Strategic Advice for Hierarchical Planners

Karen L. Myers
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com

Abstract

AI planning systems have traditionally operated as stand-alone blackboxes, taking a description of a domain and a set of goals, and automatically synthesizing a plan for achieving those goals. Such designs severely restrict the influence that users can have on the resultant plans. This paper describes an Advisable Planner framework that marries an advice-taking interface to AI planning technology. The framework is designed to enable users to interact with planning systems at high levels of abstraction in order to influence the plan generation process in terms that are meaningful to them. Advice consists of task-specific constraints on both the desired solution and the refinement decisions that underlie the planning process. The paper emphasizes *strategic advice*, which expresses recommendations on how goals and actions are to be accomplished. The main contributions are a formal language and semantics for strategic advice, and a sound and complete HTN-style algorithm for generating plans that satisfy advice.

1 Introduction

Artificial Intelligence (AI) planning technology provides powerful tools for solving problems that involve the coordination of actions in the pursuit of specified goals. The AI community has produced several planning systems whose demonstrations on realistic problems attest to the value of automated planning techniques. Nevertheless, there has been limited success in transitioning this technology to user communities. A major reason for the lack of technology transfer lies with the difficulty of using planning systems. AI planners have traditionally been designed to operate as 'blackboxes': they take a description of a domain and a set of goals and automatically synthesize a plan for achieving the goals. This design has several drawbacks. First, it explicitly limits the amount of influence that a user can have on the generated plans. Second, it requires complete and accurate formalizations of the domain, since the system is expected to operate without user intervention. Providing such comprehensive domain information is time-consuming and expensive, and represents a significant investment for each new application.

Recent trends toward *mixed-initiative* styles of planning have led to support for certain low-level interactions on the part of humans, such as ordering goals for expansion, selecting operators to apply, and choosing instantiations for planning variables [3, 16]. While a step in the right direction, these interactions are too fine-grained for most users, who want to be involved with the planning process at a higher, more strategic level.

This paper describes an effort to make AI planning technology more accessible and controllable via the paradigm of *advice-taking*. An *advisable planner* (AP) will accept a variety of instructions and advice from a user and employ those directives to guide plan construction. Advice can encompass a broad range of constructs, including partial sketches of plans for achieving a set of goals, specific subgoals to be used in pursuit of the overall objectives, and proscriptions and prescriptions on the use of specific objects and actions in certain contexts. Such advice will enable users to interact with a planning system at high levels of abstraction in order to guide and influence the planning process, with the planning system performing the time-consuming work of filling in necessary low-level details. As such, the AP model of planning embodies a shift in perspective on how planning systems should be designed: an Advisable Planner is a tool for enhancing the skills of domain users, not a replacement for them.

Within our model, advice differs from the forms of knowledge traditionally used to represent a planning domain (operator schemas, world models, *etc.*). Rather than expressing general properties of the do-

main, advice encodes *session-specific* recommendations on the application of traditional planning knowledge. As such, advice is an adjunct to the underlying domain knowledge rather than an extension to it. As will be seen, advice serves as a filter on the set of solutions to a given planning problem, thus enabling customizations to suit the needs of individual users.

To illustrate the utility of advice, we present examples from a travel planning domain. A typical HTN planner would allow a user to sketch a high-level outline of a trip, specifying information such as which locations to visit at what times and for what overall cost. The planner would then fill in the appropriate details. Most individuals, however, want to influence their itineraries to a much greater extent, specifying details such as the modes of transportation for various legs, individual carriers to use, accommodation requirements, and restrictions on costs for various aspects of the trip. Our theory of advice enables user customization of generated plans in this way.

This paper lays the foundations for the Advisable Planner. Here we emphasize *strategic advice*, which expresses recommendations on how goals and actions are to be accomplished. Section 2 presents an overall framework for an advisable planner. Section 3 describes strategic advice and its uses. Section 4 presents the formal model of HTN planning that underlies our theory of advice. Section 5 defines a formal language for representing strategic advice, while Section 6 defines *satisfaction* for advice. Section 7 defines a sound and complete HTN-style algorithm for generating plans that satisfy strategic advice. Section 8 discusses related work.

2 The Advisable Planner

This section outlines our vision for an *advisable planner* (AP), describing both the overall architecture and the principal classes of advice that an AP should support. As such, it establishes a conceptual framework in which to ground the more technical work that follows.

2.1 AP Framework

The Advisable Planner model consists of an advice-taking interface layered on top of a core planning system. Advice-taking augments the capabilities of the underlying planning system in the sense that the system does not require advice for its operation. Rather, advice simply influences the set of solutions that the system will provide for a given task. Overall, the AP contains two distinct phases: the *advice translation* phase, and the *problem-solving* phase.

Advice translation involves mapping from user-supplied advice into appropriate internal representations for the planner. The translation process involves several stages (see Figure 1). User advice, specified in some natural (or pseudo-natural) language, is *parsed* into an intermediate normalized representation. The normal form provides a planner-independent representation of the advice, thus enabling a clean semantic definition and portability amongst different planners. *Advice compilation* is the planner-dependent translation from normalized advice to internal constraints defined in terms of planner-specific operators, goals, and individuals. These constraints will be used to direct the plan construction process.

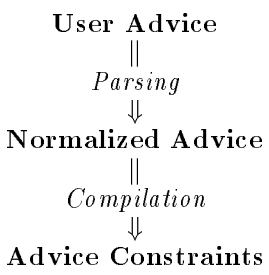

Figure 1: Phases of Advice Representation and Translation

The problem-solving phase takes the compiled advice representations, and generates only solutions that satisfy the advice (referred to as *advice enforcement*). Planning proceeds in a mixed-initiative style whereby the user can make planning-time requests to modify current plans or previously stated advice. The AP may request additional domain information from the user during planning to aid in resolving detected trade-offs in the plan, to recover from planning failures, or to clarify user-supplied advice.

This paper focuses on the presentation of a basic theory of strategic advice and accompanying algorithms for advice enforcement, ignoring the many complex issues related to advice parsing. In particular, we assume the provision of normalized advice in a language shared with the underlying planning system. For exposition reasons though, we generally use natural-language style presentations of advice.

