Planning with Conflicting Advice

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

Abstract

The paradigm of *advisable planning*, in which a user provides guidance to influence the content of solutions produced by an underlying planning system, holds much promise for improved usability of planning technology. The success of this approach, however, requires that a planner respond appropriately when presented with conflicting advice. This paper introduces two contrasting methods for planning with conflicting advice, suited to different user requirements. *Soft enforcement* embodies a heuristic approach that prefers planning choices that are consistent with specified advice but will disregard advice that introduces conflicts. Soft enforcement enables rapid generation of solutions but with suboptimal results. *Local maxima search* navigates through the space of advice subsets, using strict enforcement techniques to identify satisfiable subsets of advice. As more time is allocated, the search will yield increasingly better results. The paper presents specific algorithms for soft enforcement and local maxima search, along with experimental results that illustrate their relative strengths and weaknesses in trading computation time for advice satisfaction.

Introduction

Research in AI planning has focused primarily on fully automated techniques for generating plans that satisfy user goals. In recent years, however, there has been a growing recognition that many users are reluctant to relinquish full control to an automated system. As a result, there has been increased interest in paradigms that explicitly draw the user into the planning process (Ferguson & Allen 1998; Smith, Lassila, & Becker 1996; Burstein & McDermott 1994; Tate, Dalton, & Levine 1998).

In previous work, we introduced an approach in which users can *advise* an automated planning system in order to influence the content of the solutions that it produces (Myers 1996). Advice consists of task-specific constraints on both the desired solution and the refinement decisions that underlie the planning process. Advice is specified in a high-level language designed to be natural and intuitive for users, and then operationalized into constraints that direct the underlying planning technology during plan construction. As such, advice enables users to guide the planning process, with the planning system performing the time-consuming work of filling in low-level details and detecting potential problems.

(Myers 1996) presented a formal language for *strategic advice*, which supports recommendations on how goals and actions are to be accomplished, along with a model of *strict satisfaction* that interprets advice as hard constraints. That paper included an algorithm for plan generation with advice enforcement that is both sound and complete (under well-specified conditions) for the strict satisfaction model. While developed originally for hierarchical task network (HTN) planning (Erol, Hendler, & Nau 1994), the models apply more broadly to any hierarchical planning framework.

With strict enforcement, no solution is returned in the event that the full set of specified advice cannot be satisfied. In general, however, users may specify advice that is not satisfiable within the limits of the problem domain. For this reason, an advice enforcement mechanism is needed that will behave sensibly in the face of conflicting advice.

Two characteristics make planning with conflicting advice difficult. First, the recognition of conflicts is an intractable problem: in the worst case, the unsatisfiability of one or more pieces of advice will require exhaustive search through the space of possible solutions. Second, current plan generation frameworks treat constraints as hard, making it difficult to support relaxation of constraints derived from advice.

This paper introduces two complementary approaches for planning with conflicting advice, grounded in a model of *partial satisfaction* for a set of advice. Given the intractable nature of the problem in the general case, we focus on methods at opposite ends of the time/quality tradeoff. The first, *minimize introduced local advice violations (MILAV)*, belongs to the class of *soft enforcement* methods which are guaranteed to produce a plan but may ignore advice that could be satisfiable. MILAV performs a limited amount of analysis at each task node to rank planning choices by the number of advice violations that each would introduce. The second, *local maxima search*, is rooted in strict enforcement methods. It employs directed search through the space of subsets of the user-specified advice to identify increasingly

larger sets of satisfiable advice, along with a variant of simulated annealing to enable discovery of solutions from different portions of the search space. The local maxima search method embodies an *anytime* flavor, yielding increasingly better results when allocated additional time.

The paper begins with a review of strategic advice, covering both its definition and the corresponding model for strict satisfaction. Models for partial satisfaction of advice are then discussed, along with issues related to requirements for advice relaxation methods. Next, the soft enforcement and local maxima search methods are defined and experimental results presented that validate their effectiveness. These methods have been implemented and evaluated within an Advisable Planner prototype (Myers 1999) built on the SIPE–2 planner (Wilkins 1988).

Strategic Advice

Strategic advice expresses recommendations on how tasks are to be accomplished, in terms of 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 further constrains the type of approach used. Strategic advice designates not only properties of the resultant partially ordered set of actions generally viewed as "the plan", but also the underlying justifications for that solution.

