Refinement Planning: Status and Prospectus

Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287, rao@asu.edu

Abstract

Most current-day AI planning systems operate by iteratively refining a partial plan until it meets the goal requirements. In the past five years, significant progress has been made in our understanding of the spectrum and capabilities of such refinement planners. In this talk, I will summarize this understanding in terms of a unified framework for refinement planning and discuss several current research directions.

Introduction

Developing automated methods for generating and reasoning about plans and schedules, whether in aid of autonomous or human agents, has been part and parcel of AI research from the beginning. The need for planning arises naturally when an agent is interested in controlling the evolution of its environment. Algorithmically, a planning problem has as input a set of possible courses of actions, a predictive model for the underlying dynamics, and a performance measure for evaluating the courses of action. The output or solution is one or more courses of action that satisfy the specified requirements for performance. A planning problem thus involves deciding "what" actions to do, and "when" to do them. The "when" part of the problem has traditionally been called the "scheduling" problem [20].

The simplest case of the planning problem, where the environment is static and deterministic, and the planner has complete information about the current state of the world, has come to be known as the **classical planning problem**. My talk is concerned with algorithms for synthesizing plans in classical planning. Generating plans for classical planners has received significant attention over the past twenty years. Most of the plan generation algorithms that have been developed are informally called "refinement planners", in that they iteratively refine a partial plan until it meets the specified goals. In this talk, I will attempt to provide a coherent semantic picture of refinement planning, and describe the various existing approaches in terms of this framework. I will also consider the tradeoffs inherent in refinement planning, and possible directions for developing more efficient refinement planners.

*This research is supported in part by NSF research initiation award (RIA) IRI-9210997, NSF young investigator award (NYI) IRI-9457634 and ARPA/Rome Laboratory planning initiative grants F30602-93-C-0039 and F30602-95-C-0247. Special thanks to Biplav Srivastava, Gopi Bulusu, Suresh Katukam, and Laurie Ihrig for the many hours of discussions, David McAllester for his patient correspondence regarding SNLP and refinement search, and Dan Weld for his encouragement. Portions of this paper are borrowed from a recent overview of planning approaches, which I co-authored with Tom Dean.

Preliminaries of Modeling Change: Before proceeding further, let me briefly review how classical planning problems are modeled. In most classical planning approaches, a state is described in terms of a set of boolean state variables. Suppose that we have three boolean state variables: P, Q, and R. We represent the particular state s in which P and Q are true and R is false by the *state-variable assignment*, $s = \{P = \text{true}, Q = \text{true}, R = \text{false}\}$, or, somewhat more compactly, by $s = \{P, Q, \neg R\}$.

An action is represented as a *state-space operator* α defined in terms of *preconditions* (Pre(α)) and *postconditions* (also called effects) (Post(α)). If an operator (action) is applied (executed) in a state in which the preconditions are satisfied, then the variables mentioned in the postconditions are assigned their respective values in the resulting state. If the preconditions are not satisfied, then there is no change in state.

Several syntactic extensions can be added on top of this basic operator representation, facilitating conditional effects and effects quantified over finite universes. Pednault [17] shows that this action representation is semantically equivalent to the largest subset of situation calculus for which we can get by without writing frame axioms explicitly.

Goals are represented as state-variable assignments that assign values to subsets of the set of all state variables. By assigning values to one or more state variables, we designate a set of states as the goal. We say that a state s *satisfies* a goal ϕ, notated $s \models \phi$, just in case the assignment ϕ is a subset of the assignment s. Given an initial state s_0, a goal ϕ, and a library of operators, the *objective* of the planning problem is to find a sequence of state-space operators $\langle \alpha_1, \ldots, \alpha_n \rangle$ such that $f(s_0, \langle \alpha_1, \ldots, \alpha_n \rangle) \models \phi$.

Semantic picture of Refinement Planning

Refinement planners [8] attempt to solve a planning problem by navigating the space of *sets of potential solutions (action sequences)*. The potential solution sets are represented and manipulated in the form of "partial plans." Syntactically, a partial plan π can be seen as a set of constraints (see below). Semantically, a partial plan is a shorthand notation for the set

of action sequences that are consistent with its constraints. The set of such action sequences is called the set of candidates (or candidate set) of the partial plan.