2.2 Advice Categories

Many kinds of advice can be fruitfully employed for generative planning. *Task advice* designates specific goals to be achieved and actions to be performed, thus amounting to partial specification of a solution to a planning task. A plan sketch constitutes a form of task advice, as do directives such as Do Task-1 before Task-2 and Include Task-5 in the plan. *Evaluational advice* encompasses constraints on some metric defined for the overall plan, such as resource usage, execution time, or solution quality. The directive Spend less than $800 on all accommodations is an example.

This paper focuses on *strategic advice*, which expresses constraints on how to solve tasks, in terms of both specific approaches to pursue and entities to employ. Strategic advice comes in two forms: *role* and *method*. Role advice constrains the use of domain entities in solving tasks, while method advice constrains the type of approach used. As will be seen, strategic advice designates not only properties of the resultant partially ordered set of actions that is generally viewed as 'the plan', but also the underlying justifications for that solution. For this reason, advice enforcement requires explicit representation of those justification structures.

Our model of advice does not encompass *control* of problem-solving [2, 14], such as the order in which to expand goals or instantiate variables. While control is an important issue for automated planning systems, effective control of the planning process requires deep insight into the mechanics of the planning system itself. As such, it is not a responsibility that should be borne by the user (who generally will not be an expert in AI planning) and thus is not an appropriate topic for user advice.

3 Strategic Advice

Strategic advice is formulated in terms of prescriptions and restrictions on *roles*, *fillers*, and *activities*. *Activities* represent abstract operations relative to the underlying planning domain, and are defined in terms of *features* and *roles*. A feature designates a characteristic of interest for an activity. For travel planning, there may be transport activities, vacation activities, bike activities, and accommodation activities; each of these characteristics could be modeled as a feature. A given activity can have multiple features; for example, an activity corresponding to a bike tour could have the features Vacation, Bike, and Inexpensive. *Roles* correspond to capacities in which domain individuals are to be used in an activity. For instance, transport activities could have roles such as Origin, Destination, and Carrier. *Fillers* are specifications of objects to be used in certain roles; they may name explicit individuals, or consist simply of a set of required and prohibited attributes.

As described later, activities, roles, and fillers are grounded in planning constructs such as goals, operators, variables, and bindings. The benefit of the activities/roles framework is that it provides users with a simpler, more abstract model for expressing advice than do the low-level planning constructs.

3.1 Role Advice

Role advice either prescribes or restricts the use of domain entities for filling certain capacities in the plan. Role advice is characterized by the template: `<Use/Don't Use> <object> in <role> for <target-activity>`.

In general, role advice consists of one or more object-role specifications (called a *role-fill*), a *target activity*, and a *polarity* indicating whether the advice is prescribing or prohibiting the role-fill. The following directives provide examples of role advice:

> Stay in 3-star ensuite hotels while vacationing in Scotland.
> Layovers longer than 90 minutes are unacceptable for domestic flights.

The first directive imposes requirements on accommodations during vacations in a given area. The second prohibits flights with long layovers. Here, we use natural language renderings of advice to aid understandability, but it is easy to map to our structured model. For the first example, the target activity is defined as operations with feature Vacation, and with role Location filled by Scotland. The advice dictates that the filler for the role Accommodation be a 3-star hotel with ensuite facilities.

3.2 Method Advice

Method advice imposes restrictions on the approaches that can be used in solving a goal or class of goals. It is characterized by the template: `<Use/Don't use> <advised-activity> for <target-activity>`. Thus, method advice consists of target and advised activities, along with a polarity expressing prescription or proscription. For example:

> Find a package bike tour starting in Athens for the vacation in Greece.
> Don't fly between cities less than 200 miles apart.

The first piece of method advice declares that the approach used for a particular portion of the trip should have certain features (*i.e.*, Bike, Package) and role constraints (*i.e.*, start location is Athens). The second specifies restrictions on the approach to be taken for solving a class of transport goals.

3.3 Observations

Advice can be either *abstract* or *grounded*. The former constitutes recommendations that apply to a class of goals and operations; the latter provides recommendations relative to a specific goal or activity. For instance, the advice Use TWA for transatlantic flights is abstract, relating to travel for a class of destinations; in contrast, the advice Use TWA to fly to London relates to a specific city (and hence, a specific travel goal).

Ascertaining whether a piece of advice is satisfied requires more than an examination of the final set of partially-ordered actions that are generally viewed as

'the plan'. As an illustration, consider the advice Stay in 3-star ensuite hotels while vacationing in Scotland, in the context of a trip that includes both business and holiday travel. A final plan for this trip would consist of a set of actions at the level of movements to destinations, stays in accommodations, and tours of various sights. Direct examination of a particular accommodation action in the final plan will not reveal its purpose (business or pleasure); hence, it is not possible to verify that the supplied advice has been followed. In general, verification of advice satisfaction requires examination of the overall problem-solving context in which planning decisions were made.

3.4 Parsing Issues

The examples above illustrate that a range of different surface forms can be mapped into the advice templates. As noted earlier, the *advice parser* is responsible for this mapping. While a discussion of advice parsing is beyond the scope of this paper, we briefly mention some relevant issues.

Advice parsing will generally involve domain-specific interpretation of words to extract role-fills and activities, and to identify use prescription or prohibition. For example, interpretation of Stay in 3-star hotels in Scotland as advice requires an understanding that Stay corresponds to a use prescription.

Roles may not be named explicitly in advice. In the advice Use JAL to fly to Japanese destinations, the role Destination is explicit but the role Carrier is not. In many cases, inference can be used to extract implicit roles. For instance, since JAL is an air carrier, its role could be deduced by identifying variables of the appropriate type in air-travel operators.

Surface-level advice such as Travel first-class on trains can be interpreted in multiple ways. The most basic interpretation is as a directive to employ a particular approach for tasks that involve train travel (should the need for such transportation arise). A more complex interpretation would further attribute an implicit goal to use trains (say, rather than driving). The former corresponds to strategic advice, the latter to a combination of strategic and task advice. The intended interpretation is impossible to ascertain without additional information. Because our focus here is on strategic advice, we ignore any such implicit task designations.

4 Planning Model

Our definition of advice satisfaction requires a model for the underlying planning framework. We employ a Hierarchical Task Network (HTN) model, based loosely on that in [4].

4.1 HTN Planning

The cornerstones of HTN planning are *task networks* and *operator schemas*. Informally, a task network is a partially-ordered set of tasks (goals and actions)[1] to be achieved, along with conditions on the world state before and after tasks are executed. Operator schemas specify methods for reducing an individual goal to some new set of subgoals and actions, under appropriate conditions. HTN planning consists of taking a description of an initial world state, an initial task network, and a set of operator schemas for goal refinement, and then repeatedly applying operator schemas until no further reductions are possible. Each such refinement may contribute additional task-ordering and world constraints to the successive HTNs.

