Invited talk given at AAAI-96

Refinement Planning: Status and Prospectus

Subbarao Kambhampati
Department of Computer Science &Engg.
Arizona State University
Tempe, AZ 85287-5406
http://rakaposhi.eas.asu.edu

Today, I will be talking about refinement planning, the dominant paradigm for plan synthesis in artificial intelligence.

The area of refinement planning has been around for over twenty five years. However, our understanding of the foundations of refinement planning have improved markedly in the recent years.

My aim today is to share this understanding with you and outline several current research directions.

Overview

- Classical planning problem
- Refinement Planning: Formal Framework
- Existing refinement strategies
- Tradeoffs in Refinement Planning
- Scaling up Refinement Planning
- Conclusion

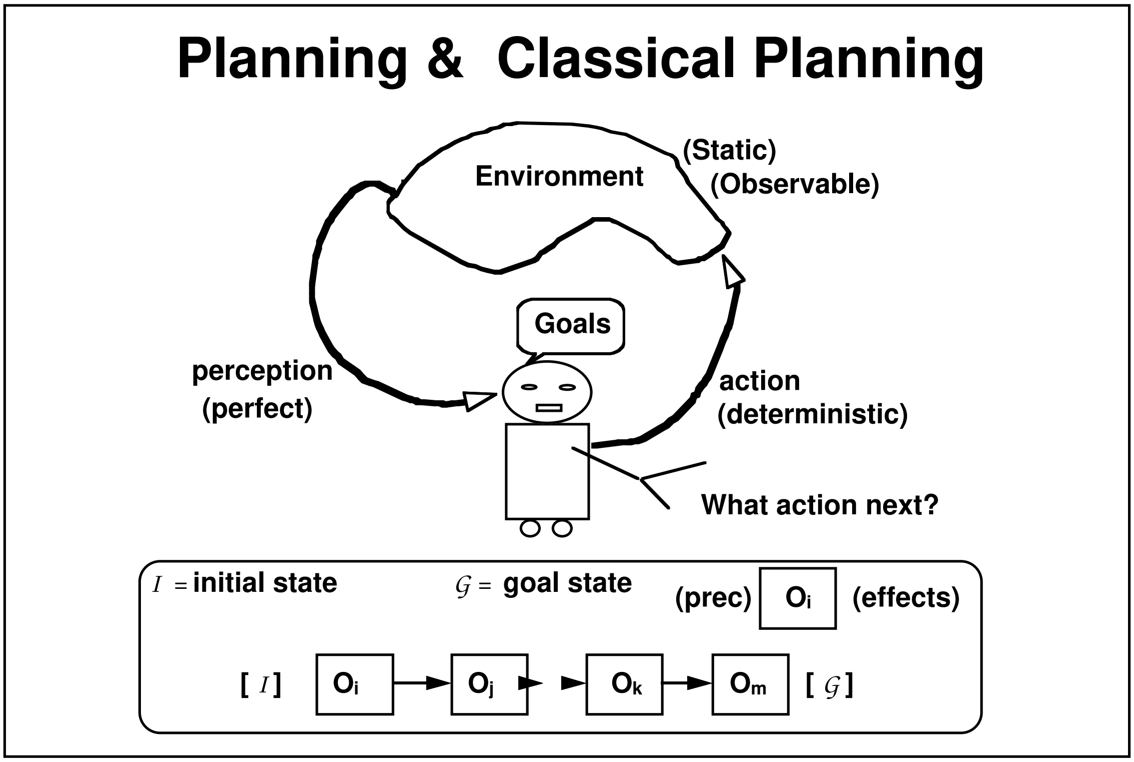
Here is the outline of my talk. I will briefly discuss the classical planning problem in Artificial Intelligence, and propose refinement planning as a way of unifying the many classical planning algorithms.

I will then present the formal framework of refinement planning

Next, I will describe the refinement strategies that correspond to existing planners

I will then discuss the tradeoffs among different refinement planning algorithms.

Finally, I will describe some promising directions for scaling up refinement planning algorithms.



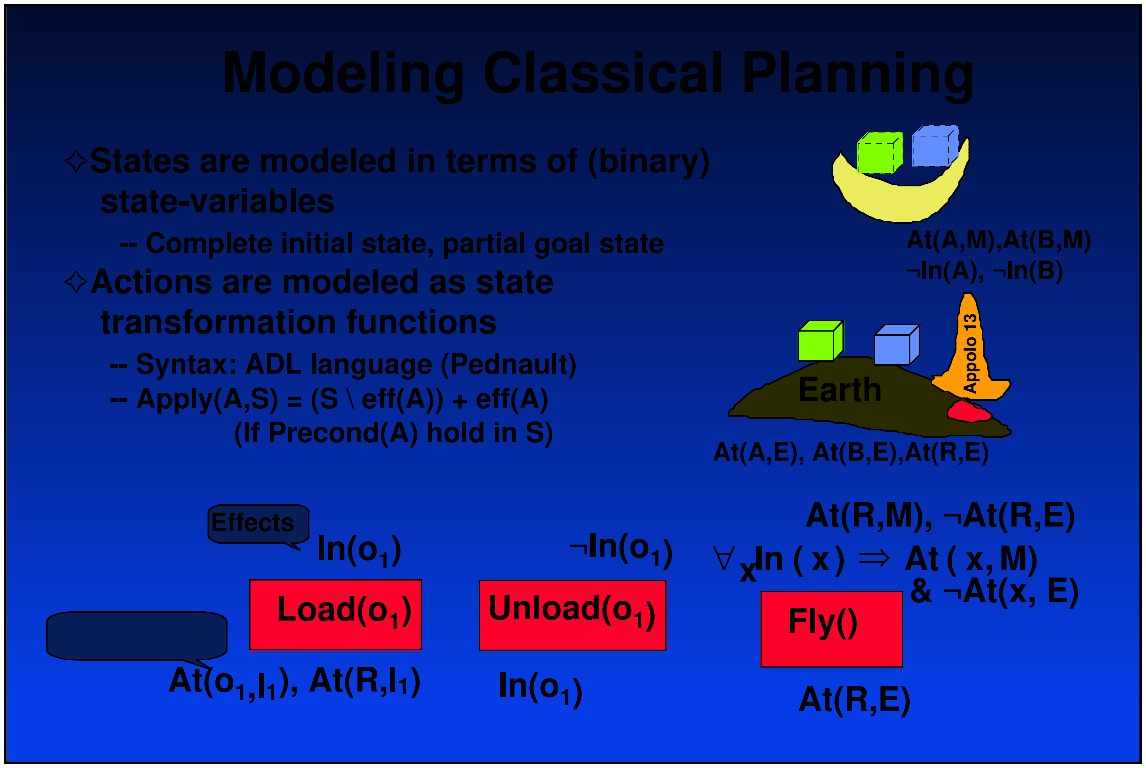
Intelligent agency involves controlling the evolution of external environments in desirable ways. Planning provides a way in which the agent can maximize its chances of effecting this control. Informally, a plan can be seen as a course of actions that the agent decides upon based on its overall goals, information about the current state of the environment, and the dynamics of its evolution.

The complexity of plan synthesis depends on a variety of properties of the environment and the agent. Perhaps the simplest case of planning occurs when the environment is static, (*in that it changes only in response to the agents actions*), observable and the agent's actions have deterministic effects on the state of the environment.

Plan synthesis under these conditions has come to be known as the classical planning problem.

Classical planning problem is thus specified by describing the initial state of the world, the desired goal state, and a set of deterministic actions. The objective is to find a sequence of these actions, which when executed from the initial state leads the agent to the goal state.

Despite its limitations, classical planning problem is still computationally hard, and has received a significant amount of attention. Work on classical planning has historically helped our understanding of planning under non-classical assumptions.