Strategic advice is defined in terms of a *domain metatheory*. A standard planning domain is modeled in terms of three basic types of element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world and individual world states, and *operators* that describe ways to achieve tasks. The domain metatheory provides an abstracted characterization of the planning domain that specifies key semantic differences among operators, planning variables, and instances. This abstraction provides users with the means to describe desired solution characteristics in natural, semantically meaningful terms. The metatheory is built around two main constructs: *features* and *roles*.

A *feature* designates an attribute of interest for an operator that distinguishes it from other operators that could be applied to the same task. For example, operators that can be used to refine tasks of moving from location X to location Y may involve travel by air, land, or water; each of these media could be modeled as a feature. Because there can be multiple operators that apply to a particular task, features provide a way of abstracting from the details of an operator to distinguishing attributes that might be of interest to users. Note that features differ from operator preconditions in that they do not directly restrict use of operators by the planner.

A *role* corresponds to a capacity in which a domain object is to be used within an operator. Roles map to individual variables within a planning operator. For instance, an air transportation operator within a travel domain could have variables `location.1` and `location.2` that fill the roles of Origin and Destination, as well as a variable `airline.1` that fills the role of Carrier. A comparable sea transportation operator may have these same roles, although with the planning variable `cruise-ship.1` for the role Carrier.

Strategic advice is formulated in terms of *role-fills* and *activities*. *Activities* constitute abstract tasks relative to the underlying planning domain, and are defined in terms of *features* and *roles*. A given activity can have multiple features; for example, an activity corresponding to a bike tour could have the features Vacation, Bike, and Inexpensive. *Role-fills* are specifications of objects to be used in certain roles; they may name explicit individuals, or declare a set of required and prohibited attributes.

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 <context-activity>`. In general, role advice consists of one or more *role-fill* specifications, a *context 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 context activity is defined as tasks with feature Vacation, and with role Location filled by Scotland. The advice dictates that the filler for the role Accommodation be an object that belongs to the class 3-star-hotel and have ensuite facilities as an attribute.

Method Advice

Method advice imposes restrictions on the approaches that can be used in solving a class of goals. It is characterized by the template: `<Use/Don't use> <advised-activity> for <context-activity>`. Thus, method advice consists of context 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.

Strict Satisfaction Model

The model of *strict satisfaction* of strategic advice is grounded in the overall problem-solving context in which planning decisions are made, rather than being restricted to the final set of actions that results. To see why, 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 would consist of a set of actions at the level of movements to destinations, stays in accommodations, and tours of various sights. 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.

The formal definition of satisfaction for strategic advice can be found in (Myers 1996); we provide an informal summary here. Strict advice satisfaction builds on the concepts of *satisfaction* for role-fills and *matches* for activities by a *plan wedge* (i.e., a node and all of its descendants). A given plan wedge satisfies a role-fill iff for each node in the plan wedge where the role is *resolved*, the associated role constraint is satisfied. A role is resolved iff the operator applied to that node contains a variable that has been declared in the domain metatheory to fill that role. A plan wedge matches the specification of an *activity* iff the operator applied to the node at the root of the wedge has the features specified by the activity, and the role-fills for the activity are satisfied.

For a piece of positive role advice, satisfaction requires that for any plan node that matches the *context-activity* for that advice, every descendant node satisfies the role-fill constraints in the advice. For negative role advice, the negation of the role-fill constraints must be satisfied. A node that matches the context-activity for a piece of advice is called a *trigger node* for the advice.

The semantics for method advice differ from those for role advice. In the negative polarity case, for any planning node that matches the context activity, there should be no descendant node that matches the advised activity. Positive method advice requires that for any plan node that matches the advice *context-activity*, the conditions of *existence* and *prescription* hold. Existence requires that there be some descendant node that matches the *advised activity* of the advice (i.e., *do it at least once*). Prescription requires that there be no descendant node that both does not match the *advised activity* and could be replaced by one that does (i.e., *do it whenever possible*).

Advice Relaxation Models

Partial satisfaction of an advice set can be defined at different levels of granularity. An *advice-level* model would be grounded in the satisfaction of an individual piece of advice, encompassing the full set of constraints that it introduces. As such, an advice-level model treats advice as a non-decomposable unit. In contrast, a *constraint-level* model would be grounded in the satisfaction of the constituent constraints that comprise individual pieces of advice. With such models, satisfaction of a subset of the constraints for a piece of advice is meaningful. As such, the constraint-level models support partial satisfaction for individual pieces of advice.