Formally, we define a task network $\tau = \langle T, L, W \rangle$, where T is a set of tasks, L is a set of ordering constraints on tasks, and W is a set of world constraints. Tasks can be either *primitive* or *nonprimitive*, with the former having no possible further refinements. We say that a task network is primitive when it contains only primitive tasks. In this document, we use the terms *task network* and *plan* interchangeably; *partial plan* is used to designate a nonprimitive task network.

An HTN planning problem $\mathcal{P} = \langle \mathcal{O}, \tau_0, S_0 \rangle$ is modeled as a set of operator schemas $\mathcal{O}$, an initial task network τ_0, and a set of propositions S_0 denoting the initial world state. An operator schema is characterized by its purpose $Purpose(O)$ (*i.e.*, the goals to which it can be applied), the preconditions for applying the schema $Preconds(O)$, and the task network $Tasks(O)$ to which a goal matching the purpose can be reduced by applying the schema. The world state, goals, purpose, and preconditions are modeled using a first-order language $\mathcal{L} = \langle Vars, Preds, Consts \rangle$. A solution to an HTN problem is a refinement of the original task network that contains only primitive tasks, provided that all constraints in the task network can be *resolved*. We call the resultant resolved task network a *completed* task network.

To enable representations of strategic advice, an HTN domain is extended to include declarations of *feature* and *role* information for operator schemas. Such declarations constitute metalevel information about the domain and generally are not captured in standard planning models. We use the notation $Features(O)$ to designate the features of an operator schema; role information is captured by a *role-resolution* function $RoleVar(O, R)$ that maps an operator schema and a role to the variable that implements the role in the schema (if one exists). Advice is represented in a metalanguage defined over $\mathcal{L}' = \langle Vars, Preds, Consts, Features, Roles \rangle$.

[1] We adopt an action-oriented rather than state-based interpretation of goals, as is consistent with HTN planning.

An AP problem $\mathcal{AP}$ generalizes an HTN problem to include a set of advice $\mathcal{A}$; that is, $\mathcal{AP} = \langle \mathcal{O}, \tau_0, S_0, \mathcal{A} \rangle$. Informally, a plan is a solution to an AP problem iff it is a solution to the underlying HTN problem and further *satisfies* the advice in $\mathcal{A}$ (as formalized in Section 6). Thus, advice acts as a filter on the set of acceptable solutions to a given planning problem.

4.2 Plan Refinement Structures

As noted above, advice imposes restrictions not only on the completed plan but also on the task refinement process by which the plan is derived. For this reason, a model of advice satisfaction must reference the overall context in which plan refinement decisions were made. We define the *(partial) plan refinement structure* for a (partial) plan (similar to the *hierarchical task network* for a plan in [8]) to be $\pi = \langle \mathcal{P}, \mathcal{N}, \mathcal{D} \rangle$ where $\mathcal{P}$ is the set of task networks produced, $\mathcal{N}$ is the set of nodes in any of the task networks, and $\mathcal{D}$ defines a directed acyclic graph of the refinement relations from a node to each of its descendants.

Each node in a plan refinement structure has attributes defined by its associated task network. One key attribute for processing advice is the partial world model $State(n)$, which captures the relations that necessarily hold prior to the execution of the node's task [1]. Additional node attributes include the goal/task for the node $Goal(n)$, the operator schema that has been used to expand that node $OprSchema(n)$, and the operator bindings for the expansion $\sigma(n)$. $Desc(n)$ designates a node and its descendants.

5 Advice Representation Language

In this section, we define a formal language in which to represent strategic advice, and show how instances of the advice templates map into these representations. For simplicity, we assume a common language of features, roles, and relations for the representations of normalized advice and the planning domain, thus obviating the need for a translation step between them.

Representations for advice are constructed from *role restrictions* and *generalized activities*.

Role Restrictions A role restriction $\langle R, F[x] \rangle$ consists of a *role identifier* R from the set of role symbols *Roles*, and a *filler* $F[x]$ represented as a propositional formula defined over variable x. This formula is composed from the language $\mathcal{L}$ of the underlying planning domain.

Generalized Activities A generalized activity is a pair $< \mathcal{F}, \mathcal{R} >$ consisting of a set of required and prohibited operator features $\mathcal{F} = \mathcal{F}^+ \cup \mathcal{F}^-$, and a set of role restrictions $\mathcal{R}$. A generalized activity does not necessarily map to an instance of any particular operator. In particular, a generalized activity may include role restrictions that span multiple levels of task refinement (*i.e.*, operator application). As such, a generalized activity represents an abstract specification of a plan wedge.

Role and Method Advice As described in Section 3, role advice consists of one or more role-fill descriptions, a target activity, and a polarity (prescribing positive or negative use). Similarly, method advice is characterized as an advised activity, a target activity, and a polarity. A piece of role advice with role-fill ρ and target activity α_G is translated into the representation $(ROLE^+ \ \rho \ \alpha_G)$ if the advice is positive, and $(ROLE^- \ \rho \ \alpha_G)$ if negative. Similarly, a piece of method advice with advised activity α_M and target activity α_G is represented as $(METHOD^+ \ \alpha_M \ \alpha_G)$ if the advice is positive, and $(METHOD^- \ \alpha_M \ \alpha_G)$ if negative.

6 Advice Satisfaction

We now proceed with the definition of *satisfaction* for advice, which is grounded in plan refinement structures for primitive task networks. Overall, we say that a plan refinement structure satisfies a piece of advice if each node in the structure satisfies the advice. Satisfaction of advice by a node is defined below, building on definitions for satisfaction for role restrictions and matches for generalized activities.

Here, we adopt certain notational conventions. We write $v : c \in \sigma$ to indicate that variable v is bound to the value c in the substitution σ. Instantiations of an operator schema O or formula ϕ for bindings in σ are written as O^σ and ϕ^σ. The notation $\overline{\phi}$ is used to represent the logical complement of a formula ϕ. Finally, we define $Opr(n) = OprSchema(n)^{\sigma(n)}$.

Satisfaction of a role restriction ρ by a node requires that the variable that models the role for the node's operator be bound to a term that satisfies the fill constraints in ρ.