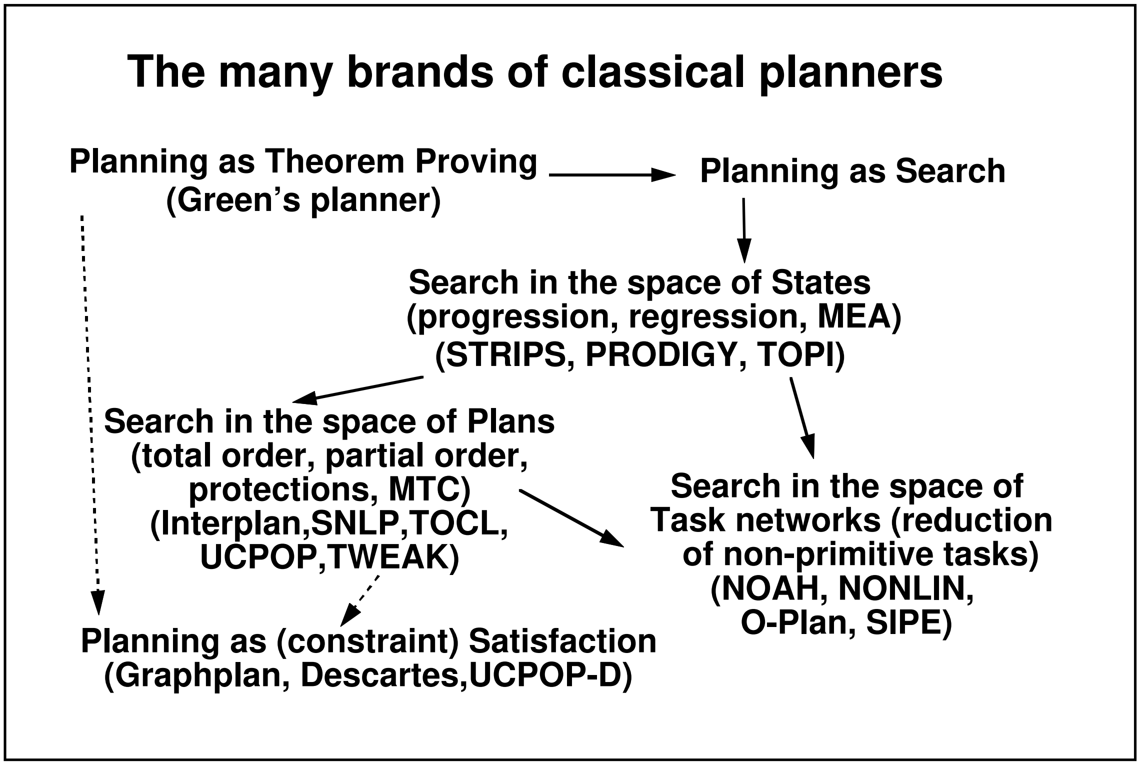
We shall now look at the way the classical planning problem is modeled. Let us use a very simple example scenario -- that of transporting two packets from earth to Moon, using a single (and somewhat out-of-shape) rocket.

States of the world are modeled in terms of a bunch of binary state-variables. Initial state is assumed to be completely specified, so negated conditions need not be seen. Goals involve achieving the specified values for certain state variables.

Actions are modeled as state-transformation functions, with pre-conditions and effects. A widely used action syntax is Pednault's creatively named Action Description Language, where preconditions and effects are first order quantified formulas (with no disjunction in the effects formula).

We have three actions in our rocket domain -- load which makes a package to be IN the rocket, unload, which gets it out, and Fly, which takes everything in the rocket over to the moon. Notice the quantified and negated effects in the case of FLY. Its second effect says that every box that is inside the rocket-- before the FLY action is executed -- will be at the moon after the action.

An action can be executed in any state where its preconditions hold and upon execution the state is modified such that state-variables named in the effects have the specified values.



Plan generation under classical assumptions had received wide-spread attention and a large variety of planning algorithms have been developed.

These include theorem proving approaches, search in the space of states, search in the space of plans, search in the space of task networks and more recently, planning as satisfaction.

One of my aims in this talk is to put these ideas in a logically coherent framework so we can see the essential connections between them.

I will use the "refinement planning" framework to effect such a unification.

Overview

- Classical planning problem
- Refinement Planning: Formal Framework
  - Partial plan: Syntax and Semantics
  - Refinement strategies
  - Refinement planning templates
- Existing refinement strategies
- Tradeoffs in Refinement Planning
- Scaling up Refinement Planning
- Conclusion

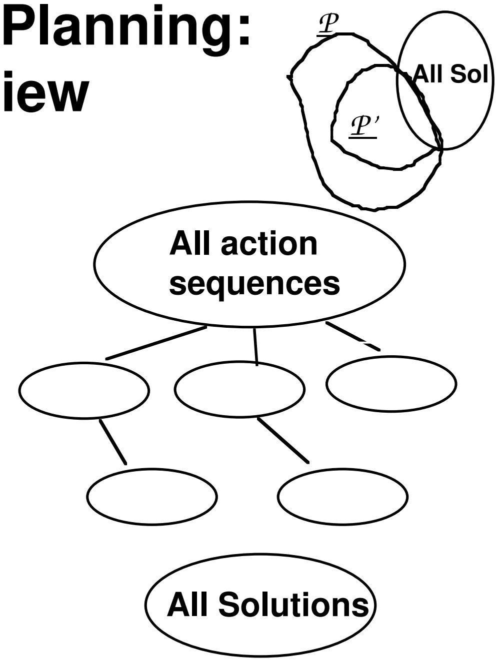
So, let us move on to the formal foundations of refinement planning -- including the syntax and semantics of partial plans, the refinement strategies and finally the generalized algorithm templates for refinement planning.

Refinement Planning: Overview

Search in the space of sets of action sequences

Each set is represented by a partial plan

Refinements narrow the sets by adding constraints to partial plans



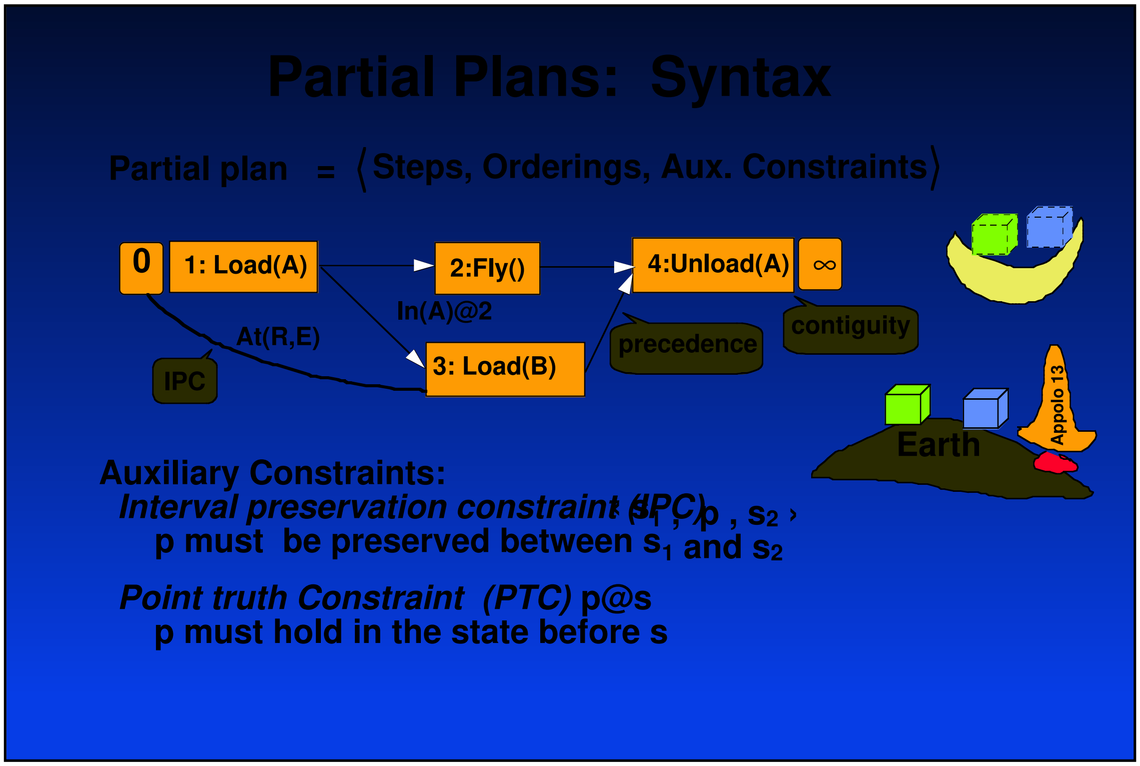
Refinement planning can be thought of as a process of narrowing down the set of all actions sequences by eliminating the sequences that cannot be solutions to the problem.

