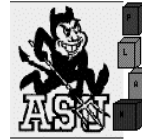


Recent Advances in AI Planning: A Unified View

AAAI-2000 Tutorial
[MA2]



Subbarao Kambhampati
Arizona State University

rao@asu.edu

<http://rakaposhi.eas.asu.edu/planning-tutorial>

URL will have the up to date version

This file contains the slides that were used in my
AAAI-2000 tutorial (July 31st, 2000). They differ only
in minor aspects from the published tutorial notes.

Please note that the slides here include some that I had skipped
over due to lack of time--these include:

- Disjunctive refinement of non-state space plans
- Automated learning of domain-specific control knowledge
- Extensions of classical planning model to stochastic and
partially observable worlds

I would appreciate any feedback you might have on the slides
or the actual tutorial.

Thanks for your interest
Subbarao Kambhampati, 9th August 2000

Planning is hot...

*New people. Conferences. Workshops. Competitions.
Inter-planetary explorations. So, Why the increased interest?*

- ◇ **Significant scale-up in the last 4-5 years**
 - Before we could synthesize about 5-6 action plans in minutes
 - Now, we can synthesize 100-action plans in minutes
 - » Further scale-up with domain-specific control
- ◇ **Significant strides in our understanding**
 - Rich connections between planning and CSP(SAT) OR (ILP)
 - » Vanishing separation between planning & Scheduling
 - New ideas for heuristic control of planners
 - Wide array of approaches for customizing planners with domain-specific knowledge