The strict satisfaction model for strategic advice combines both domain-level constraints and restrictions on operator selection, resulting in a heterogeneous space of decision points. This heterogeneity makes it difficult to define a constraint-level model of partial satisfaction for advice that is both conceptually appealing and readily implementable. For this reason, we adopt an advice-level model. In particular, we seek to maximize advice satisfaction in accord with the following definition.

Definition 1 (Maximal Advice Satisfaction) *Let* A *be a set of advice for a planning problem* P. *A subset* $A' \subseteq A$ *is* maximally satisfiable *for* A *and* P *iff there is some solution to* P *that satisfies* A' *but no solution that satisfies some* A'' *such that* $A' \subset A'' \subseteq A$. *A solution to* P *that satisfies a maximally satisfiable advice set is called an* advice maximal plan.

Ideally, a framework for advice relaxation would produce a set of advice maximal plans that covers the space of maximal advice subsets. However, the time required to identify such solutions can be prohibitive. An advice set of size n defines 2^n advice subsets, each of which is potentially advice maximal. In the worst case, exhaustive search would be $O(2^n * m)$, where m is an upper bound on the cost of generating a solution for a given set of advice. Although one would expect to improve upon the worst-case behavior in practice, exhaustive approaches will be impractical for many applications.

When dropping the requirement for a complete set of advice maximal plans, the strategy to adopt will vary with the needs of the user and the complexity of the problem domain. Certain users may be content with a quick-and-dirty solution, which they could subsequently improve by modifying advice or making plan changes directly. Others may prefer to invest more time to produce a single, high-quality solution. Still others may seek a range of solutions that satisfy different subsets of the specified advice. The *soft enforcement* method described below addresses the quick-and-dirty case, while *local maxima search* provides a framework for the remainder.

Soft Enforcement

Soft enforcement methods are heuristic in nature, treating advice as preferences that can be disregarded when they conflict with the plan under development. Soft enforcement methods can produce plans quickly but will yield suboptimal results.

Different strategies can be adopted for soft enforcement that trade time for *quality* (measured in terms of amount of satisfied advice). At one extreme, advice could simply be ignored during solution generation: although this *naive approach* will guarantee a solution (if one exists) and incurs no overhead, the quality of the solution will be low. At the

other extreme, detailed analyses of the domain could be performed to detect and prevent conflicts. This type of approach would produce better results, although with a corresponding increase in solution generation time.

The MILAV Method

We have developed a soft enforcement approach that performs one-step lookahead for each task node as a way to reduce the amount of advice that is violated by a generated plan. This approach, *minimize introduced local advice violations (MILAV)*, considers the impact of all choices (i.e., operator selection, variable instantiations) for a given task node and ranks them according to the number of new advice violations that each will introduce. Search control exploits this information to select options that minimize the introduction of new advice violations at that node. This approach is heuristic in nature, in that these locally optimal decisions may not be globally optimal. In particular, the selection of an operator and set of variable instantiations that minimizes violations at a node may engender more violations at lower levels than would a nonminimal selection.

MILAV can be implemented as a variant of the strict enforcement method described in (Myers 1996). Strict enforcement dynamically adds constraints to task nodes that are derived from a combination of the advice and the problem-solving context. These constraints are built from the advised role-fill constraint ϕ^A and the context role-fill constraint ϕ^C. Advice with positive polarity produces constraints of the form

$$\overline{\phi^C} \vee \phi^A \quad (1)$$

while advice with negative polarity produces constraints of the form

$$\overline{\phi^C} \vee \overline{\phi^A} \quad (2)$$

The schemas (1) and (2) may be instantiated multiple times for a given piece of advice. For role advice, schemas of this type are instantiated for each trigger node for the advice, with the corresponding constraints being applied to all descendants for which the specified role-fills are resolved. For positive method advice, the schema is similarly instantiated for every trigger node, with the corresponding constraints being applied to all descendants for which both the specified role-fills are resolved and the feature recommendations of the method advice are violated. The case for negative method advice is similar, although the constraints apply to nodes for which the feature recommendations are satisfied.