We define a generic refinement planning procedure, Refine(π), as follows [8].

1. If an action sequence $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ is a candidate of π and also solves the planning problem, terminate and return the action sequence.
2. If the constraints in π are inconsistent, then eliminate π from future consideration.
3. Select a refinement strategy, and apply the strategy to π and add the resulting refinements to the set of plans under consideration.
4. Nondeterministically select a plan π' from those under consideration and call Refine(π').

The first step of the search process is the "solution construction" process, where the planner attempts to extract a solution from the current partial plan's candidate set. We shall see later that the solution constructor function checks only on the minimal candidates of the plan, since the candidate set of a partial plan can be infinitely large [8]. The second step is closely related to the first, and attempts to prune the plan from further refinement if it can be shown not to contain any solutions. The last two steps involve applying a refinement operator to the partial plan to generate new partial plans, and recursing on one of those refinements. Refinements can be understood as operations that split the candidate set of the partial plan to which they are applied. Specifically, a refinement strategy converts a partial plan π into a set of new plans $\{\pi_1, \ldots, \pi_n\}$ such that the candidate set of each π_i is a subset of the candidate set of π. A refinement operator is said to be *complete* if every solution belonging to the candidate set of the plan will be in the candidate sets of at least on of the plans generated by the refinement operator. A refinement operator is said to be *systematic* if the candidate sets of the refinements are disjoint. It is easy to see that the selection of refinement strategy does not have to backtracked over, as long as the refinement operators are complete.

The specifics of a refinement planning algorithm will differ depending on the representation of the partial plans used (i.e., what specific constraints are employed) and the type of refinements employed on that representation. We already pointed out that syntactically, a partial plan is a set of constraints. The semantic status of a plan constraint is clarified by specifying when a given action sequence is said to satisfy the constraint. Within these broad guidelines, a large variety of syntactic representations can be developed. Once a representation for a partial plan is given, a refinement operator can be specified in terms of the types of constraints that it adds to a partial plan. If the constraint sets added by refinements are mutually exclusive and exhaustive, then the refinement operators will be systematic and complete.

Representing Partial Plans

To focus our discussion, we will start by looking at a specific partial plan representation that is useful for modeling most existing planners (later, we will consider alternative representations that are promising). In this representation, *partial plan* consists of a set of *steps*, a set of *ordering constraints* that restrict the order in which steps are to be executed, and a set of *auxiliary constraints* that restrict the value of state variables over particular intervals of time. Each step is associated

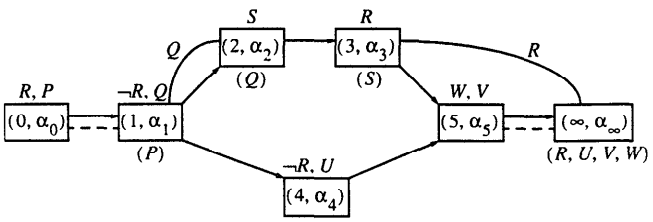

Figure 1: This figure depicts the partial plan π_{cg}. The postconditions (effects) of the steps are shown above the steps, while the preconditions are shown below the steps in parentheses. The ordering constraints between steps are shown by arrows. The interval preservation constraints are shown by arcs, while the contiguity constraints are shown by dotted lines.

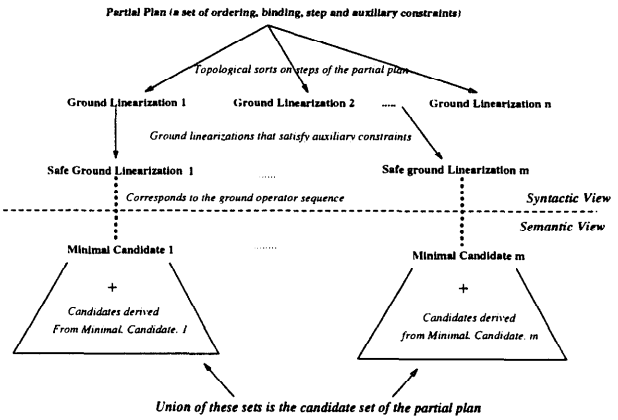

Figure 2: A schematic illustration of the relation between a partial plan and its candidate set. T