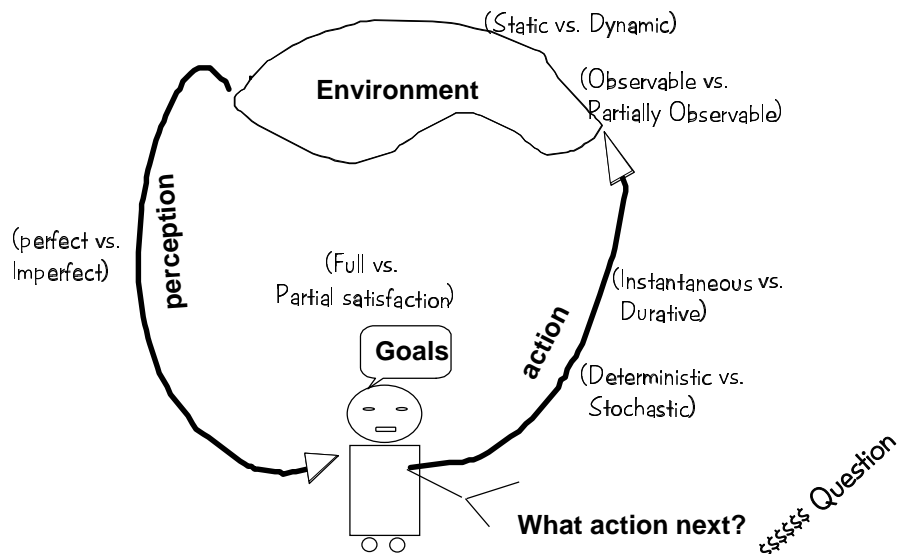
Overview

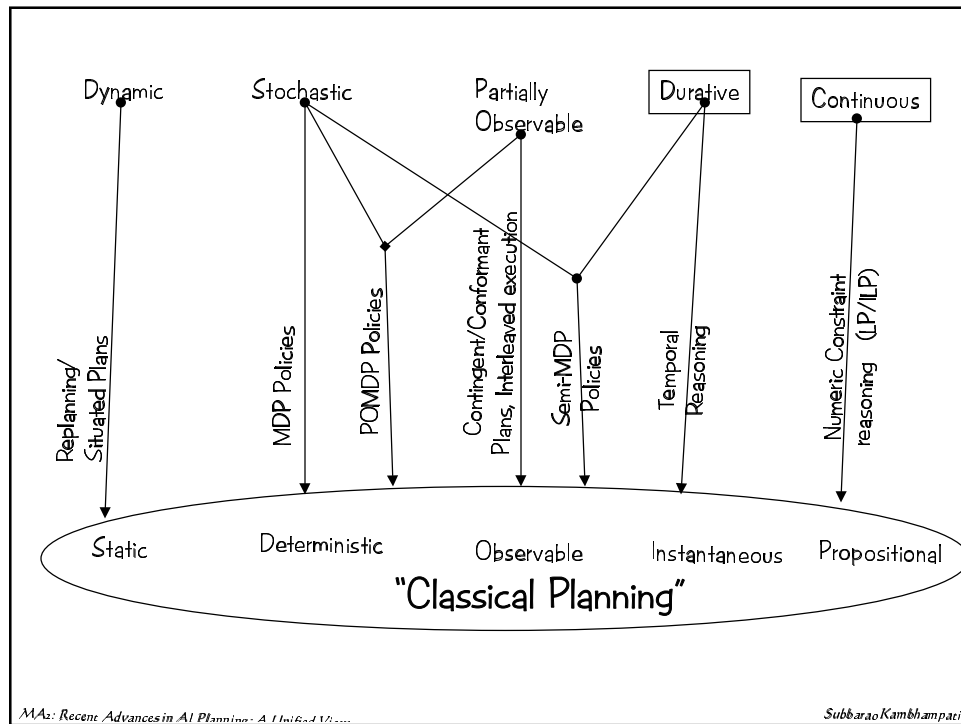
- ◇ **The Planning problem**
 - Our focus
 - Modeling, Proving correctness
- ◇ **Refinement Planning: Formal Framework**
- ◇ **Conjunctive refinement planners**
- ◇ **Disjunctive refinement planners**
 - Refinement of disjunctive plans
 - Solution extraction from disjunctive plans
 - » Direct, Compiled (SAT, CSP, ILP, BDD)
- ◇ **Heuristics/Optimizations**
- ◇ **Customizing Planners**
 - User-assisted Customization
 - Automated customization
- ◇ **Support for non-classical worlds**

Planning : The big picture

- ◇ **Synthesizing goal-directed behavior**
- ◇ **Planning involves**
 - Action selection; Handling causal dependencies
 - Action sequencing and handling resource allocation
 - » typically called **SCHEDULING**
 - Depending on the problem, plans can be
 - » action sequences
 - » or “policies” (action trees, state-action mappings etc.)

The Many Complexities of Planning





Broad Aims & Biases of the Tutorial

AIM: We will concentrate on planning in deterministic, quasi-static and fully observable worlds

Will start with "classical" domains; but discuss handling durative actions and numeric constraints, as well as replanning

Neo-Classical Planning

BIAS: To the extent possible, we shall shun brand-names and concentrate on unifying themes

- Better understanding of existing planners
- Normalized comparisons between planners
- Evaluation of trade-offs provided by various design choices
- Better understanding of inter-connections
- Hybrid planners using multiple refinements
- Explication of the connections between planning, CSP, SAT and ILP

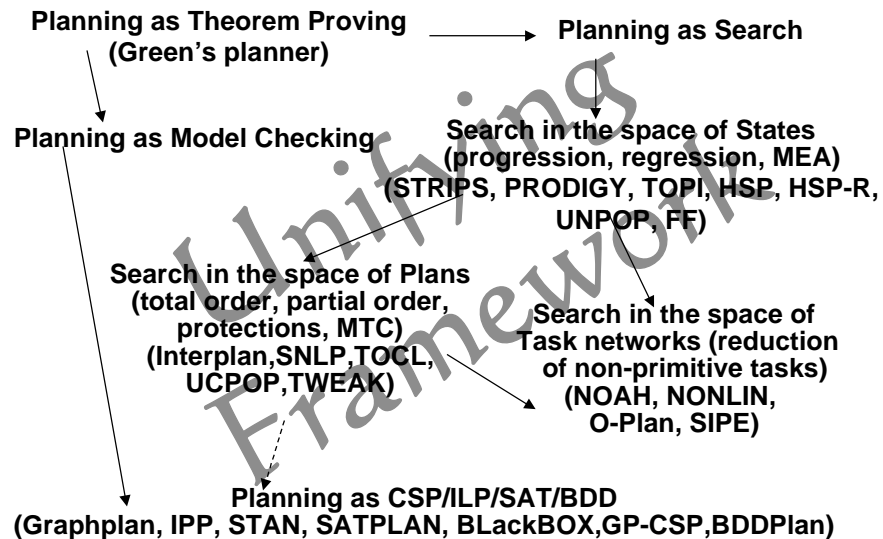
Why Care about “neo-classical” Planning?