To implement MILAV, we introduce a *tautology predicate* $TRUE(A_i)$ that the planner recognizes as satisfiable. Every instance of the advice constraint schemas (1) and (2) introduced by a piece of advice A_i is extended to include a disjunct of the form $TRUE(A_i)$. Similarly, for each operator choice on a given node that would violate a piece of advice A_i, a corresponding constraint $TRUE(A_i)$ is dynamically inserted. As constraints are combined, these tautology predicates become embedded arbitrarily; by decomposing formulas into disjuncts, we then employ a *reasoning by cases* strategy to disjunct satisfaction. A search control strategy is employed that prefers operator/disjunct combinations that minimize the number of new $TRUE(A_i)$ constraints that have to be satisfied, thus minimizing the advice that is violated by a given planning decision.

MILAV provides a simple mechanism for ensuring solutions in the face of conflicting advice (modulo the completeness restrictions of the underlying planner). In particular, every constraint that is added is guaranteed satisfiable by the inclusion of the $TRUE(A_i)$ disjuncts. However, it introduces two types of cost. First, the disjunctive constraints derived from the schemas (1) and (2) increase the overall search space for planning. Second, there is the cost of preprocessing all possible cases for a node (i.e., operator/disjunct combinations) to identify those that minimize introduced advice violations. In contrast, standard plan generation needs only to identify one applicable operator for a given task. Note, however, that the introduced advice constraints could either increase or decrease the "hardness" of the planning problem, depending on the underlying structure of the problem (Hogg, Huberman, & Williams 1996).

The MILAV strategy is biased towards solutions satisfying advice that applies to higher-level decisions in the planning process, given that those decisions precede low-level decisions in standard hierarchical planning. Since decisions at higher levels tend to be more significant, this attribute is generally desirable.

Evaluation of MILAV

To evaluate the effectiveness of MILAV, we conducted experiments in a travel domain that involved selecting itineraries, schedules, accommodations, modes of travel, and carriers for business and pleasure trips. Problems in this domain are weakly constrained, with the result that there is generally a broad range of possible solutions. This solution-rich characteristic makes the domain a good candidate for the use and evaluation of advice.

The experiments compared the naive approach and MILAV for varying-size advice sets selected randomly from a predefined library. The library contains 63 pieces of advice, focused primarily on selecting locations to visit, modes of transportation, and accommodations. This advice was developed by an individual not associated with the research reported in this paper (Mayer 1997).

Figure 1 compares the degree of advice satisfaction for the naive approach and MILAV over a range of advice set sizes. For each size, 30 sets of advice were selected randomly and a plan generated for each. As can be seen, the MILAV strategy significantly increases the amount of satisfied advice.[1]

The improvements from the MILAV method come at a cost. Figure 2 displays the solution generation times using

[1] It may seem surprising that the naive approach manages to satisfy so much advice. Numerous pieces of advice within the test library are highly contextualized, making it possible for the advice to be satisfied when the context doesn't arise during plan generation. For example, a piece of advice When renting a car, use Hertz will be satisfied trivially by a plan that does not engage a rental car.