with a state-space operator. To distinguish between multiple instances of the same operator appearing in a plan, we assign to each step a unique integer i and represent the ith step as the pair (i, α_i) where α_i is the operator associated with the ith step. Figure 1 shows a partial plan π_{eg} consisting of seven steps. The plan π_{eg} is represented as follows.

$$\langle \quad \{(0,\alpha_0),(1,\alpha_1),(2,\alpha_2),(3,\alpha_3),(4,\alpha_4),(5,\alpha_5),(\infty,\alpha_\infty\},$$
$$\{(0 \bar{\prec} 1),(1 \prec 2),(1 \prec 4),(2 \prec 3),(3 \prec 5),(4 \prec 5),(5 \bar{\prec} \infty)\},$$
$$\{(1 \overset{Q}{-} 2),(3 \overset{R}{-} \infty)\} \ \rangle$$

An ordering constraint of the form $(i \prec j)$ indicates that Step i precedes Step j. An ordering constraint of the form $(i \bar{\prec} j)$ indicates that Step i is contiguous with Step j, that is Step i precedes Step j and no other steps intervene. The steps are *partially ordered* in that Step 2 can occur either before or after Step 4. An auxiliary constraint of the form $(i \overset{P}{-} j)$ is called an *interval preservation constraint* and indicates that P is to be preserved in the range between Steps i and j (and therefore no operator with postcondition $\neg P$ should occur between Steps i and j). In particular, according to the constraint $(3 \overset{R}{-} \infty)$, Step 4 should not occur between Steps 3 and ∞.

Figure 2 shows the schematic relations between a partial plan in such a representation and its candidate set, and we

will illustrate it with respect to the example plan in Figure 1. Each partial plan corresponds to a set of topological sorts (e.g. $\langle 1,2,3,4,5\rangle$ and $\langle 1,2,4,3,5\rangle$). The subset of these that satisfy the auxiliary constraints of the plan (e.g. $\langle 1,2,4,3,5\rangle$) are said to be the safe-ground linearizations of the plan. Each safe ground linearization of the plan corresponds to an action sequence which is a minimal candidate of the partial plan (e.g. $\langle \alpha_1, \alpha_2, \alpha_4, \alpha_3, \alpha_5\rangle$). An infinite number of additional candidates can be derived from each minimal candidate of the plan by augmenting (padding) it with additional actions without violating the auxiliary constraints (e.g. $\langle \alpha_1, \alpha_2, \alpha_2, \alpha_4, \alpha_3, \alpha_5\rangle$). Thus, the candidate set of a partial plan is infinite, but the set of its minimal candidates is finite. The solution constructor functions search the minimal candidates of the plan to see if any of them are solutions to the planning problem. Refinement process can be understood as incrementally increasing the size of these minimal candidates so that action sequences of increasing lengths are examined to see if they are solutions to the problem. The search starts with the null plan $\langle\{(0,\alpha_0),(\infty,\alpha_\infty)\},\{(0\prec\infty)\},\{\}\rangle$, where α_0 is a dummy operator with no preconditions and postconditions corresponding to the initial state, and α_∞ is a dummy operator with no postconditions and preconditions corresponding the goal.

Refining Partial Plans

There are several possible ways of refining partial plans, corresponding intuitively to different ways of splitting the set of potential solutions represented by the plan. In the following sections, I outline several popular refinement strategies employed in the planning literature.

State-Space Refinements

The most straightforward way of refining partial plans involves using progression to convert the initial state into a state satisfying the goal conditions, or using regression to convert a set of goal conditions into a set of conditions that are satisfied in the initial state. From the point of view of partial plans, this corresponds to growing prefix or the suffix of the plan. The refinements are called state-space refinements since given either the prefix or the suffix of a plan, we can uniquely determine the nature of the world state following the prefix and preceding the suffix.