Definition 1 (Role Restriction: Node) *Let n be a node in a plan refinement structure with $O =$ OprSchema(n), and let ρ be a role restriction $\langle R, F[x] \rangle$. Then:*

- *n violates ρ iff* RoleVar$(O, R) = v$ *is defined,* $v : c \in \sigma(n)$, *and* State$(n) \not\models F[c]$,
- *n directly satisfies ρ iff* RoleVar$(O, R) = v$ *is defined,* $v : c \in \sigma(n)$ *and* State$(n) \models F[c]$.

Note that a node n may neither directly satisfy nor violate a role restriction. Such a situation arises when

there is no variable in $O = OprSchema(n)$ that models the designated role (*i.e.*, $RoleVar(O, R)$ is undefined).

Building on the above definition, we define satisfaction of a role restriction by a plan wedge. We distinguish two categories of satisfaction, *strong* or *weak*, corresponding to the cases where some descendant node does or does not directly satisfy the role restriction.

Definition 2 (Role Restriction: Wedge) *Let n be a node in a plan refinement structure with plan wedge* W_n. *Then* W_n weakly satisfies *the role restriction ρ iff no node in* $\mathrm{Desc}(n)$ *violates ρ. Also,* W_n strongly satisfies *ρ iff there is some node in* $\mathrm{Desc}(n)$ *that directly satisfies ρ, and no node that violates ρ.*

A node is said to *match* a generalized activity if its operator schema satisfies the feature requirements and its plan wedge satisfies the role restrictions for the activity. Intuitively, a generalized activity describes conditions for an actual operation; hence, the strong version of satisfaction for role restrictions is imposed (thus ensuring that the specified roles are defined and properly filled).

Definition 3 (Match: Feature Set) *An operator schema O* matches *a set of feature specifications $\mathcal{F} = \mathcal{F}^+ \cup \mathcal{F}^-$ iff $\mathcal{F}^+ \subseteq \mathrm{Features}(O)$ and $\mathcal{F}^- \cap \mathrm{Features}(O) = \emptyset$. A goal g* matches *$\mathcal{F}$ iff there is some operator schema $O \in \mathcal{O}$ that matches $\mathcal{F}$ for which* $\mathrm{Purpose}(O)$ *matches g.*

Definition 4 (Match: Generalized Activity) *A node n in a plan refinement structure* matches *a generalized activity $\langle \mathcal{F}^+ \cup \mathcal{F}^-, \mathcal{R} \rangle$ iff* $\mathrm{OprSchema}(n)$ *satisfies $\mathcal{F}^+ \cup \mathcal{F}^-$ and* W_n *strongly satisfies every role restriction in $\mathcal{R}$.*

Satisfaction of positive role advice by a plan structure node requires that either the node not match the target activity, or the advised role restrictions are satisfied by that node or one of its descendants. Here, we require only weak satisfaction for the advised role constraints, which better matches the intuitive character of role advice. For negative role advice, no descendant of a node matching the target activity should directly satisfy the role restriction.

Definition 5 (Role Advice) *A node n in a plan structure* weakly (strongly) satisfies $(\mathrm{ROLE}^+\ \rho\ \alpha_G)$ *iff either n does not match α_G, or* W_n *weakly (strongly) satisfies ρ. Also, n* satisfies $(\mathrm{ROLE}^-\ \rho\ \alpha_G)$ *iff n does not match α_G, or no $n' \in \mathrm{Desc}(n)$ directly satisfies ρ.*

The weak and strong versions of satisfaction for positive role advice are equivalent when the roles in the advice are necessarily defined for the wedge underneath the node matching the target activity. This condition often holds. For instance, consider the advice Use TWA for transatlantic flights. All plans for such flights will involve one or more operators that contain the roles Destination and Carrier. We refer to this condition of guaranteed existence in the expansion as *role comprehensiveness.*

Definition 6 (Role Comprehensiveness) *A role R is* comprehensive with respect to a goal *g iff every expansion of g contains a node n such that* $\mathrm{RoleVar}(\mathrm{OprSchema}(n), R)$ *is defined.*

Proposition 1 *Let n be a node in a plan refinement structure with plan wedge* W_n. *If role R is comprehensive with respect to* $\mathrm{Goal}(n)$ *then* W_n *weakly satisfies a role restriction ρ iff it strongly satisfies ρ.*

Role comprehensiveness guarantees the duality of $ROLE^-$ and $ROLE^+$, as captured by the following proposition. This result is significant in that it enables positive and negative role advice to be processed in a uniform manner when *role comprehensiveness* holds.

Proposition 2 *If role R is comprehensive with respect to goals that match the feature set of α_G then* $(\mathrm{ROLE}^-\ \langle R, F[x] \rangle\ \alpha_G)$ *is satisfied precisely when* $(\mathrm{ROLE}^+\ \langle R, \overline{F}[x] \rangle\ \alpha_G)$ *is satisfied.*

Satisfaction for negative method advice is straightforward: it requires that no descendant of a node matching the target activity matches the advised activity.

Definition 7 (Method Advice: Negative) *A plan structure node n* satisfies $(\mathrm{METHOD}^-\ \alpha_M\ \alpha_G)$ *iff n does not match α_G, or there is no planning node $n' \in \mathrm{Desc}(n)$ that matches α_M.*

Satisfaction for positive method advice presents a more complex case. Consider an approach that requires only that any node matching the target activity have some descendant node that matches the advised activity. Such a semantics is unsuitable for advice such as Fly between locations further than 200 miles apart; given this advice, a traveler would be unhappy with a plan in which he or she did not fly both legs of a trip from Boston through Chicago to Seattle. At the other extreme, the advice should not necessarily apply 'everywhere': consider a trip that includes a destination D that is both inaccessible by air and more than 200 miles from any other location to be visited. In this case, it should be acceptable to generate a plan that involves driving to D.

We adopt the middle ground for satisfaction of positive method advice: given a node that matches the target activity, it requires both the existence of some descendant node that matches the advised activity and a restriction that operators matching the advised activity are applied to the 'maximum extent possible'. We believe that this definition best matches user intent for method advice.