Sets of action sequences are represented intensionally by a set of constraints called partial plans. The action sequences represented by a partial plan are called its candidates.

A refinement operation narrows the candidate set of a partial plan by adding constraints to it..

If no solutions are eliminated in this process, we will eventually progress towards set of all solutions. Termination can occur as soon as we can pick up a solution using a bounded time operation.

To make these ideas precise, we shall now look at the syntax and semantics of partial plans and refinement operations.



A partial plan can be seen as any set of constraints that together delineate which action sequences belong to the plan's candidate set and which do not.

One representation that is sufficient for our purposes models partial plans as a set of steps, ordering constraints between the steps, and auxiliary constraints. Each plan step corresponds to an action. There are two types of ordering constraints -- precedence and contiguity constraints. The latter require that two steps come immediately next to each other.

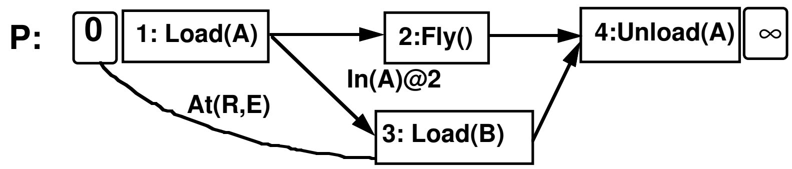
Auxiliary constraints involve statements about truth of certain conditions over time intervals. These come in two important types -- *interval preservation constraints* which require preservation of a condition over an interval, and *point truth constraints* that require the truth of a condition at a particular time point.

Here is an example plan from our rocket domain in this representation. The steps 0 and 1 are contiguous, 2 precedes 4, and the condition At(R,E) must be preserved between 0 and 3.

Finally, the condition In(A) must hold in the state preceding the execution of step 2. (*This is in addition to the constraint that all preconditions of an action must hold in the state preceding the action.*)

Partial Plans: Semantics

Candidate is any action sequence that
-- contains actions corresponding to all the steps,
-- satisfies all the ordering and auxiliary constraints



Candidates (∈ «P»)	Non-Candidates (∉ «P»)
[Load(A),Load(B),Fly(),Unload(A)]	[Load(A),Fly(),Load(B),Unload(B)]
Minimal candidate. Corresponds to the safe linearization [01324∞]	Corresponds to unsafe linearization [01234∞]
[Load(A),Load(B),Fly(), Unload(B),Unload(A)]	[Load(A),Fly(),Load(B), Fly(),Unload(A)]

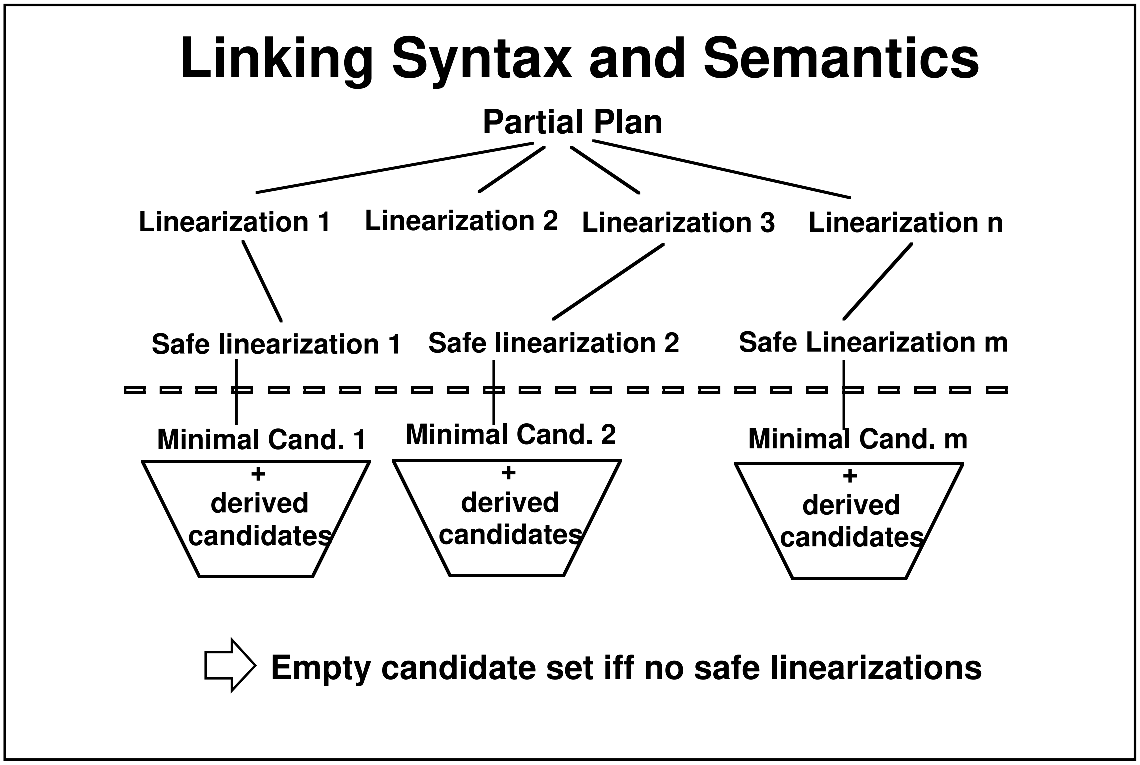
The semantics of the partial plans are given in terms of candidate sets. An action sequence belongs to the candidate set of a partial plan if it contains the actions corresponding to all the steps of the partial plan, in an order consistent with the ordering constraints on the plan, and it also satisfies all auxiliary constraints.

For the example plan shown here, the sequences on the left are candidates while those on the right are non-candidates. Notice that the candidates may contain more actions than are present in the partial plan.

Candidates that only contain the actions in the plan are called "minimal candidates". These correspond to the syntactic notion of safe linearizations.

Safe linearizations are linearizations (or topological sorts) of the plan steps that also satisfy the auxiliary constraints. The linearization 0-1-3-2-4-infty is a safe one while the linearization 0-1-2-3-4-infty is not (since step 2 will violate the IPC on At(R,E).

The sequences on the right are non-candidates because both of them fail to satisfy the auxiliary constraints



Here then is the connection between the syntax and semantics of a partial plan. Each partial plan has at most exponential number of linearizations, some of which are safe with respect to the auxiliary constraints.

Each safe linearization corresponds to a minimal candidate of the plan. Thus, there are at most exponential number of minimal candidates. A potentially infinite number of additional candidates can be derived from each minimal candidate by padding it with new actions without violating auxiliary constraints.

Thus, a plan with no safe linearizations will have an empty candidate set.

Refinements add new constraints to a partial plan. They thus simultaneously shrink the candidate set of the plan, and increase the length of its minimal candidates.

Thus, one incremental way of exploring the candidate set of a plan for solutions is to check through its minimal candidates after refinements.