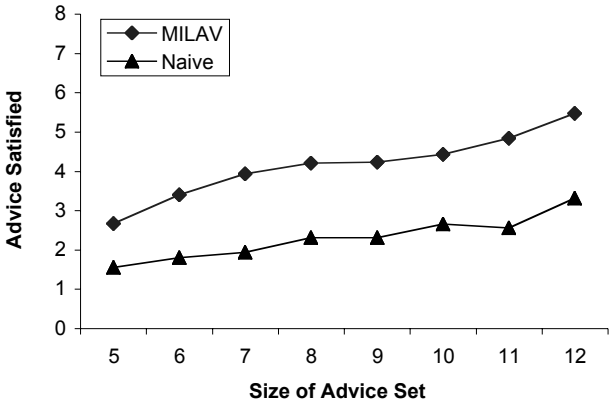

Figure 1: Advice Satisfaction: MILAV vs. Naive Soft Enforcement

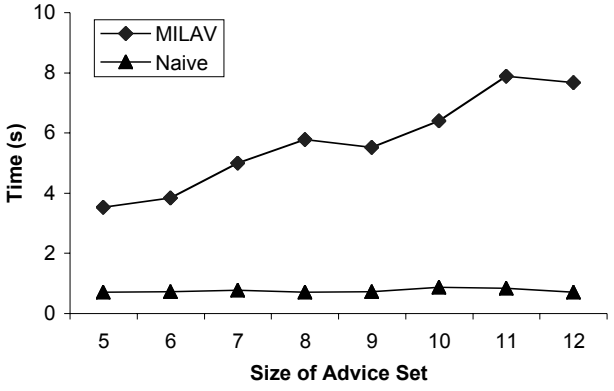

Figure 2: Solution Time: MILAV vs. Naive Soft Enforcement

MILAV and the naive approach for the 30 advice sets selected for each size. Because the naive approach disregards the advice, its solution times are independent of the size of the advice sets (approximately .75 second). MILAV requires a significant, but tolerable, increase in generation time, that increases roughly linearly with the size of the advice sets.

The absolute difference between the naive and MILAV solution times should be kept in perspective: because the underlying planning problems are highly unconstrained, the naive solution times are correspondingly low. More constrained problems should yield similar absolute differences in solution times between the two methods, but much smaller relative differences.

Local Maxima Search

The local maxima search (LMS) framework is built on navigation through the space of subsets of the user-specified advice, with strict enforcement applied in turn to generate plans for selected subsets.

The framework produces some initial (not necessarily good) *seed solution* that satisfies a selected subset of the user advice. It then iteratively improves on the current best solution by adding a single piece of advice and testing whether a plan can be generated for the expanded set under strict enforcement of advice. This depth-first expansion continues until every addition of a single piece of advice results in failure to generate a plan, indicating that the current node is a local maximum. At that point, the search "jumps" to a new seed node and begins again. Through this design, the LMS framework embodies an *anytime* flavor, yielding plans that satisfy increasing amounts of advice as more time is allotted. The cautious nature of this incremental search strategy, in which additional advice is considered one piece at a time, is important given the high cost of verifying the unsatisfiability of a set of advice for a given problem.

When trying to satisfy some designated advice subset, additional advice may be satisfied serendipitously. An *advice checker* capability enables determination of whether a given plan satisfies some arbitrary piece of advice. Such serendipitous satisfaction of advice can occur frequently given the cautious strategy of expanding the search by only one piece of advice at a time (rather than adopting a more aggressive approach of considering larger extensions of known satisfiable subsets).

Figure 3 presents the details of the LMS algorithm. The algorithm incrementally constructs a lattice L of nodes, each corresponding to a subset of the given advice A_{all}. The empty set lies at the top of the lattice, and A_{all} at the bottom. Each lattice node stores information about any plans that have been generated that satisfy the advice set for the node, any failed attempts to generate plans for the advice, and a corresponding *score* derived from the node's advice in the case where plans have been successfully generated. Links among parents and children in the lattice are also maintained.

The outer loop selects seed nodes through application of the function *GetNewSeed*(A_{all}, L). This seed selection function returns an *open* node in the lattice, meaning a node that has both an associated solution and at least one *candidate advice extension* for the node (defined below in the discussion of the inner loop). Exploration of new seeds terminates when there are no open nodes remaining in the advice lattice. Different seed selection strategies are possible, such as a best-first approach that would select among the open nodes with highest score. The next section presents an alternative approach based on *heuristic reseeding*.

The inner loop performs a hill-climbing search through the sublattice beneath a seed node. Given a node N for which a plan has been generated that satisfies the advice A, the function *SelectAddAdvice*(N, A_{all},L) selects a single piece of advice to extend A. That selected advice must satisfy the requirements that there not yet be a node in the lattice for $\{a\} \cup A$, or a successful node whose advice subsumes $\{a\} \cup A$. We say that such a piece of advice is a *candidate advice extension* for N. The inner loop continues until a node is reached that has no candidate advice extensions.

For the selected advice, an attempt is made to generate a plan using strict advice enforcement. If the attempt fails,

then a node for that advice is inserted into the lattice to record the failure. If the attempt succeeds, then a check is performed to determine whether additional advice is satisfied by the plan beyond that which was being explicitly enforced. If there is no node in the lattice for the set of advice A_{sat} satisfied by the plan, then a new current node is created in the lattice and annotated with the plan. If there already is a node for A_{sat}, then it simply becomes the current node. The serendipity checks can significantly improve the efficiency of the overall search process, since the cost of testing whether a given plan satisfies a piece of advice is significantly lower than the cost of generating a new plan that incorporates the advice.