The set of steps $\{\sigma_1,\sigma_2,\ldots,\sigma_n\}$ with contiguity constraints $\{(\sigma_0 \bar{\prec} \sigma_1),(\sigma_1 \bar{\prec} \sigma_2),\ldots,(\sigma_{n-1} \bar{\prec} \sigma_n)\}$ is called the *header* of the plan π. The last element of the header, σ_n, is called the *head step*. The state defined by $f(s_0,\langle\alpha_{\sigma_1},\ldots,\alpha_{\sigma_n}\rangle)$, where α_{σ_i} is the operator associated with σ_i is called the *head state*. In a similar manner, we can define the *tail*, *tail step*, and *tail state*. As an example, the partial plan π_{eg} shown in Figure 1 has the Steps 0 and 1 in its header, with Step 1 being the head step. The head state (which is the state resulting from applying α_1 to the initial state) is $\{P,Q\}$. Similarly, the tail consists of Steps 5 and ∞, with Step 5 being the tail step. The tail state (which is the result of regressing the goal conditions through the operator α_5) is $\{R,U\}$.

Progression (or forward state-space) refinement involves advancing the head state by adding a step σ, such that the preconditions of α_σ are satisfied in the current head state, to the header of the plan. The step σ may be newly added to the plan or currently present in the plan. In either case, it is made contiguous to the current head step and becomes the new head step.

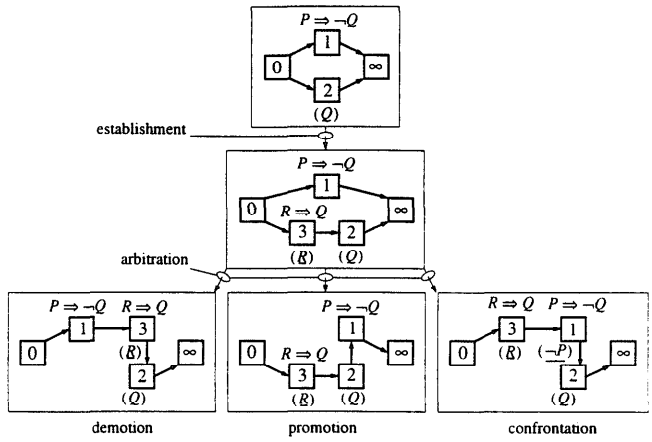

Figure 3: Example of plan-space refinement

As an example, one way of refining the plan π_{eg} in Figure 1 using progression refinement would be to apply an instance of the operator α_2 (either the instance that is currently in the plan $(2,\alpha_2)$ or a new instance) to the head state (recall that it is $\{P,Q\}$). This is accomplished by putting a contiguity constraint between $(2,\alpha_2)$ and the current head step $(1,\alpha_1)$ (thereby making the former the new head step).

In realistic problems, many operators may be applicable in the head state and very few of them may be relevant to the top level goals. To improve efficiency, some planners use **means-ends analysis** to focus on relevant operators. The general idea is the following: Suppose we have an operator α whose postconditions match a goal of the problem. Clearly, α is a relevant operator. If the preconditions of α are satisfied in the head state of the current partial plan, we can apply it directly. Suppose they are not all satisfied. In such a case, we can consider the preconditions of α as subgoals, look for an operator α' whose postconditions match one of these subgoals, and check if it is applicable to the head state. This type of recursive analysis can be continued to find the set of relevant operators, and focus progression refinement [14].

We can also define a refinement strategy based on regression, which involves regressing the tail state of a plan through an operator. For example, the operator α_3 is applicable (in the backward direction) through this tail state (which is $\{R,U\}$), while the operator α_4 is not (since its postconditions are inconsistent with the tail state). Thus, one way of refining π_{eg} using regression refinement would be to apply an instance of the operator α_3 (either the existing instance in Step 3 or a new one) to the tail state in the backward direction. This is accomplished by putting a contiguity constraint between $(3,\alpha_3)$ and the current tail step.

