p$ *unifies with* e *under substitution* σ

(2) $\exists p \mid p \in pre(OP_n)$, *and* p *unifies with* G_2 *under* σ

(3) $\exists e \mid e \in eff(OP_1)$, *and* e *unifies with* G_1 *under* σ .

In the case of n=1, a single operator, OP, exists whose effects includes the effect $e \in eff(OP)$ that unifies with G_1 under the simple substitution, σ, and whose preconditions includes $p \in pre(OP)$ that unifies with G_2 also under σ.

Given Definition 2, control rule Prefer-Top-Most-Goal in Figure 5 can choose goals bound to supergoal <G1> over any of its subgoals bound to <G2> when the meta-predicate `subgoal-of-p` returns t. This occurs when some operator in Prodigy's subgoal tree for <G1> also achieves <G2> earlier in the plan.

```
(CONTROL-RULE Prefer-Top-Most-Goal
 (if  (and (candidate-goal <G1>)
           (candidate-goal <G2>)
           (subgoal-of-p <G2> <G1>)))
 (then prefer goal <G1> <G2>) )
```

Fig. 5. Given two goals, prefer one if making the other true solves one of the preconditions for an operator that results in the preferred

In a large domain, a direct implementation of this control rule will result in inefficiency because when two goals are independent, the metapredicate must search the entire space of plans for both goals. To alleviate exponential search, a heuristic can be incorporated into the metapredicate that either places a bound on, n (the number of operators in the chain specified in condition 1 of Definition 2), or it can refer to a cache table maintained during past planning episodes that map goal-subgoal relations.

6.2 Opportunistic Planning

The policy established by Prefer-Top-Most-Goal creates another problem. If a top level goal is established first, such as `G-143`, then no guarantee exists that bindings established in plan operators such as SECURE-AIRPORT will agree with deferred subgoals. In the plan for deploying the A-10A, external security may be established by binding <external-security> with an instance of type TROOPS other than the 21st Division Ready Brigade.

Figure 6 shows a control rule that watches for propitious opportunities to optimize a plan by preferring bindings that also achieve additional pending goals. Given candidate bindings for the current operator in the search tree, meta-predicate `match-constraining-goals` identifies pending subgoals that unify with preconditions of the current operator. New bindings are then generated that satisfy this goal. Out of the candidate bindings, therefore, the control rule distinguishes ones that are consistent and are not consistent with such pending goals, preferring the former to the latter.

```
(CONTROL-RULE Prefer-Bindings-Opportunistically
(if (and   (current-operator <OP>)
           (candidate-bindings <CB>)
           (match-constraining-goals <G> <OP>)
           (generate-new-bindings <NB> <G> <OP>)
           (identify-worse-bindings <CB> <NB> <WB><OP>)
           (identify-better-bindings <CB> <NB> <BB><OP>)))
(then prefer bindings <BB> <WB>))
```

Fig. 6. Given current operator and candidate set of bindings, prefer bindings that opportunistically solve another pending goal

7 Conclusion

The success of these control rules in managing the variety of goal types provided to Prodigy by the ForMAT user has been established and validated through trials conducted in real-time across the internet with Prodigy/Analogy operating from CMU in Pittsburgh and ForMAT from Mitre in Boston. (see [12]). Military planners have used the system in a limited fashion, but one that combines all three situations described herein.

Our observation is that a major trade-off exists between the goal of maintaining the simplifying assumptions of traditional planning systems and the goal of allowing the user more control and decision-making flexibility in mixed-initiative systems. Experience from the technology integration experiment performed between the Prodigy and ForMAT planning systems shows that even simple tasks presents challenges to the automated component. The underlying planning system must be flexible enough to allow the user to express desires and goals in terms most natural to the user and, therefore, cannot unconditionally expect to impose all of its decision-making representation on the user community if it is to be effective. The choice for the system developer then is to choose the right user restrictions on which to insist (e.g., requiring the user to explicitly represent their planning goals). Here we have discussed three potential problems that the technology can manage with internal control, thus avoiding user compromise altogether.