For a complete planner, failure to find a solution for a given set of advice indicates the presence of a conflict. However, planners whose development has been motivated by real-world applications often employ heuristics that sacrifice completeness for efficiency gains (Wilkins 1988). The local maxima search framework embodies a *defeasible negation as failure* philosophy: while an unsuccessful attempt to generate a solution for a given set of advice is recorded as a failure and such a case will never be directly tried again, a solution produced for another problem may serendipitously result in satisfaction of advice previously labeled as failed. The framework compensates for such occurrences by changing the status of the corresponding previously tested lattice node (as well as any failed ancestors in the lattice) and considering it as a future candidate for expansions.

```
L ← {}  ;; Initialize the Advice Lattice
LOOP  ;; Outer Loop: select seeds
   N ← GetNewSeed(A_all, L)
   WHEN N
      LOOP  ;; Inner Loop: expand below seed
         a ← SelectAddAdvice(N, A_all,L)

         IF a =⊥
            ClosedNode ← TRUE
         ELSE
            A ← A∪{a}
            P ← Solve(A)
            IF P
               A_sat ← CheckAdvice(A_all, P)
               N≈ ← NodeForAdvice(A_sat,L)

               ;; Test for an equivalent node
               IF N≈
                  N ← N≈
                  UNLESS Success(N≈)
                     ConvertToSuccess(N≈)
               ELSE
                  N ← AddNode(A, P, L)
            ELSE
               AddNode(A, ⊥, L)
      UNTIL ClosedNode
UNTIL N =⊥
```

Figure 3: Local Maxima Search Framework

Heuristic Seeding

The use of soft enforcement for generating seed solutions can expedite the identification of a local maximum, since it enables search to begin further down in the lattice.

Figure 4 presents a seed selection algorithm based on soft enforcement. The `IF` branch handles selection of the initial seed while the `ELSE` branch selects subsequent seeds (a process referred to as *reseeding*). For initial seed selection, soft enforcement is employed for the entire set of available advice. A corresponding lattice node is created for the advice that is satisfied by the resultant solution. For reseeding, the function *ReseedAdvice*(A_{all},L) generates potential reseed cases that are required to be different from the advice on lattice nodes that have been explored previously. However, there is no guarantee that the advice satisfied by a plan generated for a reseed case using soft enforcement will be distinct from all previously explored nodes. For this reason, soft enforcement of individual reseed cases must be successively tried until a plan is generated for which no previously explored lattice node satisfies the same advice. As with the exploration loop in the main LMS algorithm (Figure 3), nodes that were previously identified as failures are updated to correspond to successes in the event that a plan is generated that satisfies the same advice.

The objective in reseeding is to move to a portion of the advice lattice that will yield plans that satisfy different advice than do known solutions, with the intent of discovering a new local maximum as quickly as possible. One intuitively appealing strategy is to select as reseeds those advice sets that would map to *maximal unexplored nodes* in the current lattice. Here, *unexplored* means that there is not yet a node for that advice set A in the lattice; *maximal* means that solution generation has failed for every node in the lattice whose advice subsumes A, and there is no unexplored advice set $A' \supset A$ for which this same property holds.

The maximal open nodes can be computed using a strategy based on *minimal hitting sets* (Garey & Johnson 1979). A hitting set for a given collection of sets $\theta = \{S_1 \ldots S_n\}$ is a set that contains at least one element from each S_i; a minimal hitting set for θ, denoted by $MHS(\theta)$, is a set for which no subset is a hitting set. To generate reseed cases, we take the complement of the minimal hitting sets of advice satisfied by known local maxima, with respect to the original advice set.