In both progression and regression, solution constructor function can be simplified as follows: check to see if head state is a super set of the tail state, and if so, return the header concatenated with tail.

Plan-Space Refinements

State-space refinements have to guess correct answers to two questions up front: (a) whether a specific action is relevant to

the goals of the planning problem and (b) where exactly in the final plan does the action take place. Often, it is easier to see whether or not a given action is relevant to a plan, but much harder to guess the precise position at which a step must occur in the final plan. The latter question more naturally falls in the purview of "scheduling" and cannot be answered well until all of the steps have been added. To avoid this premature forced commitment, we would like to introduce the new action into the plan, without committing to its position in the final solution. This is the intuition behind plan-space refinements. The refinement is named "plan-space" because when we allow an action to be part of a plan without constraining it to be either in the prefix or the suffix, the partial plan does not represent a unique world state. Thus, the search cannot be recast in terms of the space of world states.

The main idea in plan-space refinement is to shift the attention from advancing or regressing the world state to establishing goals in the partial plan. A precondition P of a step (i, α_i) in a plan is said to be *established* if there is some step (j, α_j) in the plan that precedes i and causes P to be true, and no step that can possibly intervene between j and i has postconditions that are inconsistent with P. It is easy to see that if every precondition of every step in the plan is established, then that plan will be a solution plan. Plan-space refinement involves picking a precondition P of a step (i, α_i) in the partial plan, and adding enough additional step, ordering, and auxiliary constraints to ensure the establishment of P. One problem with this precondition-by-precondition establishment approach is that the steps added in establishing a precondition might unwittingly violate a previously established precondition. Although this does not affect the completeness of the refinement search, it can lead to wasted planning effort, and necessitate repeated establishments of the same precondition within the same search branch. Many variants of plan-space refinements avoid this inefficiency by *protecting* their establishments using IPCs. When the planner uses plan-space refinements exclusively, its refinement process can terminate as soon as any of the safe ground linearization of the plan correspond to solutions.

Let me illustrate the main ideas in precondition establishment through an example. Consider the partial plan at the top in Figure 3. Step 2 in this plan requires a precondition Q. To establish this precondition, we need a step which has Q as its postcondition. None of the existing steps have such a postcondition. Suppose an operator α_3 in the library has a postcondition $R \Rightarrow Q$. We introduce an instance of α_3 as Step 3 into the plan. Step 3 is ordered to come before Step 2 (and after Step 0). Since α_3 makes Q true only when R is true before it, to make sure that Q will be true following Step 3, we need to ensure that R is true before it. This can be done by posting R as a precondition of Step 3. Since R is not a normal precondition of α_3, and is being posted only to guarantee one of its conditional effects, it is called a *secondary precondition* [17]. Finally, we can protect the establishment of precondition Q by adding the constraint $3 \overset{Q}{-} 2$. If we also want to ensure that 3 remains the sole establisher of Q in the final solution, we can add another auxiliary constraint $3 \overset{\neg Q}{-} 2$. In [13], McAllester shows that adding these two auxiliary constraints ensures systematicity of plan-space refinement.

Tractability Refinements: Since the position of the steps in the plan is not uniquely determined after a plan space refinement, there is uncertainty regarding (a) the state of the world preceding or following a step, (b) the relative order of steps in the plan and (c) the truth of IPC constraints in the plan. A variety of refinement strategies exist that attempt to make the reasoning with partial plans tractable by pushing the complexity into the search space. These refinements, called *tractability refinements*, fall into three broad classes: *pre-positioning, pre-ordering and pre-satisfaction* refinements. The first pick a pair of steps α_1 and α_2 in the plan and generate two refinements one in which $\alpha_1 \bar{\prec} \alpha_2$, and the other in which $\alpha_1 \not\bar{\prec} \alpha_2$. The pre-ordering refinements do the same thing except they enforce ordering rather than contiguity constraints between the chosen steps. Finally, the pre-satisfaction refinements pick an IPC in the plan, and enforce constraints such that every ground linearization of the plan satisfies the IPC (see below).