- ❖ **Most of the recent advances occurred in neo-classical planning**
- ❖ **Many stabilized environments satisfy neo-classical assumptions**
 - **It is possible to handle minor assumption violations through replanning and execution monitoring**
 - ” This form of solution has the advantage of relying on widely-used (and often very efficient) classical planning technology” Boutilier, 2000
- ❖ **Techniques developed for neo-classical planning often shed light on effective ways of handling non-classical planning worlds**
 - **Currently, most of the efficient techniques for handling non-classical scenarios are still based on ideas/advances in classical planning**

Applications (Current & Potential)

- ❖ **Scheduling problems with action choices as well as resource handling requirements**
 - Problems in supply chain management
 - HSTS (Hubble Space Telescope scheduler)
 - Workflow management
- ❖ **Autonomous agents**
 - RAX/PS (The NASA Deep Space planning agent)
- ❖ **Software module integrators**
 - VICAR (JPL image enhancing system); CELWARE (CELCorp)
 - Test case generation (Pittsburgh)
- ❖ **Interactive decision support**
 - Monitoring subgoal interactions
 - » Optimum AIV system
- ❖ **Plan-based interfaces**

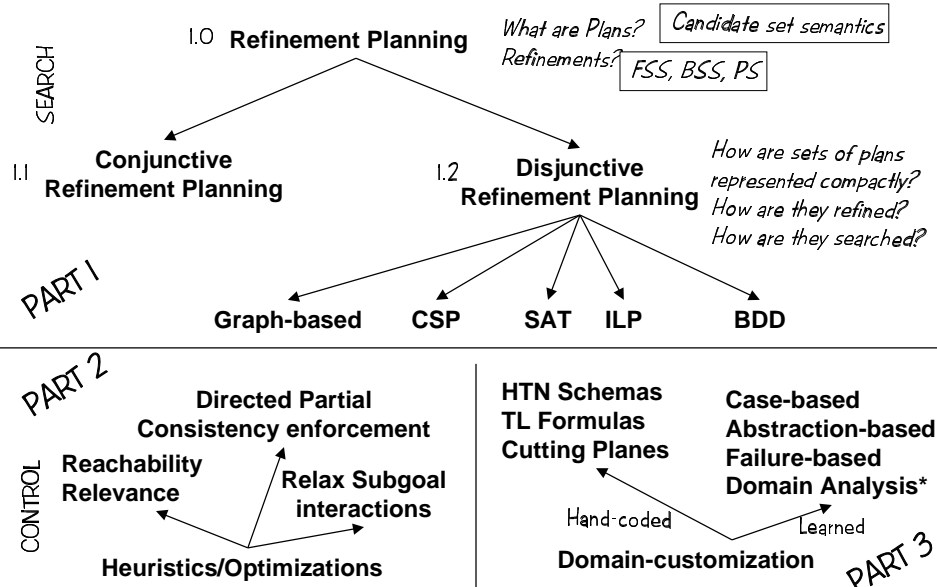
The (too) many brands of classical planners



MA: Recent Advances in AI Planning - A Unified View

Subbarao Kamthampati

A Unifying View



MA: Recent Advances in AI Planning - A Unified View

Subbarao Kamthampati

Modeling Planning Problems: Actions, States, Correctness

Modeling Classical Planning

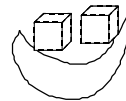
- States are modeled in terms of (binary) state-variables

-- Complete initial state, partial goal state

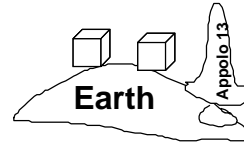
- Actions are modeled as state transformation functions

-- Syntax: ADL language (Pednault)

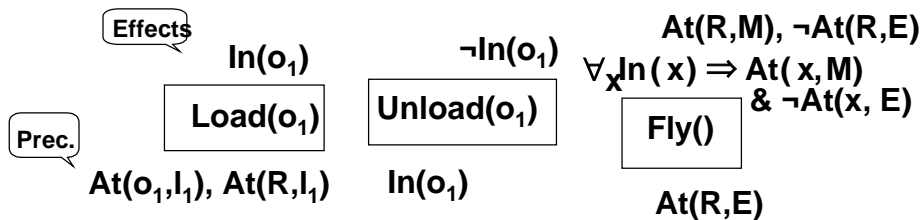
-- $\text{Apply}(A, S) = (S \setminus \text{eff}(A)) + \text{eff}(A)$
(If $\text{Precond}(A)$ hold in S)