Definition 2 (Minimal Hitting Set Reseeding) *Let* $M = \{\mathrm{A}_1^{\max} \ldots \mathrm{A}_k^{\max}\}$ *be the set of local advice maxima for a given set of advice* $\mathrm{A_{all}}$. *The* set of minimal hitting set reseeds, *denoted by* $R(M)$, *is defined to be*

$$R(M) = \{\mathrm{A_{all}} - \mathrm{H}_i \mid \mathrm{H}_i \in \mathrm{MHS}(M)\}$$

Reseeding Example Consider the advice set A =`{a1 a2 a3 a4}` and suppose that search has so far produced maximal solutions with the advice sets

```
HeuristicSeed(A_all, L)
  IF L = {}
    P ← HeuristicSolve(A_all)
    A_sat ← CheckAdvice(A_all, P)
    N ← AddNode(A_sat, P, L)
  ELSE
    N ← ⊥
    LOOP FOR A ∈ ReseedAdvice(A_all,L)
      P ← HeuristicSolve(A)
      A_sat ← CheckAdvice(A_all, P)
      N≈ ← NodeForAdvice(A_sat,L)
      IF N≈
        UNLESS Success(N≈)
          ConvertToSuccess(N≈)
          N ← N≈
      ELSE
        N ← AddNode(A, P, L)
    UNTIL N
  RETURN N
```

Figure 4: Heuristic Reseeding Seed Selection

`{a1 a2}` and `{a2 a3}`. In this case, the minimal hitting sets are `{{a1 a3} {a2}}`, resulting in the possible reseed cases `{{a2 a4} {a1 a3 a4}}`.

It is straightforward to prove the following proposition establishing that minimal hitting set reseed cases correspond to the *maximal open nodes* in a partially explored advice lattice (see the Appendix).

Proposition 1 (Maximal Unexplored Candidates) *For an advice lattice with an open node, the set of minimal hitting set reseeds is precisely the set of maximal candidate nodes.*

Related Work

The constraint reasoning community has a rich tradition of work on relaxation in the face of conflicting constraints (see (Freuder & Wallace 1992) for an overview). Similarly, the scheduling community has long focused on overconstrained problems that require relaxation methods (Fox 1987; Smith 1995). In contrast, there has been relatively little work on relaxation techniques for planning.

(Haddawy & Hanks 1998) presents a model for *partial goal satisfaction* within decision-theoretic planners that describes a framework for relaxing logical, quantitative, and temporal aspects of goals. The model of *approximately correct plans* relaxes the standard correctness requirements of causal link models (Ginsberg 1995); this weaker model of correctness has been used as the basis for a planning framework that can incrementally extend a plan to increase the scope of situations in which it is correct. Our requirement for accommodating conflicting advice constraints requires a different form of constraint relaxation than is addressed by those two models.

The GARI system for generating machining plans most closely matches the work described in this paper. GARI refines a skeletal plan through the application of rules that add temporal, equality, and instantiation constraints (Descotte & Latombe 1985). Rules consist of *condition/advice* pairs, where the conditions characterize world state and plan information and the advice is a weighted set of domain constraints to be used to refine the current plan. (In contrast to our model of advice as a task-specific adjunct to a given planning knowledge base, the advice within GARI constitutes core planning knowledge with embedded control information.) In a given state, a rule of highest priority is applied. Rule application will change the plan state and hence possibly the rules whose conditions are satisfied. Conflicts arise when rules recommend incompatible plan changes, and are resolved by backtracking to the most recently applied rule of lowest weight. This approach guarantees that the maximum weight of active advice contradicted by a solution is minimum over the set of possible plans.

Conclusions

This paper has presented a framework for partial satisfaction of conflicting advice, along with two algorithms that support different approaches to conflict resolution. For users interested in quick solutions, the MILAV method provides reasonable solutions at relatively low costs. For users interested in maximal solutions, the lattice-based framework for incremental construction of maximal solutions, along with a reseeding strategy, provides an anytime approach to solution generation.

The ideas presented here constitute a first step toward a complete framework for planning with conflicting advice. Here, we mention three areas for future work.

We envision an *advisable planner* as a core component of a mixed-initiative decision aid in which the planner and user work together to develop a final plan. Explanations as to what made certain pieces of advice unsatisfiable would provide users with insights that could help direct them in their search for better plans. Suggestions for satisfiable variants on stated advice would also be beneficial.

Given the computational cost of determining the unsatisfiability of a given set of advice, analytic techniques should be developed that can efficiently detect conflicts (or potential conflicts) among advice. Such techniques could be used to restrict the advice subsets for which solution generation is considered in local maxima search, resulting in increased efficiency.

The minimal hitting set reseed strategy identifies maximal open nodes within the advice lattice. However, it lacks any consideration of semantic differences among plans. Ideas from the work on generating *qualitatively different plans* (Myers & Lee 1999) could be incorporated to select reseed cases that would focus search toward plans with semantic differences from known local maxima.