Refinement Strategies

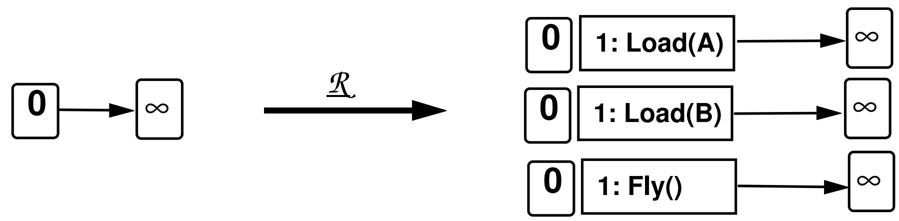
§ A ***plan set*** $\underline{\mathcal{P}}$ is a set of partial plans $\{P_1, P_2 \ldots P_m\}$

✧A refinement strategy $\underline{\mathcal{R}}$: $\underline{\mathcal{P}} \twoheadrightarrow \underline{\mathcal{P}'}$ («$\underline{\mathcal{P}'}$» a subset of «$\underline{\mathcal{P}}$»)

– $\underline{\mathcal{R}}$ is complete if «$\underline{\mathcal{P}'}$» contains all the solutions of «$\underline{\mathcal{P}}$»

– $\underline{\mathcal{R}}$ is progressive if «$\underline{\mathcal{P}'}$» is a proper subset of «$\underline{\mathcal{P}}$»

– $\underline{\mathcal{R}}$ is systematic if components of $\underline{\mathcal{P}'}$ don't share candidates



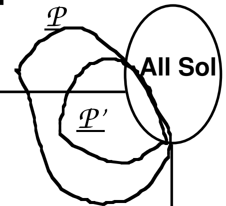
We will now formally define a refinement strategy. Refinement strategies operate on sets of partial plans. We thus define a plan-set as a set of partial plans, with the understanding that the candidate set of the planset is the union of the candidate sets of its component plans.

A refinement strategy R maps a plan set P to another plan set P′ such that the candidate set of P′ is a subset of the candidate set of P. R is said to be complete if P′ contains all the solutions of P. It is said to be progressive if the candidate set of P′ is a strict subset of the candidate set of P. It is said to be systematic if no action sequence falls in the candidate set of more than one component of P′.

Completeness ensures that we don't lose solutions by the application of refinements. Progressiveness ensures that refinement narrows the candidate set. Systematicity ensures that we never consider the same candidate more than once.

At the bottom is an example refinement, for our rocket problem, which takes the null plan, corresponding to all action sequences and maps it to a plan set containing 3 components. (In this case, the refinement is complete since no solution can start with any other action, progressive since it eliminated action sequences beginning with unload(A) etc, and systematic since all the candidates of the three components will have different prefixes.)

Refinement Planning Template



Refine ($\mathcal{P}$: Plan set)

0*. If «$\mathcal{P}$» is empty, Fail.
1. If a minimal candidate of $\mathcal{P}$ is a solution, return it. End
2. Select a refinement strategy $\mathcal{R}$
Apply $\mathcal{R}$ to $\mathcal{P}$ to get a new plan set $\mathcal{P}'$
3. Call Refine($\mathcal{P}'$)

--Termination ensured if $\mathcal{R}$ is complete and progressive

-- Solution extraction (step 2) involves checking exponentially many minimal candidates
-- Can be cast as propositional model-finding (satisfaction)

We are now in a position to present the general refinement planning template. It has three main steps.

If the current plan-set has an extractable solution -- which is found by inspecting its minimal candidates, we terminate.

If not, we select a refinement strategy R and apply it to the current plan set to get a new plan set.

As long as the selected refinement strategy is complete, we will never lose a solution. As long as the refinements are progressive, for solvable problems, we will eventually reach a planset one of whose minimal candidates will be a solution (the figure on the top right illustrates this).

The solution extraction process involves checking an exponential number of minimal candidates (corresponding to safe linearizations). This can be cast as a model-finding or satisfaction process.

Some recent planners like Graphplan and Satplan can be seen as instantiations of this general refinement planning template. However, most earlier planners use a specialization of this template that we shall discuss next.

Combining Refinement with Search

Motivation: Push complexity of solution extraction into search space

Refine ($\underline{\mathcal{P}}$: Plan)

0*. If «$\underline{\mathcal{P}}$» is empty, Fail.
1. If SOL($\underline{\mathcal{P}}$) returns a solution, terminate with success.
2. Select a refinement strategy $\underline{\mathcal{R}}$
Apply $\underline{\mathcal{R}}$ to $\underline{\mathcal{P}}$ to get a new plan set $\underline{\mathcal{P}'}$
3. Non-deterministically select a component $\underline{\mathcal{P}'_i}$ of $\underline{\mathcal{P}'}$
Call Refine($\underline{\mathcal{P}'_i}$)

State-space,
Plan-space,
HTN,
Tractability

The algorithm template in the previous slide does not have any search in the foreground All the search is pushed into the solution extraction function.

It is possible to add "search" to the refinement process in a straightforward way. The usual motivation for doing this is to make the solution construction phase cheaper. (*Afterall, checking for solution in a single partial plan is cheaper than searching for a solution in a plan set*.)

The algorithm template shown here introduces search into refinement planning. It is worth noting the two new steps that made their way. First, the components of the plan set resulting after refinement are pushed in to the search space, and are handled separately. We thus confine the application of refinement strategies to single plans.

Notice that this search process will have backtracking even for complete refinements since we do not know which of the individual components contain a solution.

Second, once we work on individual plans, we can consider solution extraction functions that are cheaper than looking at all the minimal candidates.

This simple algorithm template forms the main idea behind all existing refinement planners. These planners differ in terms of the specific refinement strategies they use in step 2.

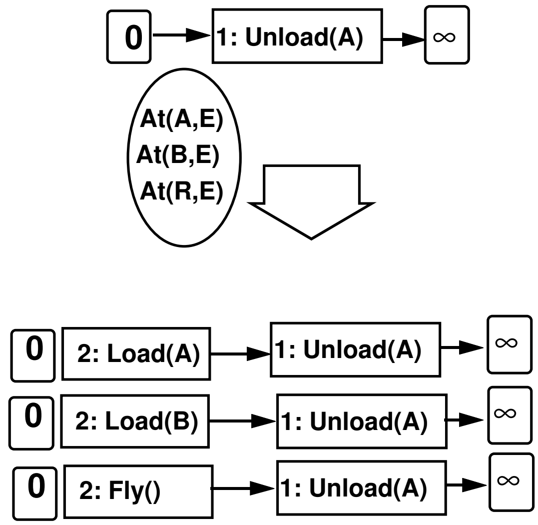
Overview

- Classical planning problem
- Refinement Planning: Formal Framework
- Existing refinement strategies
  - State-space refinements
  - Plan-space refinements
    - Using non-primitive actions
  - Tractability refinements
- Tradeoffs in Refinement Planning
- Scaling up Refinement Planning
- Conclusion

We will now concentrate on the specific refinement strategies that are used in the second step of the refinement planning template. There are broadly three types-- state-space, plan space and tractability refinements

Forward State-space Refinement

- Grow plan prefix by adding applicable actions
  - Complete
    - consideration of all executable prefixes
  - Progressive
    - elimination of unexecutable prefixes
  - Systematic
    - each component has a different prefix



Forward state space refinement involves growing the plan prefix by considering all actions -- whether in the plan or in the library -- whose preconditions hold in the state of the world expected at the end of the prefix.

By prefix we mean the steo 0 and any steps contiguous to.

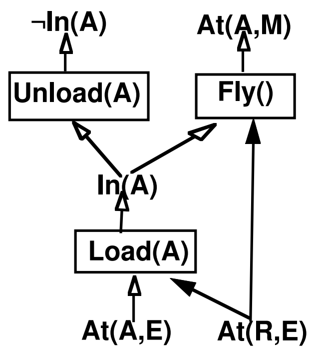
Here the prefix contains single step 0, and the expected state of the world is the same as the initial state. Three actions are applicable in this state, and accordingly forward state space refinement produces a plan set with three components.

It is easy to see that forward state space refinement is complete as it considers all exectuable prefixes and every solution must have an executable prefix. Similarly, it eliminates sequences with non-exectuable prefixes, thus giving progressiveness, and its components have candidates with different prefixes, thus giving systematicity.