$\text{At}(A, M), \text{At}(B, M)$
 $\neg \text{In}(A), \neg \text{In}(B)$



$\text{At}(A, E), \text{At}(B, E), \text{At}(R, E)$



Some notes on action representation

- ❖ STRIPS Assumption: Actions must specify all the state variables whose values they change...
- ❖ No disjunction allowed in effects
 - Conditional effects are NOT disjunctive
 - » (antecedent refers to the previous state & consequent refers to the next state)
- ❖ Quantification is over finite universes
 - essentially syntactic sugaring
- ❖ All actions can be compiled down to a canonical representation where preconditions and effects are propositional
 - Exponential blow-up may occur (e.g removing conditional effects)
 - » We will assume the canonical representation

Action A

Eff: If P then R
If Q then W



Action A1
Prec: P, Q
Eff: R, W

Action A2
Prec: P, ~Q
Eff: R, ~W

Action A3
Prec: ~P, Q
Eff: ~R, W

Action A4
Prec: ~P, ~Q
Eff:

Actions with Resources and Duration

Load(P:package, R:rocket, L:location)

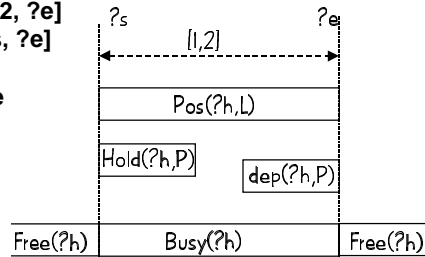
Resources: ?h : robot hand

Preconditions: Position(?h,L) [?s, ?e]
Free(?h) ?s
Charge(?h) > 5 ?s

Effects: holding(?h, P) [?s, ?t1]
depositing(?h,P,R) [?t2, ?e]
Busy(?h) [?s, ?e]
Free(?h) ?e
Charge = .03*(?e - ?s) ?e

Constraints: ?t1 < ?t2
?e - ?s in [1.0, 2.0]

Capacity(robot) = 3



Planning vs. Scheduling

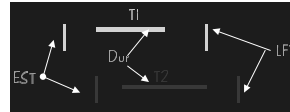
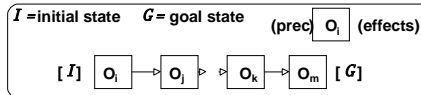
← A Continuum →

Planning

- Initial state & a set of Goals,
 - A library of actions
 - » Preconditions/effects
 - Discrete/Continuous
 - » Resource requirements
- Synthesize a sequence of actions capable of satisfying goals

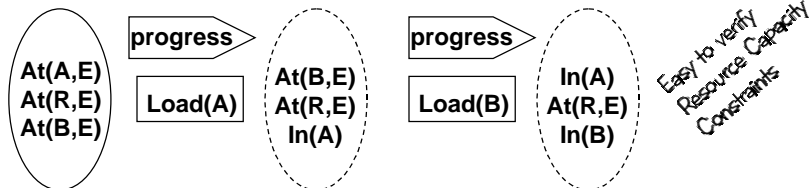
Scheduling

- Set of jobs (may have of tasks in some (partial) order)
 - Temporal constraints on jobs
 - » EST, LFT, Duration
 - Contention constraints
 - » Each task can be done on a subset of machines
- Find start times for jobs that are optimal (wrt make-spans, resource consumption etc)

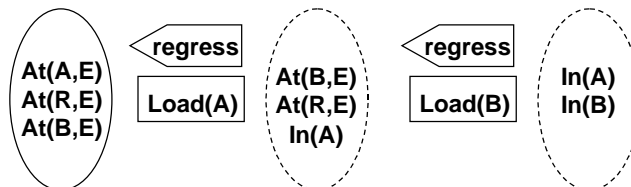


Checking correctness of a plan: The State-based approaches

- ◇ **Progression Proof:** Progress the initial state over the action sequence, and see if the goals are present in the result



- ◇ **Regression Proof:** Regress the goal state over the action sequence, and see if the initial state subsumes the result



Checking correctness of a plan: The Causal Approach

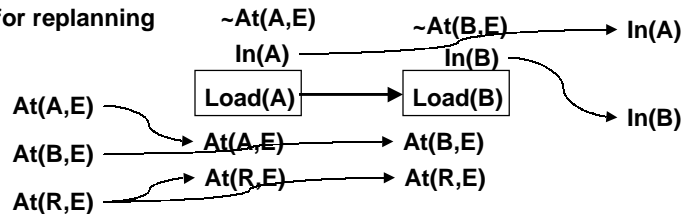
Contd..