Specifically we examined problems faced by traditional planning systems when obtaining goal input from the user. Most planning systems make three implicit assumptions that the user will invariably negate. These assumptions are (1) the goal input is in terms of desired states of the world; (2) these goal states are literals grounded in specific instances; and (3) the goals are strictly top-level goals. In mixed-initiative planning systems, however, the user will present goals to the planner in terms of actions, the goals will range in specificity along the domain's abstraction hierarchy, and the user will mix both subgoals and top-level goals in the input. Our solutions to these problems have been a mix of preprocessing and internal planning control. Preprocessing is used to translate actions into states. Control rules prefer appropriate abstract operators for the given level of goal abstraction. Control rules also prefer top-level goals before lower-level goals and then prefer bindings for operators that opportunistically solve user-provided subgoals.

Unexpected goals are not simply problems to be overcome; rather, they represent user-provided hints on how to construct good plans. The use of abstract goals can allow the system to avoid overfitting the plan. Most automated planners construct plans that are too specific for subsequent plan steps when uncertainty exists as to the outcome of previous plan steps. Managing relative plan abstraction is essential for effective planning given substantial temporal extent. Moreover, the subgoal constraint information that users provide often originates from details outside the transitive closure provided by the domain theory. Given a more detailed domain theory, the planner could infer these constraints directly, but at the cost of considerable search. Thus, being able to handle constraint information from the user allows the planner to be more efficient in practical problems. The challenge involved with mixed-initiative systems is to engineer a more robust mechanism that is flexible with respect to user needs. The opportunity is to leverage user experience.

Acknowledgments

This research is sponsored as part of the DARPA/RL Knowledge Based Planning and Scheduling Initiative under grant number F30602-95-1-0018. The authors thank Eugene Fink, Alice Mulvehill, and Gary Pelton for comments and suggestions on earlier drafts of this publication.

References

[1] Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, A.; Reilly, S.; Veloso, M. M.; and Wang, X. 1992. *PRODIGY4.0: The Manual and Tutorial,* Tech. Rep., CMU-CS-92-150, Deptartment of Computer Science, Carnegie Mellon University.

[2] Fink, E, and Yang, Q. in press. Automatically Selecting and Using Primary Effects in Planning: Theory and Experiments. *Artificial Intelligence.*

[3] Hayes-Roth, B., and Hayes-Roth, F. 1979. A Cognitive Model of Planning. *Cognitive Science* 3: 275-310.

[4] McDermott, D. 1978. Planning and Acting. *Cognitive Science 2*: 71-109.

[5] Myers, K. L. 1996. Strategic Advice for Hierarchical Planners. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning*, 112-123. San Francisco: Morgan Kaufmann.

[6] Mulvehill, A. 1996. Building, Remembering, and Revising Force Deployment Plans, In A. Tate ed. *Advanced Planning Technology*, 201-205. Menlo Park, CA: AAAI Press.

[7] Mulvehill, A., and Christey, S. 1995. *ForMAT - a Force Management and Analysis Tool*. Bedford, MA: MITRE Corporation.

[8] Pollack, M. E. 1990. Plans as Complex Mental Attitudes. In P. R. Cohen, J. Morgan and M. E. Pollack eds. *Intentions in Communication*, 77-104. Cambridge,MA: MIT Press.

[9] Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. New York: Springer-Verlag.

[10] Veloso, M. M. 1996. Towards Mixed-Initiative Rationale-Supported Planning. In A. Tate ed. *Advanced Planning Technology*, 277-282. Menlo Park, CA: AAAI Press.

[11] Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical Artificial Intelligence 7*(1): 81-120.

[12] Veloso, M.; Mulvehill, A.; and Cox, M. in press. Rationale-Supported Mixed-Initiative Case-based Planning. To appear in *Proceedings of the Ninth Annual Conference on Innovative Applications of Artificial Intelligence*. Menlo Park, CA: AAAI Press.

[13] Wilkins, D., and Desimone, R. 1994. Applying an AI Planner to Military Operations Planning. In M. Zweben and M. Fox eds. *Intelligent Scheduling*, 685-709. San Mateo, CA: Morgan Kaufmann.