Goal-directed State-space Refinements

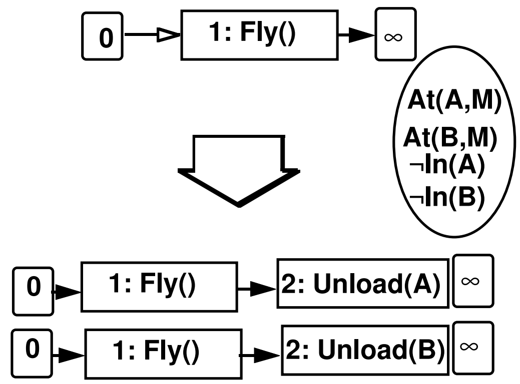
Force FSR to consider only *relevant* actions

-- Use subgoaling structure (operator graphs, Greedy regression graphs)



Consider Backward State-space refinements

-- Grow suffixes by adding “useful” actions



(Fikes & Nilsson, 1972; Veloso et. al. 1992, McDermott, 1996)

FSR refinement, as stated, considers all actions executable in the state after the prefix. In real domains, there may be a large number of applicable actions, very few of which are relevant to the top level goals of the problem. We can make the state-space refinements goal-directed in one of two ways.

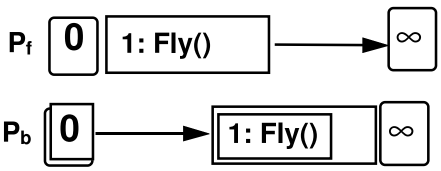
First, we can force FSR to consider only those actions which are going to be relevant to the top level goals. The relevant actions can be recognized by examining a subgoaling tree of the type shown below. Here the top level goal can be helped by the actions Unload(A), and Fly, and their preconditions in turn are helped by the action Load(a). This idea is known as Means-ends-analysis and has been used by one of the first planners called STRIPS. More recently, a powerful heuristic for focusing means-ends-analysis has been proposed by McDermott.

The second way of making state-space refinements goal directed is to consider growing suffixes by applying actions in the backward direction. Here, the state of the world before the suffix, which can be seen as the set of weakest conditions for ensuring goal satisfaction, are shown in the brown oval. Two actions, Unload(A) and Unload(B) are useful in that they can give some of the conditions of the state, without violating any others. Fly action is not useful since it can violate the ¬In(x) goals. (*This analysis can be done by comapring the action effects to the regressed state at the beginning of the suffix*).

Position, Relevance and Commitment

FSR and BSR must commit to both position and relevance of actions

- **+ Gives state information**
- **- Leads to premature commitment**



Plan-space refinement (PSR) avoids constraining position

- **+ Reduces commitment**
- **- Increases plan-handling costs**

The state space refinements have to decide both the "position" and "relevance" of a new action to the overall goals. Often times, we may know that a particular action is relevant, but not know where exactly in the eventual solution it will come. (*For example, we know that a fly action is going to be present in the solution for the rocket problem, but do not know exactly where in the plan it will occurs)*

In such cases, it helps to introduce an action into the plan, without constraining its absolute position. This is what the plan-space refinement does. Of course, the disadvantage of not fixing the position is that we will not have state information. This makes plan-handling costly.

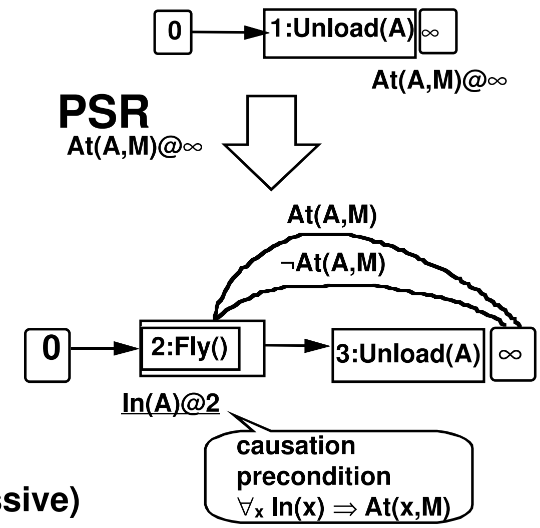
The difference between FSR,BSR and PSR has traditionally been understood in terms of least commitment, which in turn is related to candidate set size. Plans with precedence relations have larger candidate sets than those with contiguity constraints. For example, it is easy to see that although all three plans shown here contain the single fly action, the solution sequence load(a),load(b), fly, unload(A), unload(B), belongs only to the candidate set of the plan with precedence constraints.

Since each search branch corresponds to a component of the plan set produced by the refinement, planner using state-space refinements are more likely to backtrack from a search branch.

One other point worth noting is that the backtracking is an artificat of splitting plan set components into the search space. Since all refinements are complete, backtracking will never be required if we had worked with plan sets without splitting.

Plan-space Refinement

Goal selection:
  Select a precondition
Establishment:
  Select a step and make it give the condition
De-clobbering:
  Force intervening steps to preserve the condition
Book-keeping: (Optional)
  Add IPCs to preserve the establishment
  ⇒ Systematicity



(PSR is complete, and progressive)

(Sacerdoti, 1972; Pednault, 1988; McAllester, 1991)

This brings us to the specifics of plan-space refinement. A plan space refinement starts by picking any precondition of any step in the plan, introducing step,ordering constraints to ensure that the precondition is provided by some step and is preserved by the intervening steps.

In this example, we pick the precondition At(A,M) of the last step (which stands for the top level goal). We add the new step Fly() to support this condition. In order to force Fly to give At(A,M), we add the condition In(A) as a precondition to it. This is called a causation precondition. At this point we need to make sure that any step possibly intervening between Fly and the last step preserves At(A,M). In this example, only Unload(A) intervenes and it does preserve the condition, so we are done.

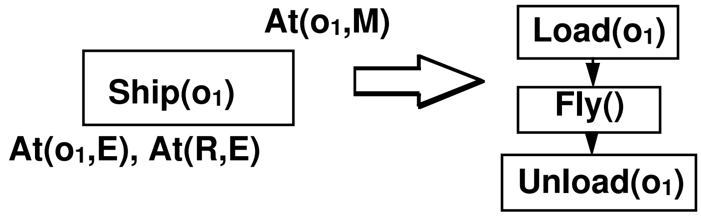
If we had a different action that can delete the condition, we must handle this conflict by either putting the action out of the interval between Fly and the last step, or adding preservation preconditions to the step to force it to preserve At(A,M).

A third optional part of plan-space refinement involves adding IPCs to preserve this establishment during subsequent refinement oprations (when new actions may come between Fly and the last step). This is done by adding either one or both of the Interval preservation constraints At(A,M) and ¬At(A,M).

If we add both,we can show that the refinement is systematic, despite the fact that the positions of none of the steps are fixed.

PSR has many instantiations depending on whether bookkeeping strategies are used.

PSR with non-primitive actions



- **Not all solutions are created equal.**
- **How do we bias search towards desirable solutions?**
  - **Use non-primitive actions during establishment, and replace them later with user supplied schemas**
    - **+ Reduction schemas naturally available**
    - **+ Communicate user preferences**
    - **—Tricky to ensure correctness of reduction schemas**

(Sacerdoti, 1972; Tate, 1978; Wilkins, 1985; Erol, 1995; Barrett, 1995; Chien, 1995)

The refinements that we have seen till now consider all action sequences that reach the goal state as equivalent. In many domains, the users may have significant preferences among the solutions. For example, when I call my travel-agent, I may not want a travel plan that involves hitch-hiking to travel from Phx to Portland. The question is how do we communicate these biases to the planner such that it will not waste any time progressing towards bad solutions?

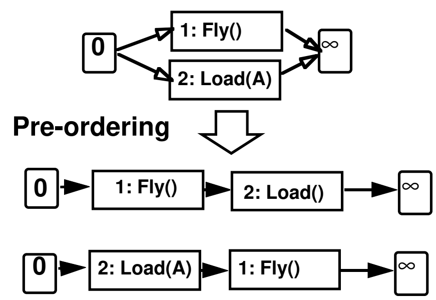
One natural way turns out to be to introduce non-primitive actions, and restrict their reduction to primitive actions through user-supplied reduction schemas. Here the non-primitive action SHIP has a reduction schema that translates it to a plan-fragment containing three actions. Typically, there may be multiple possible legal reductions for a non-primitive action. The reduction schemas restrict the planners access to the primitive actions and thus stop progress towards undesirable solutions.

For this method to work however, we do need the user to provide us reduction schemas over and above domain dynamics. The experience of people who used refinement planners for large scale applications has been that in most real domains, where humans routinely build plans, such reduction knowledge is available naturally. Of course, checking whether the reduction schemas are "correct" can be a tricky proposition.