Definition 8 (Method Advice: Positive) *A plan structure node* n satisfies (METHOD$^+$ α_M α_G) *iff* n *does not match* α_G*, or the following conditions hold:*

- *there is some node* $n' \in$ Desc(n) *that matches* α_M (Existence)*,*
- *no descendant of* n *that does not match* α_M *could be replaced by a node that does* (Prescription)*.*

It is not the case that a node satisfies ($METHOD^-$ α_M α_G) iff it does not satisfy ($METHOD^+$ α_M α_G). For example, a node can fail to satisfy the positive version in situations where there are multiple descendant nodes for which an operator matching the advised activity could be applied, and some but not all are used. In this case, the negative version of the advice is also unsatisfied.

Advice satisfaction for a plan refinement structure is defined as follows.

Definition 9 (Advice Satisfaction) *A plan structure node satisfies the set of advice* $\mathcal{A}$ *iff it satisfies each piece of advice in* $\mathcal{A}$*. A plan structure* π *satisfies* $\mathcal{A}$ *iff each node in* π *satisfies* $\mathcal{A}$*.*

We note that our definitions of satisfaction for both role and method advice are *nondirectional* in that the conditions of the target activity and the role or method prescriptions mutually constrain each other. For example, consider the advice Don't drive in cities with more than 300,000 people. In a partial itinerary that includes a visit to a city with more than 300,000 people, this advice prohibits the use of a personal automobile for local transport. In addition, it would restrict the choice of cities to visit for time periods in which a rental car has already been secured.

7 Advice Enforcement

Strategic advice acts as a filter on the set of solutions to a planning problem. As such, it is straightforward to define an algorithm that generates advice-satisfying plans: use a Generate-and-Test scheme with a systematic plan-generation engine and a filter that validates the advice satisfaction conditions. Such an approach is impractical for nontrivial domains because of the size of the underlying search space. Instead, an algorithm is required in which advice *informs* the search process. The challenge is to define techniques in which advice influences local planning decisions (the choices for variable instantiations, operator selections, *etc.*), in order to minimize backtracking that results from unsatisfied advice. Such backtracking will necessarily arise: since the definition of advice satisfaction refers to a completed plan refinement structure, planning choices must be made before their correctness with respect to advice satisfaction can be determined.

Here, we define an HTN-style planning algorithm called PSA *(Plan-generation with Strategic Advice)* that enforces advice. The PSA algorithm is proven *sound* in that it produces only plans that satisfy strategic advice, and *complete* in that it will find such a plan if one exists (given certain assumptions).

In essence, PSA works by adding *advice constraints* to nodes in a task network. Advice constraints are determined by considering the operator choices and world state for an HTN node, along with the context of the current partial plan refinement structure. Advice constraints are operationalized to a combination of restrictions on the use of operators, and *planning constraints* formulated exclusively in the language of the basic domain. With this reduction, advice can be enforced using the operator applicability and constraint testing procedures of the underlying planner. This approach makes it possible for advice processing to be layered on top of the core planning system, yielding a modular and portable implementation. Furthermore, the PSA algorithm can be readily adapted to any operator-based goal-refinement framework, including both generative planners and reactive plan execution systems (such as PRS [12]).

The remainder of this section describes PSA. It begins with the presentation of a high-level description of the HTN algorithm that underlies PSA. An overview of PSA is provided next, followed by a detailed description of the algorithm.

7.1 HTN Algorithm

Standard HTN planning can be characterized as a recursive algorithm on task networks. When the network contains only primitive tasks, the algorithm returns a *completion* of the task network if one is defined, else the algorithm fails. When the network contains nonprimitive nodes, one is expanded by selecting an operator schema that matches the unsolved goal on the node and whose preconditions are satisfiable, applying it to generate an expanded task network, then recursively invoking the algorithm on the new network.

Figure 2 outlines a traditional HTN algorithm (based loosely on [4]). It assumes the following standard planning capabilities. The function $Complete(\tau)$ produces a completion of the task network τ if one exists (*i.e.*, resolves all conflicts and instantiates variables). The function $SelectOpr(\mathcal{O}, n, \tau)$ nondeterministically chooses an instantiation of an operator schema from the set $\mathcal{O}$ that can be applied to the goal for node n in task network τ. This function first identifies the schemas in $\mathcal{O}$ whose purpose matches the goal for n, using the function $Match(g_1, g_2)$. Any one of those schemas whose preconditions are satisfied for the relevant bindings, as computed by $Satisfied(Preconds(O)^\sigma, n, \tau)$, can be returned. The function $Reduce(n, O^\sigma, \tau)$ refines a task network by ex-

$Solve(\mathcal{P} = \langle \mathcal{O}, S_0, \tau_0 \rangle, \tau = \langle T, L, W \rangle)$
  If τ is primitive
    Return $Complete(\tau)$ if defined,
    Else FAIL
  Else for a nondeterministically selected nonprimitive node $n \in T$
    `/* Nondeterministically select an applicable operator */`
    $O^\sigma \leftarrow SelectOpr(\mathcal{O}, n, \tau)$
    `/* Apply the selected operator */`
    $\tau' \leftarrow Solve(\mathcal{P}, Reduce(n, O^\sigma, \tau))$
    Return τ'

$SelectOpr(\mathcal{O}, n, \tau)$
  $\Phi_1 \leftarrow \{\langle O_i, \sigma_i \rangle \mid O_i \in \mathcal{O}, \sigma_i = Match(Goal(n), Purpose(O_i))$ and $\sigma_i \neq \perp\}$
  $\Phi_2 \leftarrow \{\langle O_i, \sigma_i' \rangle \mid \langle O_i, \sigma_i \rangle \in \Phi_1, \sigma_i' = Satisfied(Preconds(O_i)^{\sigma_i}, n, \tau)$ and $\sigma_i' \neq \perp\}$
  Return ${O_i}^{\sigma_i'}$ for some nondeterministically chosen $\langle O_i, \sigma_i' \rangle \in \Phi_2$

Figure 2: Traditional HTN Algorithm

panding a node n by the (possibly partially) instantiated plot of the operator O, adding ordering and world constraints as appropriate.

7.2 The PSA Algorithm

The PSA algorithm, presented in Figures 3 and 4, extends the standard HTN algorithm in a few key ways to support advice. These extensions consist of a modified operator selection process, validation of positive method advice for plan wedges, and the propagation of advice-processing information across refinement levels.

7.2.1 Overview

Before discussing the algorithm in detail, we consider the stages that a given piece of advice passes through, namely *activation* and *enforcement.*

The definition of advice satisfaction by a plan refinement structure requires that each piece of advice be satisfied by every node in the structure. For a given node, many pieces of advice are trivially satisfied in the sense that they do not match the context specified by the target activity of the advice. We say that a piece of advice is *triggered* by a planning node when it matches the target activity feature set for the advice.