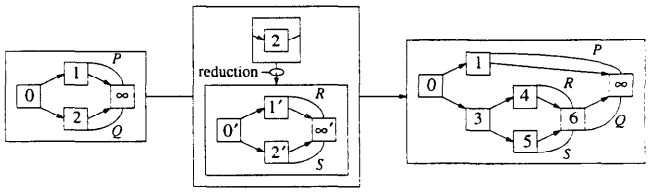

Figure 4: Step 2 in the partial plan shown on the left is reduced to obtain a new partial plan shown on the right. In the new plan, Step 2 is replaced with the (renamed) steps and constraints specified in the reduction shown in the center box.

We can illustrate the pre-satisfaction refinements through the example in Figure 3, after we have introduced Step 3 and ensured that it produces Q as a postcondition, we need to make sure that Q is not violated by any steps possibly intervening between Steps 3 and 2. In our example, Step 1, which can possibly intervene between Steps 3 and 2, has a postcondition $P \Rightarrow \neg Q$, that is potentially inconsistent with Q. To avert this inconsistency, we can either order Step 1 to come before Step 3 (demotion), or order Step 1 to come after Step 2 (promotion), or ensure that the offending conditional effect will not occur. This last option, called confrontation, can be carried out by posting $\neg P$ as a (secondary) precondition of Step 1.

Depending on whether protection strategies are used, and what tractability refinements are used, we can get a very large spectrum of plan-space refinements [8]. The effectiveness of plan space refinement in controlling the search is determined by a variety of factors, including (a) the order in which the various preconditions are selected for establishment (b) the manner in which tractability refinements are applied during search. See [8] for a discussion of some of the trade-offs.

Task-Reduction Refinements

In both the state-space and plan-space refinements, the only knowledge that is assumed to be available about the planning task is in terms of primitive actions (that can be executed by the underlying hardware), and their preconditions and postconditions. Often, one has more structured planning knowledge available in a domain. For example, in a travel planning domain, we might have the knowledge that one can reach a destination by either "taking a flight" or by

"taking a train". We may also know that "taking a flight" in turn involves making a reservation, buying a ticket, taking a cab to the airport, getting on the plane etc. In such a situation, we can consider "taking a flight" as an abstract task (which cannot be directly executed by the hardware). This abstract task can then be reduced to a plan fragment consisting of other abstract or primitive tasks (in this case "making a reservation", "buying a ticket", "going to the airport", "getting on the plane"). This way, if there are some high-level problems with the "taking flight" action and other goals, (e.g. there is not going to be enough money to take a flight as well paying the rent), we can resolve them *before* we work on low level details such as getting to the airport. The resolution is can be carried out by the generalized versions of tractability refinements used in plan-space refinement.

This idea forms the basis for task reduction refinement. Specifically, we assume that in addition to the knowledge about primitive actions, we also have some abstract actions, and a set of schemas (plan fragments) that can replace any given abstract action. Task reduction refinement takes a partial plan π containing abstract and primitive tasks, picks an abstract task σ, and for each reduction schema (plan fragment) that can be used to reduce σ, a refinement of π is generated with σ replaced by the reduction schema (plan fragment). As an example, consider the partial plan on the left in Figure 4. Suppose the operator α_2 is an abstract operator. The central box in Figure 4 shows a reduction schema for Step 2, and the partial plan shown on the right of the figure shows the result of refining the original plan with this reduction schema. At this point any interactions between the newly introduced plan fragment and the previously existing plan steps can be resolved using techniques such as promotion, demotion and confrontation discussed in the context of plan-space refinement. This type of reduction is carried out until all the tasks are primitive.

Notice that the partial plans used in task reduction planning contain one additional type of constraint -- the non-primitive tasks. Informally, when a plan contains a non-primitive task t, then every candidate of the plan must have the actions comprising at least one concretization of t (where a concretization of a non-primitive task is the set of primitive partial plans that can be generated by reducing it using task reduction schemas).

Tradeoffs in Refinement Planning

Now that we looked at a variety of approaches to refinement planning, it is worth looking at the broad tradeoffs in refinement planning. There are two classes of tradeoffs -- the first arising from algorithmic modifications to the generic refinement search, and the second arising from the match between refinements and the characteristics of the planning domain.