◇ **Causal Proof:** Check if each of the goals and preconditions of the action are

- » “established” : There is a preceding step that gives it
- » “unclobbered”: No possibly intervening step deletes it
 - Or for every preceding step that deletes it, there exists another step that precedes the conditions and follows the deleter adds it back.

Causal proof is

- “local” (checks correctness one condition at a time)
- “state-less” (does not need to know the states preceding actions)
 - » Easy to extend to *durative* actions
- “incremental” with respect to action insertion
 - » Great for replanning

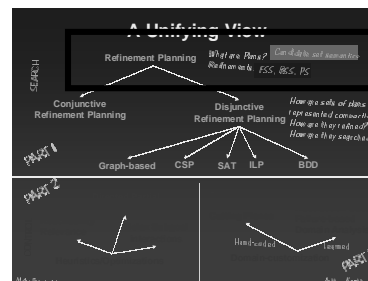


MA: Recent Advances in AI Planning - A Tutorial Volume

Subbarao Kambhampati

The Refinement Planning Framework:

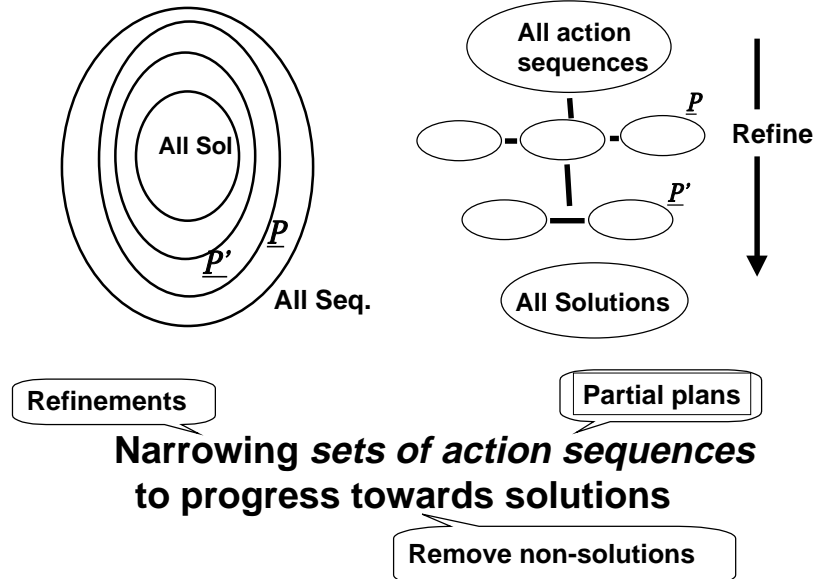
1. Syntax & Semantics of partial plans
2. Refinement strategies & their properties
3. The generic Refinement planning template



MA: Recent Advances in AI Planning - A Tutorial Volume

Subbarao Kambhampati

Refinement Planning: Overview

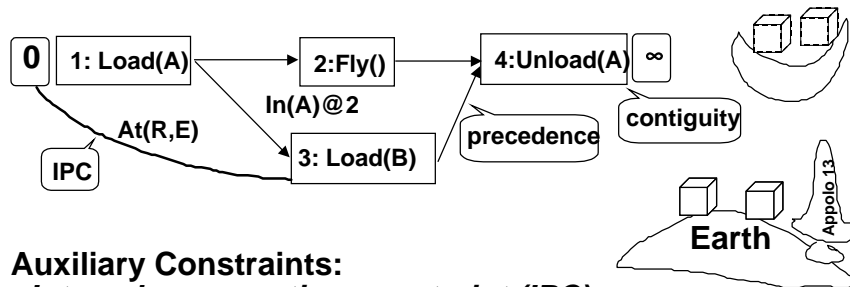


MA: Recent Advances in AI Planning - A Tutorial Volume

Subbarao Kamthampati

Partial Plans: Syntax

Partial plan = $\langle \text{Steps, Orderings, Aux. Constraints} \rangle$



Auxiliary Constraints:

Interval preservation constraint (IPC) $\langle s_1, p, s_2 \rangle$
 p must be preserved between s_1 and s_2

Point truth Constraint (PTC) $p@s$
 p must hold in the state before s

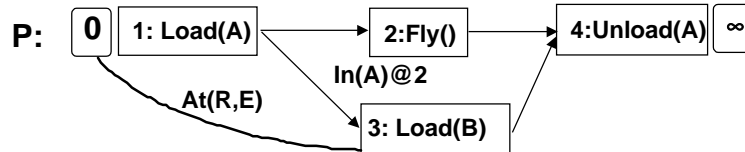
MA: Recent Advances in AI Planning - A Tutorial Volume

Subbarao Kamthampati

Partial Plans: Semantics

Candidate is any action sequence that

- contains actions corresponding to all the steps,
- satisfies all the ordering and auxiliary constraints



Candidates (\in «P»)

Non-Candidates (\notin «P»)

[Load(A), Load(B), Fly(), Unload(A)]

[Load(A), Fly(), Load(B), Unload(B)]

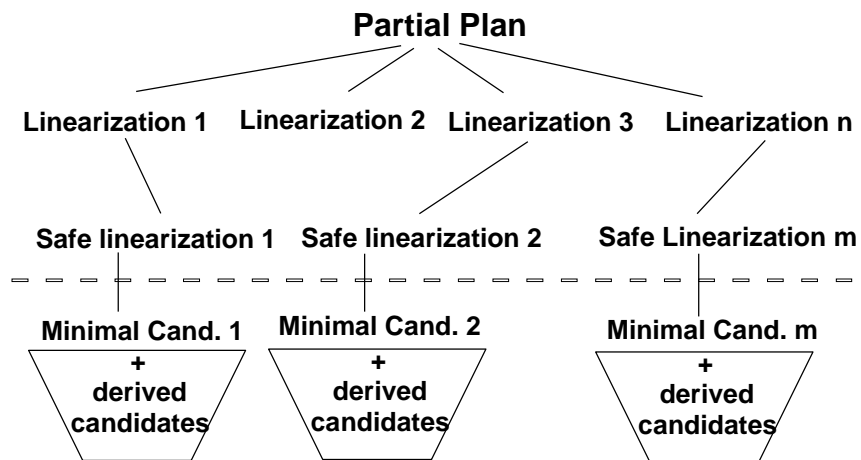
Minimal candidate. Corresponds to safe linearization [01324 ∞]

Corresponds to unsafe linearization [01234 ∞]

[Load(A), Load(B), Fly(), Unload(B), Unload(A)]

[Load(A), Fly(), Load(B), Fly(), Unload(A)]

Linking Syntax and Semantics



Refinements → Reduce candidate set size
 → Increase length of minimal candidates

Refinement (pruning) Strategies

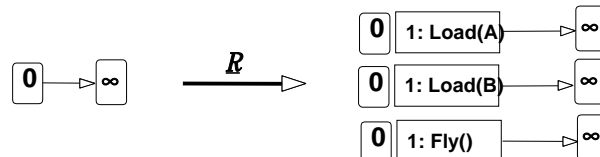
“Canned” inference procedures

actions & their inter-relations

- Prune by propagating the consequences of domain theory and meta-theory of planning onto the partial plan

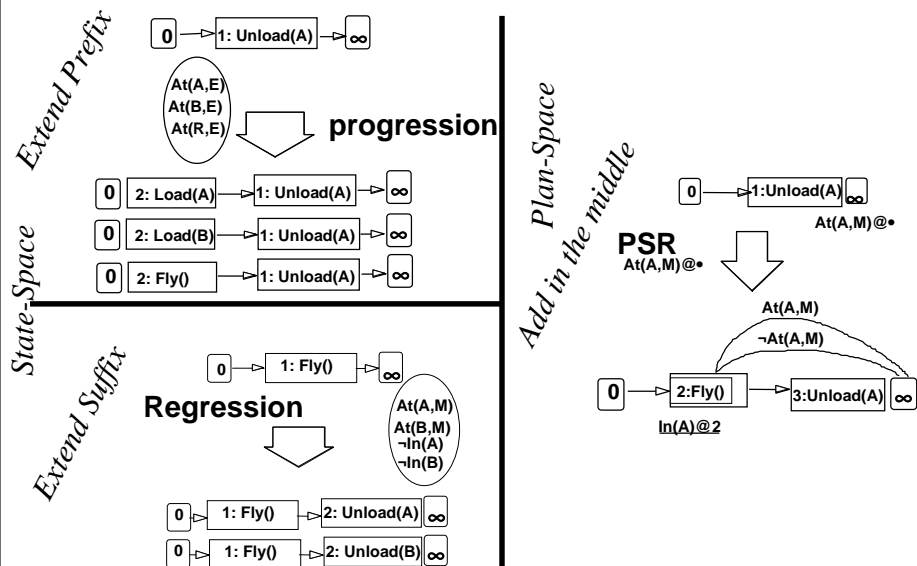
❖ A refinement strategy $\underline{R}: \underline{P} \rightsquigarrow \underline{P}'$ (« \underline{P}' » a subset of « \underline{P} »)