Tractability Refinements

Aim: Reduce plan handling costs by splitting (implicit) disjunction into the search space

- **Reduce number of linearizations**
  - **Pre-ordering**
  - **Pre-positioning**
- **Make all linearizations safe**
  - **Pre-satisfaction**
    - **Resolve threats to auxiliary constraints**
- **Reduce uncertainity in action identity**
  - **Pre-reduction**
    - **Replace a non-primitive action with the help of reduction schemas**



All the refinements we have looked at until now are progressive in that they narrow the candidate set. Many planners also use a variety of refinements that are not progressive. The motivation for their use is to reduce the plan handling costs further by splitting any implicit disjunction in the plan into the search space -- we thus call them tractability refinements.

We can classify the tractability refinements into three categories. The first attempt to reduce the number of linearizations of the plan. In this category, we have **pre-ordering refinements** which order two unordered steps, and **pre-positioning refinements** which constrain the relative position of two steps. Pre-ordering refinements are illustrated in the figure to the right -- the single partially ordered plan above is converted into two totally ordered plans below.

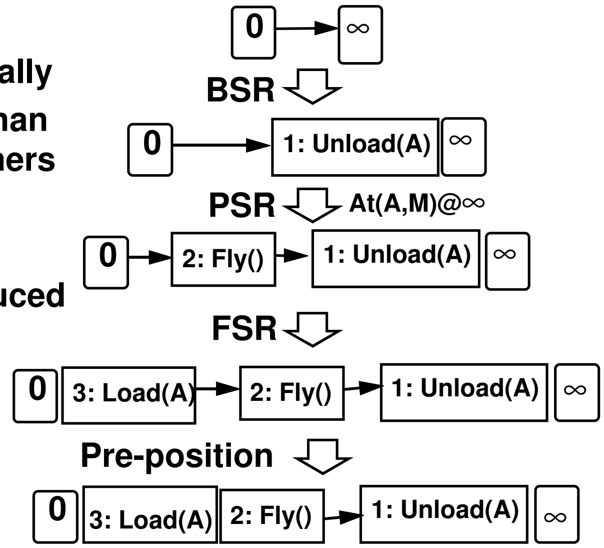
The second category of tractability refinements attempts to make all linearizations safe with respect to auxiliary constraints. Here we have Pre-satisfaction refinements, which split a plan in such a way that a given auxiliary constraint is satisfied by all linearizations of the resulting components.

The third category attempts to reduce uncertainity in the action identity. An example is pre-reduction refinement, which converts a plan containing a non-primitive action into a set of plans each containing a different reduction of that non-primitive action.

It is interesting to note that most of the prominent differences between existing algorithms boil down to differences in the use of tractability refinements.

Interleaving Refinements

- Combine different refinements opportunistically
  - Can be more efficient than single-refinement planners
  - Refinement selection criteria?
    - » # Components produced
    - » "Progress" made



(Kambhampati & Srivastava, 1995)

One of the advantages of the treatment of refinement planning that I have presented to you is that it naturally allows for interleaving of a variety of refinement strategies in solving a single problem. Here is an example of solving our rocket problem with the use of several refinements-- we start with backward state-space, then plan-space, then forward state-space and then a pre-position refinement.

Based on the specific interleaving strategy used, we can device a whole spectrum of refinement planners, which differ from the existing single refinement planners. Our empirical studies show that interleaving refinements this way can lead to superior peformance over single-refinement planners.

It must be noted however that the issue of how one selects a refinement is largely open. W e have tried refinement selection based on the number of components produced by each refinement, or the amount of narrowing of candidate set each refinement affords, with limited success.

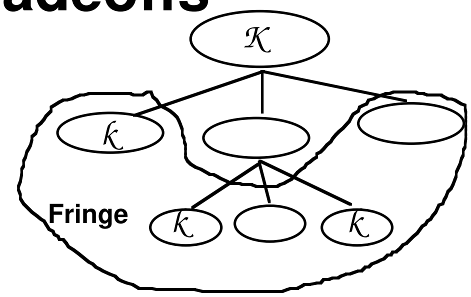
Overview

- Classical planning problem
- Refinement Planning: Formal Framework
- Existing refinement strategies
- Tradeoffs in Refinement Planning
  - Asymptotic tradeoffs and empirical evaluation
  - Selecting a refinement planner
- Scaling up Refinement Planning
- Conclusion

We have described a parameterized refinement planning template that allows for a variety of specific algorithms depending on which refinement strategies are selected and how they are instantiated.

We shall now attempt to understand the tradeoffs governing some of these choices, and see how one can go about choosing a planner, given a specific population of problems to solve.

Asymptotic Tradeoffs



Size of explored search space : $\mathcal{F} = \dfrac{p^{d} * \mathcal{K} * \rho}{k} = O\left(\mathbf{b}^{\mathbf{d}}\right)$

Time complexity: $\mathcal{T} = \mathcal{C} * \mathcal{F}$

Effect of...

Least commitment:	$\mathcal{C}\uparrow\ \mathcal{F}\downarrow$	$(k\uparrow)$
Tractability Refinements:	$\mathcal{C}\downarrow\ \mathcal{F}\uparrow$	$(\mathbf{b}\uparrow)$
Protection/Bookkeeping:	$\mathcal{C}\uparrow\ \mathcal{F}\downarrow$	$(\rho\downarrow)$
Eager Solution extraction :	$\mathcal{C}\uparrow\ \mathcal{F}\downarrow$	$(\mathbf{d}\downarrow)$

- k : **Avg cand set size of fringe plan**
- $\mathcal{F}$: **Fringe size**
- ρ : **Redundancy factor** (≥ 1)
- p : **Progress factor** (≤ 1)
- $\mathcal{K}$: **Cand set size of null plan**
- **b : Branching factor (# comp)**
- **d : depth (# refinements)**
- $\mathcal{C}$: **Plan handling costs**

Let us start with an understanding of the asymptotic trade-offs. To do this, we shall use an estimate of the search space size in terms of properties of fringe nodes.

Suppose big-K is the total number of action sequences (below a certain length, say). Let F be the number of nodes on the fringe of the search tree generated by the refinement planner, and small-k the average number of candidates in each of the plans on the fringe. Let rho be the number of times a given action sequence enters the candidate sets of fringe plans, and p be the progress factor -- the fraction by which candidate set narrows each time a refinement is done. We then have F = p^d * K * rho /k. Since F is approximately the size of the search space, the time complexity is C times F, where C is the cost of plan handling. This formula can be used to understand the asymptotic tradeoffs.

For example, using refinement strategies with lower commitment leads to plans with higher candidate set sizes and thus reduces F, but it can increase C. Using tractability refinements increases b and thus increases F, but may reduce C. The protection strategies reduce the redundancy factor, and thus reduce F. But, they may increase C since protection is done by adding additional constraints.

While instructive, this analysis does not make conclusive predictions on practical performance since the latter depends on the relative magnitudes of changes in F and C. For this, we look at empirical evaluation.

Empirical Evaluation of Tradeoffs

- **Stronger tractability refinements improve performance if increased linearization reduces the #of simple establishments**
  - **Happens in domains with *high frequency conditions***
- **Strong protection strategies improve performance only when the solution density is low**
  - **Planner is forced to explore a significant part of its search space**

(Kambhampati, Knoblock and Yang, 1995)

The parameterized and unified understanding of refinement planning allows us to ask specific questions about the utility of specific design choices, and answer them through normalized empirical comparisons.

Here, we look at two choices -- use of tractability refinements and protection strategies -- since many existing planners differ along these dimensions.

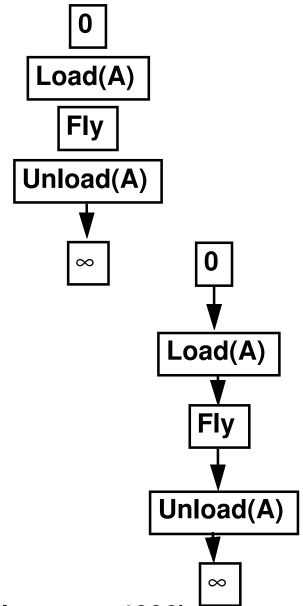
Empirical results show that tractability refinements lead to reductions in search time only when the additional linearization they cause, has the side-effect of reducing the number of establishment possibilities significantly. This happens in domains where there are conditions that are asserted and negated by many actions.