An example of the first class of tradeoffs is that between the cost of solution constructor vs. size of the search space. We can reduce the search space size by considering partial plans that can compactly represent a larger number of minimal candidates. From a planning view point, this leads to *least commitment* on the part of the planners. However, as the number of candidates represented by a partial plan grow, the cost of the picking a solution from the partial plan increases.

This tradeoff is well represented in the refinements that we have looked at. Plans produced by state-space refinements will have single minimal candidates, while those produced by plan space refinements can have multiple minimal candidates (corresponding roughly to the many topological sorts of the plan). Finally, partial plans produced using task reduction refinements may have even larger number of minimal candidates since the presence of a non-primitive tasks essentially allows any action sequence that contains any concretization of the non-primitive task as a minimal candidate.

There are also certain tradeoffs that arise from the match between the plan representations and refinements used, and the characteristics of the planning domain and problem. For example, it is known that the plan-space refinements can be more efficient compared to state-space refinements in domains where the ordering of steps cannot be guessed with reasonable accuracy *a priori* [1; 16]. The plan-space refinements also allow separation of action selection and establishment phases from the "scheduling" phase of the planning, thus facilitating easier adaptation of the plan to more situations [7], and to more closely integrate the planning and scheduling phases [4]. On the other hand, state-space refinements provide a good sense of the state of the world corresponding to the partial plan, and can thus be useful to agents who need to do non-trivial reasoning about the world state to focus their planning and execution efforts [2; 14]. Finally, task-reduction refinements facilitate user control of planner's access to the primitive actions, and are thus the method of choice in any domain where the user has preferences among the solution plans [9].

Prospectus

Although early refinement planning systems tended to subscribe exclusively to a single refinement strategy, our unifying treatment of refinement planning demonstrates that it is possible to use multiple refinement strategies. As an example, the partial plan π_{eg} shown in Figure 1 can be refined with progression refinement (*e.g.*, by putting a contiguity constraint between Step 1 and Step 2), with regression refinement (*e.g.*, by putting a contiguity constraint between Step 3 and Step 5), or plan-space refinement (*e.g.*, by establishing the precondition S of Step 3 with the help of the effect Step 2). Finally, if the operator α_4 is a non-primitive operator, we can also use task reduction refinement to replace α_4 with its reduction schema. There is some evidence that planners using multiple refinement strategies intelligently can outperform those using single refinement strategies [10]. However, the question as to which refinement strategy should be preferred when is still largely open.

We can be even more ambitious however. Most existing refinement planners have trouble scaling up to larger problems, because of the very large search spaces they generate. While application of machine learning techniques to planning [15] hold a significant promise, we can also do better by improving the planning algorithms. One way of controlling the search space blow-up is to introduce appropriate forms of disjunction into the partial plan representation. By doing this, we can allow a single partial plan to stand for a larger number of minimal candidates. The conventional wisdom in refinement planning has been to keep the solution construction function tractable by pushing the complexity into the search space [8]. Some recent work by Blum and Furst [3] shows that partial plan representations that push all the complexity into the solution construction function may actu-

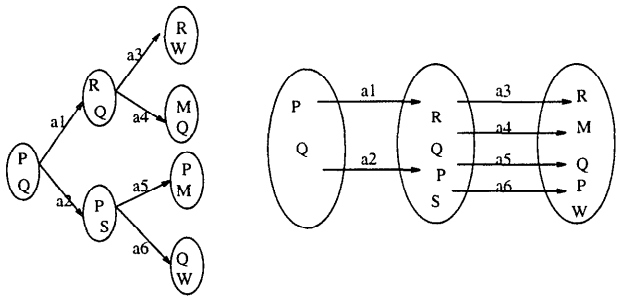

Figure 5: To the left is the search space generated by a refinement planner using progression refinement. To the right is the partial plan representation, called plan graph, used in Graphplan [3]. Each candidate plan of the plan graph must have some subset of the actions in i^{th} level coming immediately before some subset of actions in the $i+1^{th}$ level (for all i). The minimal candidates corresponding to all plans generated by the progression planner are compactly represented by a single partial plan (plan graph) in Graphplan.