Triggered advice is generally not directly enforceable; matches for the advised feature set (for method advice) and all roles must be found before any actions can be taken. Such matching may involve operator applications distributed over multiple refinement (and abstraction) levels. We say that a piece of triggered advice has been *fully-activated* when all such matches have been made; otherwise, we say that the advice is *partially-activated.*

Enforcement of fully-activated advice draws on a combination of techniques. For role advice, enforcement reduces to the addition of planning constraints to the applicability conditions of operators. Method advice additionally restricts the set of candidate operators for a goal based on information about operator features. Finally, positive method advice requires a post-planning verification of the *existence* condition.

Because both the matching required for activation and the resultant enforcement constraints can span multiple refinement levels, PSA employs an approach in which each node in an HTN has an associated set of partially-activated advice and advice constraints. This information is propagated downward in the plan refinement structure as appropriate.

7.2.2 Notation

The PSA algorithm uses the following notation. ϕ_a^A denotes a formula representing the advised role constraints for the advice a; if there are no such constraints, ϕ_a^A is simply TRUE. ϕ_a^T is defined similarly but for the target role constraints. In cases where the advice is unambiguous, we simply write ϕ^A and ϕ^T. Role constraint formulas are defined in terms of a particular role, which will map to some variable in a planning operator. The process of identifying this planning variable is referred to as *role resolution.* We use a placeholder variable labeled by the advice and the type of role (target or advised) to represent the planning variable within unresolved role constraint formulas, *e.g.*, $\phi_a^A[x_a^A]$.

The functions $Features_A(a)$ and $Features_T(a)$ return the advised and target feature set, respectively, of the advice a. The functions $MethodAdvice^+(I)$, $MethodAdvice^-(I)$, $RoleAdvice(I)$ take a set of activated (either fully or partially) advice I and return the subsets of positive method, negative method, and role advice, respectively.

$SolveAP(\mathcal{AP} = \langle \mathcal{O}, S_0, \tau_0, \mathcal{A} \rangle, \tau = \langle T, L, W \rangle)$
  If τ is primitive
    Return $Complete(\tau)$ if defined,
    Else FAIL
  Else for a nondeterministically selected nonprimitive node $N = \langle n, I, C \rangle \in T$
    `/* Nondeterministically select an applicable operator with its Advice constraints */`
    $\langle O^\sigma, I', C' \rangle \leftarrow SelectOpr_A(\mathcal{O}, N, \tau, \mathcal{A})$
    `/* Apply the selected operator and pass along advice info */`
    $\tau' \leftarrow SolveAP(\mathcal{AP}, Reduce_A(\langle n, I', C' \rangle, O^\sigma, \tau)$
    If O^σ and τ' are not well-defined then FAIL
    `/* Test Method constraints for this wedge */`
    If for each $\alpha \in MethodAdvice^+(I' - I)$
      there is some $n' \in Desc(n)$ for which $W_{n'}$ matches α
        Return τ'
        Else FAIL

Figure 3: Algorithm for Plan-generation with Strategic Advice (PSA)

7.2.3 PSA Details

PSA diverges from standard HTN planning most significantly in the operator selection process. As defined in Figure 4, the function $SelectOpr_A(\mathcal{O}, \langle n, I, C \rangle, \tau, \mathcal{A})$ extends $SelectOpr(\mathcal{O}, n, \tau)$ both to relativize operator selection and application to triggered advice and to propagate relevant activated advice and advice constraints. $SelectOpr_A$ begins likes its nonadvice counterpart, filtering those operators whose purpose does not match the goal of the node under consideration. It also ends similarly, selecting an unfiltered operator whose applicability conditions are satisfied; however, those conditions have been extended to incorporate additional constraints generated by advice processing. The bulk of the function involves this advice processing for the operators that match the goal of the current node: *advice triggering*, *extraction of advice constraints*, and *role resolution*.

Advice processing begins by identifying for each relevant operator O_i any advice that it triggers. Triggering amounts to matching the target activity feature set, as computed by the test $HasFeatures(O_i, Features_T(a))$.

The extraction of advice constraints differs for the cases of role advice, positive method advice, and negative method advice. For role advice, the extracted advice constraint is simply the disjunction $\overline{\phi^T} \vee \phi^A$. Method advice presents a more complex case because it posits higher-order constraints defined over operators and domain objects. As a result, it is not possible to map them to planning constraints in $\mathcal{L}$ that can be directly processed by the underlying planner (as is the case with role advice). Instead, constraints defined over a mixture of operators and domain objects must be managed explicitly at a higher level. We adopt a *case analysis* approach in which satisfaction is considered in turn for the different kinds of constraints (on operators and on domain objects).

Consider first negative method advice. Any operator that does not match the feature set of the advised activity requires no additional constraints. For an operator that does match, either the target role-fill constraints ϕ^T or the advised role-fill constraints ϕ^A must fail. Thus, $\overline{\phi^T} \vee \overline{\phi^A}$ is added to the applicability conditions of the operator.

Positive method advice requires a different approach: either the target activity must not match, or the existence and prescription clauses must both be satisfied. For an operator that matches the feature set of the advised activity, we require that the advised role constraints ϕ^A be satisfied. For an operator that does not match the advised feature set, if there is *some* unfiltered operator that does match, then the target activity must not match. The constraint $\overline{\phi^T}$ guarantees this condition.[2]

Advice constraints are defined over roles; before they can be enforced, the roles must be resolved. The function $ResolveRoles(O, \Delta)$ performs role resolution; it replaces any role variable in the set of constraints Δ with the variable that models that role in the operator schema (if one exists). Resolved constraints (computed by $Resolved(C)$) are added to the applicability constraints of the associated operator. Unresolved constraints (computed by $Unresolved(C)$) are passed to all descendant nodes.

The actual propagation of advice information is straightforward; $Reduce(n, O, \tau)$ is generalized to the

[2] This approach to enforcing positive method advice is overly conservative in that the added constraints may be unnecessary (because some other node in the plan satisfies the *existence* clause) yet they block application of the operator. We return to this point in the discussion of completeness below.