Acknowledgments This research was supported by DARPA Contract F30602-97-C-0067, under the supervision of Air Force Research Lab – Rome.

Appendix

Proof of Proposition 1

First, we show that each reseed $R_i \in R(M)$ is neither a subset nor a superset of any A_j^{max}. By the definition of hitting sets, each A_j^{max} contains some $h \in H_i$, which cannot be in $A_{all} - H_i$. Thus $A_j^{max} \not\subseteq R_i$. Now suppose $R_i \subset A_j^{max}$; then

$$A_{all} - H_i \subset A_j^{max}$$

and so

$$A_{all} - H_i \subseteq A_j^{max} - H_i$$

which would imply $A_{all} = A_j^{max}$. But the conditions of the proposition state that the advice lattice contains open nodes, making this equality impossible. Thus $R_i \not\subset A_j^{max}$.

Now we show that each R_i is maximal. For any $a \notin R_i$, there is some H_i such that $a \in H_i$. Necessarily, there is some A_k^{max} such that a is the only element in H_i for which $a \in A_k^{max}$; if this were not true, then H_i would not be a minimal hitting set. But then $A_k^{max} \subseteq R_i \cup \{a\}$, and since A_k^{max} is maximal, necessarily $R_i \cup \{a\}$ is a known failure node. It follows that R_i must be a maximal untested node.

To complete the proof, we show that every maximal unexplored node F is some $R_k \in R(M)$. For every A_i^{max}, $F \not\subseteq A_i^{max}$. Thus, there exists some $a \in A_i^{max}$ such that $a \notin F$, that is, $a \in A_{all} - F$. Thus, $A_{all} - F$ is a hitting set for M. If $A_{all} - F$ is not a minimal hitting set, there is some H_j such that $A_{all} - F \subset H_j$, and so $F \supset A_{all} - H_j$. That is, $F \supset R_j$. As shown above, however, each reseed set is a maximal set in the lattice, thus contradicting the fact that F is a maximal unexplored set. It follows that $A_{all} - F$ must be a minimal hitting set for M, and so is identical to some R_i.

References

Burstein, M. H., and McDermott, D. V. 1994. Issues in the development of human-computer mixed-initiative planning systems. In Gorayska, B., and Mey, J., eds., *Cognitive Technology: In Search of a Humane Interface*. Elsevier Science.

Descotte, Y., and Latombe, J. 1985. Making compromises among antagonist constraints in a planner. *Artificial Intelligence* 27:183–217.

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

Ferguson, G., and Allen, J. 1998. TRIPS: An integrated intelligent problem-solving assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 567–572.

Fox, M. S. 1987. *Constraint-directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kaufmann.

Freuder, E., and Wallace, R. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21–70.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completness*. W. H. Freeman and Company.

Ginsberg, M. L. 1995. Approximate planning. *Artificial Intelligence* 76(1-2):89–123.

Haddawy, P., and Hanks, S. 1998. Utility models for goal-directed, decision-theoretic planners. *Computational Intelligence* 14(3).

Hogg, T.; Huberman, B. A.; and Williams, C. P. 1996. Phase transitions and the search problem. *Artificial Intelligence* 81(1-2).

Mayer, M. A. 1997. A natural language interface for the Advisable Planner. Senior Honors Thesis, Stanford Univeristy.

Myers, K. L., and Lee, T. J. 1999. Generating qualitatively different plans through metatheoretic biases. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. AAAI Press.

Myers, K. L. 1996. Strategic advice for hierarchical planners. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*, 112–123. Morgan Kaufmann Publishers.

Myers, K. L. 1999. *User Guide for the Advisable Planner*. Artificial Intelligence Center, SRI International, Menlo Park, CA. Version 2.3.

Smith, S. F.; Lassila, O.; and Becker, M. 1996. Configurable, mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology*. Menlo Park, CA: AAAI Press.

Smith, S. F. 1995. Reactive scheduling systems. In Brown, D., and Scherer, W., eds., *Intelligent Scheduling Systems*. Kluwer Press.

Tate, A.; Dalton, J.; and Levine, J. 1998. Generation of multiple qualitatively different plans. In *Proceedings of the Fourth International Conference on AI Planning Systems*.

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