- \underline{R} is complete if « \underline{P}' » contains all the solutions of « \underline{P} »
- \underline{R} is monotonic if « \underline{P}' » has longer minimal candidates than « \underline{P} »
- \underline{R} is progressive if « \underline{P}' » is a proper subset of « \underline{P} »
- \underline{R} is systematic if components of \underline{P}' don't share candidates

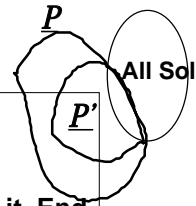


§ A plan set \underline{P} is a set of partial plans $\{P_1, P_2 \dots P_m\}$

Existing Refinement Strategies



The Refinement Planning Template



Refineplan(\underline{P} : Plan set)

Use proofs of correctness

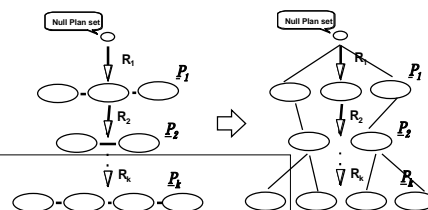
- 0*. If « \underline{P} » is empty, Fail.
1. If a minimal candidate of \underline{P} is a solution, return it. End
2. Select a refinement strategy \underline{R}
Apply \underline{R} to \underline{P} to get a new plan set \underline{P}'
3. Call Refine(\underline{P}')

- Termination ensured if \underline{R} is complete and monotonic
- Solution check done using one of the proofs of correctness

Issues:

1. Representation of plan sets (Conjunctive vs. Disjunctive)
2. Search vs. solution extraction
3. Affinity between refinement and proof used for solution check

A flexible Split&Prune search for refinement planning



Refineplan(\underline{P} : Plan)

- 0*. If « \underline{P} » is empty, Fail.
1. If a minimal candidate of \underline{P} is a solution, terminate.
2. Select a refinement strategy \underline{R} .
Apply \underline{R} to \underline{P} to get a new plan set \underline{P}'
3. Split \underline{P}' into k plansets
4. Non-deterministically select one of the plansets \underline{P}'_i
Call Refine(\underline{P}'_i)

Two classes of refinement planners

Conjunctive planners

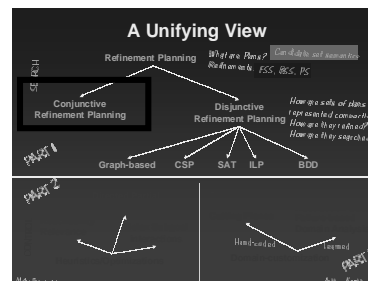
- ◇ Search in the space of conjunctive partial plans
 - Disjunction split into the search space
 - » search guidance is nontrivial
 - Solution extraction is trivial
- ◇ Examples:
 - STRIPS & Prodigy
 - SNLP & UCPOP
 - NONLIN & SIPE
 - UNPOP & HSP

Disjunctive planners

- ◇ Search in the space of disjunctive partial plans
 - Disjunction handled explicitly
 - Solution extraction is non-trivial
 - » CSP/SAT/ILP/BDD methods
- ◇ Examples:
 - Graphplan, IPP, STAN
 - SATPLAN
 - GP-CSP
 - BDDPlan, PropPlan