$SelectOpr_A(\mathcal{O}, \langle n, I, C\rangle, \tau, \mathcal{A})$
$\Phi_1 \leftarrow \{\langle O_i, \sigma_i\rangle \mid O_i \in \mathcal{O}, \sigma_i = Match(Goal(n), Purpose(O_i))$ and $\sigma_i \neq \perp\}$

/* Accumulate triggered advice for each candidate operator */
For $\langle O_i, \sigma_i\rangle \in \Phi_1$
$I_i \leftarrow \{a \mid a \in \mathcal{A}$ and $HasFeatures(O_i, Features_T(a))\}$
$I' \leftarrow \Sigma I_i$

/* Extract advice constraints */
For $\langle O_i, \sigma_i\rangle \in \Phi_1$ /* Initialize advice constraints */
$R_i \leftarrow \{\}$, $M_i^+ \leftarrow \{\}$, $M_i^- \leftarrow \{\}$

For $a \in RoleAdvice(I_i)$ /* Extract constraints from role advice */
$R_i \leftarrow R_i \cup \{\overline{\phi_a^T[x_a^T]} \vee \phi_a^A[x_a^A]\}$

For $m \in MethodAdvice^-(I \cup I')$ /* Extract constraints from negative method advice */
$Sat \leftarrow \{\langle O_i, \sigma_i\rangle \mid \langle O_i, \sigma_i\rangle \in \Phi_1$ and $HasFeatures(O_i, Features_A(m))\}$
For $\langle O_i, \sigma_i\rangle \in Sat$, $M_i^- \leftarrow M_i^- \cup \{\overline{\phi_m^A[x_m^A]} \vee \overline{\phi_m^T[x_m^T]}\}$

For $m \in MethodAdvice^+(I \cup I')$ /* Extract constraints from positive method advice */
$Sat \leftarrow \{\langle O_i, \sigma_i\rangle \in \Phi_1 \mid \langle O_i, \sigma_i\rangle \in \Phi_1$ and $HasFeatures(O, Features_A(m)\}$
If $Sat \neq \{\}$:
For $\langle O_i, \sigma_i\rangle \in Sat$, $M_i^+ \leftarrow M_i^+ \cup \{\phi_m^A[x_m^A]\}$
For $\langle O_i, \sigma_i\rangle \in \Phi_1 - Sat$, $M_i^+ \leftarrow M_i^+ \cup \{\overline{\phi_m^T[x_m^T]}\}$

/* Collect resolved advice constraints for each Operator */
For $\langle O_i, \sigma_i\rangle \in \Phi_1$
$ResolveRoles(O_i, C \cup R_i \cup M_i^+ \cup M_i^-)$
$C_i \leftarrow Unresolved(C \cup R_i \cup M_i^+ \cup M_i^-)$
$Add_i \leftarrow Resolved(C \cup R_i \cup M_i^+ \cup M_i^-)$

/* Return an operator that satisfies advice constraints along with updated advice info */
$\Phi_2 \leftarrow \{\langle O_i, \sigma_i'\rangle \mid \langle O_i, \sigma_i\rangle \in \Phi_1, \sigma_i' = Satisfied(Preconds(O_i)^{\sigma_i} \cup Add_i^{\sigma_i}, n, \tau)$ and $\sigma_i' \neq \perp\}$
Return $\langle {O_i}^{\sigma_i'}, I \cup I_i, C_i\rangle$ for some nondeterministically chosen $\langle O_i, \sigma_i'\rangle \in \Phi_2$

Figure 4: Operator Selection for PSA

function $Reduce_A(\langle n, I, C\rangle, O, \tau)$ to pass along advice constraints and activated advice.

As noted above, the existence clause for satisfaction of positive method advice cannot be verified until the wedge beneath the node has been completed. An explicit check of this condition is included as the last step in the PSA algorithm. Since the validation of the condition is straightforward, the details are omitted.

7.3 Discussion

Our presentation of PSA has been biased toward highlighting the reuse of standard HTN operations. For reasons of efficiency, the PSA algorithm should not be implemented directly as presented here. For instance, extraction of role constraints and resolution of role variables should be done only for the selected operator, rather than for all operators matching the current goal prior to the final selection decision. In contrast, extraction of method constraints should be done prior to the selection process because the method constraints can eliminate certain operators up front.

There is a similarity between the kind of *constraint-augmented planning* embodied in the PSA algorithm and Constraint Logic Programming (CLP) [7]: both consist of a problem-reduction search augmented with constraints on the overall structure being defined. For CLP, the constraints restrict instantiations for variables; for advisable planning, the constraints further restrict the choice of problem-reduction rules (*i.e.*, the operator schemas).

7.4 Properties of PSA

It is straightforward to show that the PSA algorithm reduces to standard HTN planning when there is no

advice. The following correctness result also holds.

Proposition 3 (Correctness of PSA) *Application of the PSA algorithm to an advised planning problem* $\mathcal{AP} = \langle \mathcal{O}, S_0, \tau_0, \mathcal{A} \rangle$ *will produce a plan refinement structure that both satisfies* $\mathcal{A}$ *and is a solution to the planning problem* $\langle \mathcal{O}, S_0, \tau_0 \rangle$.

We note that given *role comprehensiveness*, the strong version of satisfaction for role advice follows; otherwise, only weak satisfaction holds. The difference results because PSA folds in constraints extracted from role advice only when the roles for the constraints have been resolved; thus, if the roles are not found, the associated role advice is effectively ignored.

The PSA algorithm may fail to find solutions in certain cases where solutions exist. This incompleteness stems from our conservative approach to enforcing the prescription clause for positive method advice. In particular, the constraints added to enforce this clause are too strong in situations when a node matching the target activity has multiple descendant nodes for which constraints may be added to guarantee matching to the advised activity. As an illustration, consider a situation where there are two descendant nodes to which operators could be applied that match the advised feature set. PSA would force for both nodes the selection of operators that satisfy the advised feature set, even in cases where the use of the operators is mutually inconsistent (for instance, they each may require a consumable resource of which there is only one remaining). However, the definition of satisfaction for positive method advice requires that both be selected *only if it is consistent to do so*.