Results also show that protection strategies have an effect on performance only in the cases where solution density is so low as to make the planner look at the full search space.

In summary, for problems with normal solution density, peformance differentials between planners are often attributable to differences in tractability refinements.

Subgoal Interactions and Planner Selection

- **Every refinement planner R can be associated with a class $\mathcal{C}_R$ of partial plans**
- **G_1 and G_2 are trivially serializable w.r.t. a plan class $\mathcal{C}$ if every subplan $P_{Gi} \in \mathcal{C}$ is extendable to a plan for solving both.**
  - **commitment$\downarrow \Rightarrow$ trivial serializability $\uparrow$**
  - **commitment $\downarrow \Rightarrow$ plan handling cost$\uparrow$**
- ⇨ **Select the planner producing the class of <u>highest</u> commitment plans w.r.t. which most problems are trivially serializable**



(Korf, 1987; Barrett & Weld, 1994; Kambhampati, Ihrig and Srivastava, 1996)

Let us now turn to the general issue of selecting among refinement planners, given a set of problems. If we make the reasonable assumption that planners will solve a conjunctive goal problem by solving the subgoals serially, we can answer this question in terms of the subgoal interactions.

A subplan for a goal is a partial plan all of whose linearizations will execute and achieve the goal. On the right are two subplans for the AT(A,M) goal in the rocket problem.

Every refinement planner R can be associated with a class CR of plans it is capable of producing. For example, for the goal At(A,M) in the rocket problem. a planner using purely state-space refinements will produce the prefix plans of the sort shown on top while a pure plan-space planner will produce an elastic plan of the sort shown in the bottom.

The key question is whether a subplan in the given plan class is likely to be extended to a solution for the whole problem. Two goals G1 and G2 are said to be trivially serializable with respect to a class of plans C, if every subplan for solving one goal belonging to C is extendable to a subplan for solving both goals. Turns out that the level of commitment inherent in a plan class is a very important factor in deciding trivial serializability.

Clearly, the lower the commitment of plans in a given class, the higher the chance of trivial serializability. THe lower this commitment, the higher also is the cost of handling plans, in general.

Thus, a heuristic is to select the refinement planner with the highest commitment, and with respect to whose class of plans, most goals are trivially serializable. Empirical studies show this to be an effective strategy. (*Least commitment is not the only factor*)

Overview

- Classical planning problem
- Refinement Planning: Formal Framework
- Existing refinement strategies
- Tradeoffs in Refinement Planning
- Scaling up Refinement Planning
  - Customizing planners to domains
  - Use of disjunctive representations
- Conclusion

Although refinement planning techniques have been applied to complex real world problems like beer-brewing, electrical power production, and space craft assembly, their wide-spread use has been inhibited by the fact that most existing planners scale up poorly when presented with large problems. There has thus been a significant emphasis on techniques for improving their performance.

Let me now survey the directions that are being taken--one of these involves improving performance by customizing the planner's behavior to the problem population, and the second involves using disjunctive representations.

Improving Performance through Customization

- Biasing the search with control knowledge acquired from experts
  - Non-primitive actions and reduction schemas
  - Automated synthesis of customized planners
    - Combine formal theory of refinement planning and domain-specific control knowledge
- Use of learning techniques
  - Search control rule learning
  - Plan reuse
    - Learning /planning sessions at AAAI/IJCAI

(Minton, 1988; Katukam and Kambhampati, 1994; Veloso, 1994; Ihrig, 1996; Srivastava,

Customization can be done in a variety of ways. The first is to bias the search of the planner with the help of control knowledge acquired from the user. As we discussed earlier, non-primitive actions and reduction schemas are used for the most part to support such customization in the existing planners. There is now more research on the protocols for acquring and analyzing reduction schemas.

There is evidence that not all control knowledge of the users is available in terms of reduction schemas. In such cases, incorporating the control knowledge into the planner can be very tricky. One intriguing idea is to "fold in the control knowledge" into the planner by automatically synthesizing planners from domain specification, and the declarative theory of refinement planning using interactive software synthesis tools. We have started a project on implementing this approach using kestrel interactive software synthesis system, and the preliminary results have been promising.

Another way of doing customization is to use learning techniques and make the planner learn from its failures and successes. The object of learning may be acquisition of search control rules that advise the planner what search branch to pursue or the acquisition of typical planning cases, which can then be instantiated and extended to solve new problems. This is a very active area of research and a sampling of papers can be found in the machine learning sessions at AAAI and IJCAI.

Reducing Search through Disjunctive Representations

- What if we handle plan sets without splitting?
  - + Reduced commitment
  - + Separation of action selection and action sequencing
  - – Unwieldy plan sets
    - » Use disjunctive representations
  - – Transfer of complexity to solution-extraction
    - » Use efficient SAT solvers
    - » Use incremental constraint propagation

Another way of scaling up refinement planners to directly address the question of search space explosion. Much of this explosion is due to the fact that all existing planners reflexively split the plan set components into search space. We have seen earlier that this is not required for completeness of refinement planning.

So, let us look at what happens if we don't split plan sets. Of course, we reduce the search space size and avoid the premature commitment to specific plans. We also separate the action selection and action sequencing phases, so that we can apply scheduling techniques for the latter.

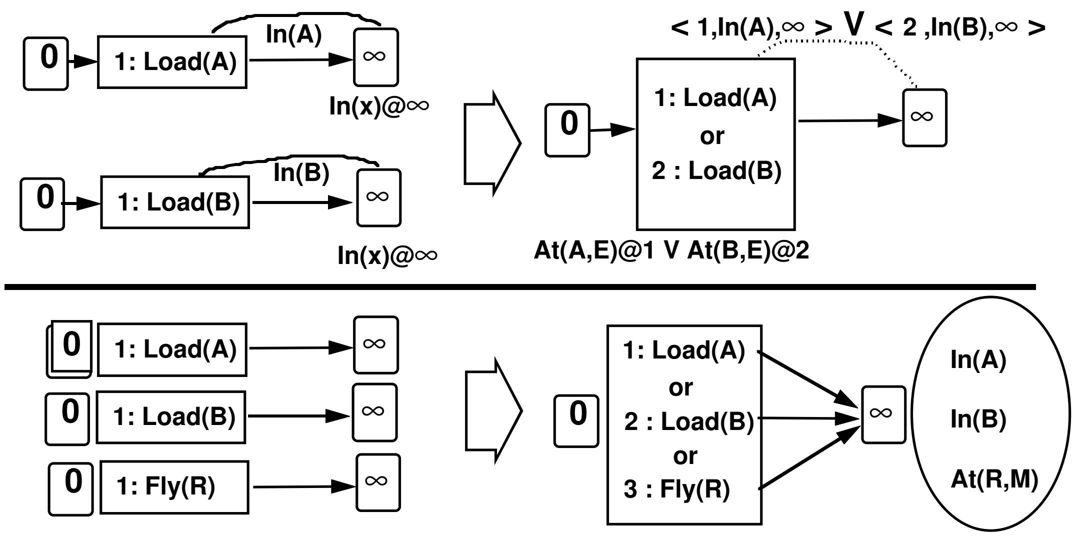
There can be two potential problems however. First off, keeping plan sets together may lead to very unwieldy data structures. The way to get around this is to "internalize" the disjunction in the plan sets so that we can represent them more compactly. The second potential problem is that we may just be transfering the complexity from one place to another -- from search space size to solution extraction. This may be true. However, there are two reasons to believe that we may still win.

First, as we mentioned earlier, solution extraction can be cast as a model-finding activity, and there have been a slew of very efficient search strategies for propositional model finding. Second, we may be able to do even better by using constraint propagation techniques that simplify plans and reduce refinement possibilities.