ally perform much better in practice. They describe a system called Graphplan in which the partial plan representation, called plan graph, corresponds to a disjunctive representation of the search space of a progression planner (see Figure 5) [11]. The Graphplan refinement process (i.e., the process of growing the plan-graph) does not introduce any branching into the search space. Thus, all the complexity is transferred to the solution construction process which has to search the plan graph structure for minimal candidates that are solutions. Empirical results demonstrate this apparently extreme solution to the refinement and solution construction tradeoff in fact leads to significant improvements in performance.

The success of Graphplan shows that there is a lot to be gained by considering other disjunctive partial plan representations. An important issue in handling disjunctive partial plans is how to avoid losing all the search space savings in increased plan handling costs. One of the tricks in increasing least commitment without worsening the overall performance significantly seems to be to use constraint propagation techniques to enforce local consistency among the partial plan constraints. In CSP problems [18], refinement is used hand-in-hand with local consistency enforcement through constraint propagation to improve search performance. Although most refinement planning systems ignored the use of constraint propagation in planning, the situation is changing slowly. In addition to Graphplan [3], which uses the constraint propagation process in both the partial plan construction, and solution construction phase, there are also systems such as Descartes [6], which attempt to incorporate constraint propagation techniques directly into existing refinement planners. Solution construction process can also be represented as an instance of propositional satisfiability problem, and there is some recent evidence [12] that nonsystematic search techniques such as GSAT can give very good performance on such SAT instances.

Summary

In this talk, I described the current state of refinement planning algorithms using a unified framework for refinement planning. The framework explicates the tradeoffs offered by plan representation and refinement strategies. I have concluded by outlining several directions in which refinement planning algorithms can be made more efficient. These involve using disjunctive partial plan representations, and the using of CSP techniques for handling partial plans.

References

[1] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, Vol. 67, No. 1, 1994.

[2] F. Bachus and F. Kabanza. Using Temporal Logic to Control Search in a forward chaining planner. In *Proc European Planning Workshop*, 1995.

[3] A. Blum and M. Furst. Fast planning throug planning graph analysis. In *Proc. IJCAI-95*, 1995.

[4] K. Currie and A. Tate. O-Plan: The open planning architecture. *Artificial Intelligence*, 51(1):49--86, 1991.

[5] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189--208, 1971.

[6] D. Joslin and M. Pollack. Passive and active decision postponement in plan generation. In *Proc. 3rd European Workshop on Planning*, 1995.

[7] L. Ihrig and S. Kambhampati. Derivational replay for partial order planning. In *Proc. AAAI-94*.

[8] S. Kambhampati, C. Knoblock, and Q. Yang. Refinement search as a unifying framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, 76(1-2), 1995.

[9] S. Kambhampati. A comparative analysis of partial-order planning and task-reduction planning. *ACM SIGART Bulletin*, 6(1), 1995.

[10] S. Kambhampati and B. Srivastava. Universal Classical Planner: An algorithm for unifying state space and plan space approaches. In *Proc European Planning Workshop*, 1995.

[11] S. Kambhampati. Planning Methods in AI (Notes from ASU Planning Seminar). ASU CSE TR 96-004. *http://rakaposhi.eas.asu.edu:8001/yochan.html*

[12] H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search In *Proc. AAAI-96*.

[13] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. 9th AAAI*, 1991.

[14] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. AIPS-96*, 1996.

[15] Steve Minton, editor. *Machine Learning Methods for Planning and Scheduling*. Morgan Kaufmann, 1992.

[16] S. Minton, J. Bresina and M. Drummond. Total Order and Partial Order Planning: a comparative analysis. Journal of Artificial Intelligence Research 2 (1994) 227-262.

[17] E.P.D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4(4):356--372, 1988.

[18] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, San Diego, California, 1993.

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

[20] M. Zweben and M.S. Fox, editors. *Intelligent Scheduling*. Morgan Kaufmann, San Francisco, California, 1994.