We can show that the PSA algorithm is complete when there is at most one node for which constraints are added to enforce the prescription clause; we refer to this condition as the *Uniqueness of the Advised Activity (UAA)* for a given piece of positive method advice relative to a set of operators. The UAA condition holds frequently in practice. For instance, in planning a holiday there is generally a single high-level vacation type to select (*e.g.*, bike tour *vs* camping tour *vs* driving tour). Thus, advice such as Use a bike tour for the vacation in California would satisfy UAA, since the choice of tour type would only be made once. Furthermore, the UAA condition is easily verified by straightforward syntactic analysis of operator schemas. However, one can formulate natural problems for which UAA is violated. For instance, one could formulate a multi-phase holiday operator that encompasses several sub-vacations, each of which would require a choice of vacation type.

Proposition 4 (Completeness of PSA) *Application of the PSA algorithm to an advised planning problem* $\mathcal{AP} = \langle \mathcal{O}, S_0, \tau_0, \mathcal{A} \rangle$ *for which UAA holds will produce a plan refinement structure that both satisfies* $\mathcal{A}$ *and is a solution to the planning problem* $\langle \mathcal{O}, S_0, \tau_0 \rangle$, *provided such a plan exists.*

8 Related Work

Until recently, there had been few attempts to develop domain-independent advice-taking systems. Concerns for the usability of AI systems and problems with knowledge acquisition have prompted a marked increase in activity in this area during the past few years.

The TRAINS [5] project seeks to provide users with the means to interactively guide the construction and execution of a plan through a cooperative, mixed-initiative effort. While its overall objectives are similar to ours, the research directions of the two efforts are highly complementary. HCI issues are a major focus for the TRAINS project (*e.g.,* language processing, dialog management, multi-media), while our work emphasizes the identification of advice idioms and corresponding enforcement algorithms.

The *reactive scheduling* model of the DITOPS system [13] also has much in common with our work. At a high level, its *spread-sheet* metaphor provides a good characterization of advisability: the user should be able to specify characteristics of the desired solution and have the system sort out the details. One key difference is that DITOPS focuses on scheduling rather than planning. In addition, it emphasizes changes to domain constraints, while the our work focuses on advice that describes high-level properties of solutions.

The TRAINS and DITOPS projects are similar to our work in that they emphasize *product-related* advice that describes characteristics of the desired end-product. In contrast, *performance-related* advice encodes base-level problem-solving expertise. As such, product-related advice serves as an adjunct to the underlying problem-solving knowledge, while performance advice amounts to an extension of it.

The original work on performance advice is Mostow's [11], which models advice as a high-level description of a task for which there is no explicit method to achieve it. The work focuses on *operationalization* – the transformation of advice (using sound and heuristic methods) into directives that can be executed directly by the problem-solving system being advised. Its notion of advice-taking amounts to 'filling in' gaps in planning operators, in contrast to our approach of restricting how operators should be instantiated and applied.

There has been a recent surge of interest in improving the performance of reinforcement learners through user guidance (see [9] for a good overview). Broadly speaking, that work focuses on the provision of additional domain knowledge to improve the overall performance of a system. Many of these efforts are actually a form of 'programming by example'. However, certain

of them have more of a flavour of performance-related advice-taking, using general-purpose languages to encode the additional domain knowledge [9, 6]. One interesting feature of these efforts is that advice is refined during problem-solving, in response to the success with which it has been applied previously.

9 Conclusions

The notion of problem-solving systems that can take advice from humans has been around since the start of AI [10]. Despite the conceptual appeal, there has been little success to date in building automated advice-taking systems because of the intractability of the task in its most general form. We believe that the paradigm can be made tractable for specific classes of applications and tasks by grounding advice in a focused set of problem-solving activities. The research described here presents a step toward this goal for the paradigm of generative planning. Its primary contributions are the presentation of an advice-taking framework for AI planning systems, the formalization of strategic advice, and the definition of a sound and complete HTN planning algorithm for enforcing strategic advice.

We have built an initial Advisable Planner prototype that implements our theory of strategic advice. The system consists of an advice manager layered on top of SIPE–2, a mature HTN planner [15, 16]. Intrusions into the underlying code were minor: for the most part, the advice processing is completely separate from the core planning capabilities. Our prototype has been used to enforce both travel planning advice similar to the examples presented in this paper and advice for a crisis-action planning domain [17]. Currently, we are applying the system to air-campaign planning, with the goal of encouraging nonexperts to embrace AI planning technology.

Immediate next steps for this work are to develop richer idioms for strategic advice, and to define comparable formal models and enforcement algorithms for other categories of advice (including evaluational and task advice). Further down the road, we intend to explore utility models for partial satisfaction of advice that will allow intelligent trade-offs to be made among sets of conflicting advice.

Acknowledgements

This research has been supported by DARPA Contract F30602-95-C-0259. The author would like to thank David Wilkins for providing insight into the workings of SIPE–2.

References

[1] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[2] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 32(1), 1991.

[3] B. Drabble and A. Tate. O-Plan mixed initiative planning capabilities and protocols. Technical Report 24, University of Edinburgh, 1995.

[4] K. Erol, J. Hendler, and D. S. Nau. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland, 1994.

[5] G. Ferguson, J. Allen, and B. Miller. TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third International Conference on AI Planning Systems*. AAAI Press, 1996.

[6] D. Gordon and D. Subramanian. A multistrategy learning scheme for agent knowledge acquisition. *Informatica*, 17:331–346, 1994.

[7] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *ACM Symposium on Principles of Programming Languages*, 1987.

[8] S. Kambhampati and J. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2):192–258, 1992.

[9] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–282, 1996.

[10] J. McCarthy. Programs with common sense. In *Symposium on the Mechanization of Thought Processes*, pages 77–84, 1958.

[11] D. J. Mostow. *Mechanical Transformation of Task Heuristics into Operational Procedures.* PhD thesis, Computer Science Dept., Carnegie-Mellon University, 1981.

[12] K. L. Myers. *User's Guide for the Procedural Reasoning System.* Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

[13] S. F. Smith and O. Lassila. Toward the development of flexible mixed-initiative scheduling tools. In M. H. Burstein, editor, *ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop Proceedings.* Morgan Kaufmann, February 1994.

[14] M. Stefik. Planning and meta-planning. *Artificial Intelligence*, 16(2), 1981.

[15] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.

[16] D. E. Wilkins. *Using the SIPE-2 Planning System: A Manual for Version 4.3.* Artificial Intelligence Center, Menlo Park, CA, August 1993.

[17] D. E. Wilkins and R. V. Desimone. Applying an AI planner to military operations planning. In M. Fox and M. Zweben, editors, *Intelligent Scheduling.* Morgan Kaufmann, 1994.