Let me illustrate these ideas.

Disjunctive Representations

Allow disjunctive step, ordering and auxiliary constraints

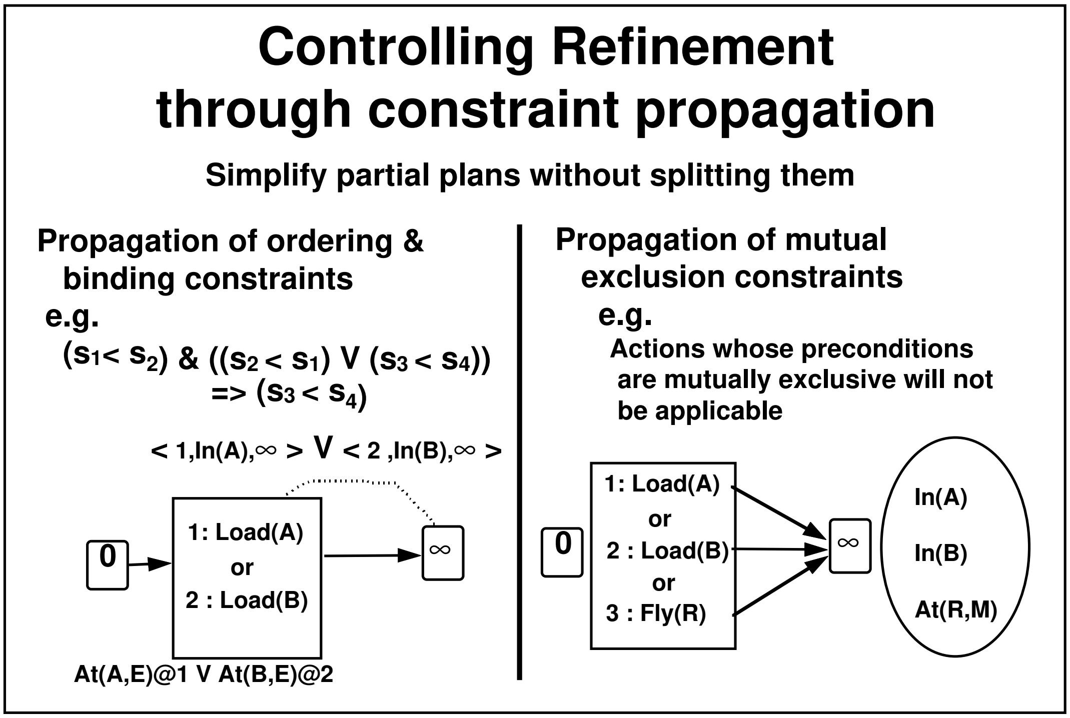


The general idea of disjunctive representations is to allow disjunctive step, ordering, and auxiliary constraints into the plan. Here are two examples that illustrate the compaction we can get through them.

The two plans on the top left corner can be compacted by using a single disjunctive step constraint, a disjunctive precedence constraint, a disjunctive IPC and a disjunctive PTC.

Similarly, the three plans in the bottom right can be compacted into a single disjunctive step, with disjunctive contiguity constraints.

Candidate set semantics can be given naturally from the interpretation of disjunctive constraints.



Disjunctive representations clearly lead to a significant increase in the cost of plan handling. For example, in the left plan, we don't know whether steps 1 or 2 or both will be present in the eventual plan. Thus we don't know whether we should work on At(A,E) precondition or the At(B,E) precondition. Similarly, in the right hand side we don't know which of the steps will be coming next to 0 and thus we don't quite know what the state of the world will be after the disjunctive step.

At first glance this might look hopeless as the only reasonable way seems to be to split the disjunction into the search space again or depend solely on ultra-efficient solution extraction strategies. However, it turns out that we are underestimating the power of what we do know, and how that knowledge can be used to constrain further refinements.

For example, knowing that either 1 or 2 must precede the last step and give the condition tells us that if 1 doesn't then 2 must. This is an instance of constraint propagation on orderings and reduces the number of establishment possibilities that PSR has to consider at the next iteration.

Similarly, knowing that only 1, 2 or three can be the first steps in the plan tells us that the state after the first step can only contain the conditions In(A), In(B) and At(R,M). This, coupled with the knowledge that both 1 and 3 can't occur in the first step (since their preconditions and effects are mutually exclusive) tells us that the second state may either have In(A) or At(R,M), but not both. This is an instance of propagation of mutual exclusion constraints and can be used to reduce the number of actions considered in the next step by FSR (actions that require both In(A) and At(R,M) can be ignored.

Planning with Disjunctive Representations

- **Disjunction over state-space refinements**
  - **Graphplan (Blum & Furst, 1995)**
  - **SATPLAN (Kautz & Selman, 1996)**
- **Disjunction over plan-space refinements**
  - **Descartes (Joslin & Pollack, 1995)**
  - **UCPOP-D (Kambhampati & Yang, 1996)**

Issues:

1. Tradeoffs in eliminating splitting
-- Splitting may facilitate further constraint propagation

2. Relative support provided by various refinements

3. Integrating solution-extraction and constraint propagation

Lest all this sounds like a pipe-dream, let me hasten to add that in the last year or so several efficient planners have been developed which use disjunctive representations. Some of these are listed here.

There are however many issues that need careful attention. For example, research in constraint satisfaction literature shows that propagation and refinement can have synergistic interactions. (*A case in point is 8-queens problem.* .) This raises the possibility that best planners may be doing controlled splitting of plan sets (rather than no splitting at all) to facilitate further constraint propagation. (*Extent of propagation depends on the amount of shared sub-structure between the disjoined plans.)*

The relative support provided by various types of refinements for planning with disjunctive representations needs to be understood. The old analyses based on least commitment etc. are mostly inadequate when we don't split plan set components.

Finally, the interaction between the solution extraction process and the constraint propagation process needs to be better understood so we can make them synergistic.

Conclusion

- Refinement planning continues to provide the theoretical backbone to the AI Planning techniques
  - Allows a coherent unification of all classical planning techniques
  - Facilitates a clear analysis of tradeoffs
  - Outlines avenues for the development of efficient planning algorithms
  - Provides insights into planning under non-classical assumptions
    - » Simultaneous action: Zeno
    - » Stochastic dynamics: Buridan, SUDO
    - » Partially accessible environments: XII, Cassandra

Let me now conclude by re-iterating that refinement planning continues to provide the theoretical backbone to most of the AI planning techniques.

The general framework I presented in this talk allows a coherent unification of all classical planning technqieus, provides insights into design tradeoffs and also outlines avenues for the development of more efficient planning algorithms.

Perhaps equally important, I believe that a clearer understanding of refinement planning under classical assumptions will provide us valuable insights on planning under non-classical assumptions.

As evidence for this last statement, let me name several non-classical planners described in the literature, whose operation can be understood from refinement planning point of view.

.

Research Issues

- Are there better refinement strategies than FSR, BSR and PSR?
- How should we select between refinement strategies?
- Understanding the trade-offs in planning with disjunctive representations.
- Porting refinement planning techniques and insights to non-classical planning scenarios.

There are a variety of exciting avenues open for further research in refinement planning. Let me list a few of them with the hope that they might inspire a smart graduate student to run off and get busy solving them.

To begin with, we have seen that despite the large number of planning algorithms, there are really only two fundamentally different refinement strategies -- plan-space and state-space ones. It would be very interesting to see if there are novel refinement strategies with better properties. To some extent this might depend on the type of representations we use for partial plans. (*In a paper to be presented at KR-96, Matt Ginsberg describes a partial plan representation and refinement strategy that differs significantly from the state-space and plan-space ones. It will be interesting to see how it relates to the classical refinements.*)

We also do not understand enough about what factors govern the selection of specific refinement strategies.

We need to take a fresh look at tradeoffs in plan synthesis, given the availability of planners using disjunctive representations. Concepts like subgoal interactions do not make too much sense in these scenarios.

Finally, there is a lot to be gained by porting the refinement planning techniques to planning under non-classical scenarios. As I mentioned earlier, already a lot of work has been done in this area, but new developments in classical scenarios continue.