CONJUNCTIVE REFINEMENT PLANNING

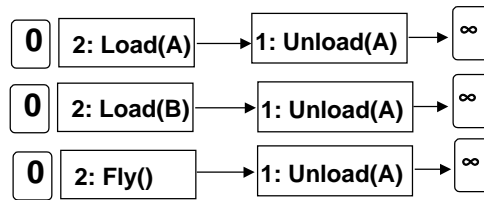
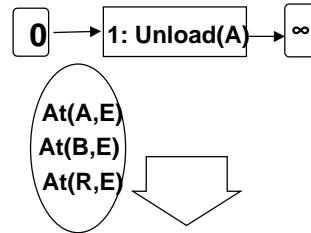
Part II



Forward State-space Refinement

- ◇ Grow plan prefix by adding applicable actions

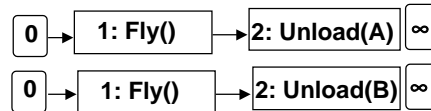
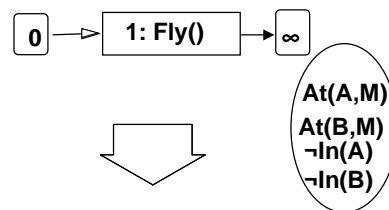
- Complete, Monotonic
 - » *consideration of all executable prefixes*
- Progressive
 - » *elimination of unexecutable prefixes*
- Systematic
 - » *each component has a different prefix*
- + Completely specified state
 - » Easier to control?
- Higher branching factor



Backward State-space Refinement

- ◇ Grow plan suffix by adding relevant actions

- Complete, Monotonic
 - » *consideration of all relevant suffixes*
- Progressive
 - » *elimination of irrelevant suffixes*
- Systematic
 - » *each component has a different suffix*
- + Goal directed
 - » Lower branching
- Partial state
 - » Harder to detect inconsistency

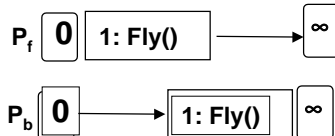


Tradeoffs among Refinements

*The Tasks great/
Less Filling holy wars*

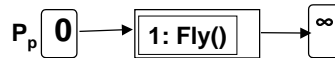
FSR and BSR must commit to both position and relevance of actions

- + Give state information (Easier plan validation)
- Leads to premature commitment
- Too many states when actions have durations



Plan-space refinement (PSR) avoids constraining position

- + Reduces commitment (large candidate set /branch)
- Increases plan-validation costs
- + Easily extendible to actions with duration



Generating Conjunctive Refinement Planners

Refineplan (\underline{P} : Plan)

- 0*. If « \underline{P} » is empty, Fail.
1. If a minimal candidate of \underline{P} is a solution, terminate.
2. Select a refinement strategy \underline{R} .
Apply \underline{R} to \underline{P} to get a set of new plans $P_1...P_j$
Add all plans to the search queue.
4. Non-deterministically select one of the plans P_i
Call Refine(P_i)

*Keep all linearizations safe
(Tractability refinements)
Select the proof strategy so it becomes
incremental with respect to the refinement
Stick to a single refinement strategy*

Issues in instantiating Refineplan

- ✧ Although a planner can use multiple different refinements, most implementations stick to a single refinement
- ✧ Although refinement can be used along with any type of correctness check, there is affinity between specific refinements and proof techniques (support finite differencing)
 - FSR and Progression based proof
 - BSR and Regression based proof
 - PSR and Causal proof
- ✧ Although it is enough to check if *any one* of the safe linearizations are solutions, most planners refine a partial plan until all its linearizations are safe
 - Tractability refinements (pre-order, pre-satisfy)

With these changes, an instantiated and simplified planner may “look” considerably different from the general template

Case Study: UCPOP

Refineplan (\underline{P} : Plan)

- 0*. If \underline{P} is order inconsistent, FAIL.
1. If no open conditions and no unsafe IPCs, SUCCESS.
 - 2. Generate new plans using either 2' or 2''
Add the plans to the search queue
- 2'. Remove an open condition $c@s$ in \underline{P} .
 - 2.1. For each step s' in \underline{P} that gives c , make a new plan
 $\underline{P}' = \underline{P} + (s' < s) + \text{IPC } s'-c-s$
 - 2.2. For each action A in the domain that gives c , make a new plan $\underline{P}' = \underline{P} + sn:A + (sn < s) + \text{IPC } sn-c-s$.
 - 2.2.1. For each precondition c' of A , add $c'@sn$ to the list of open conditions of \underline{P}' .
 - 2.2.2. For each IPC $s'-p-s''$, if sn deletes p , add $[s'-p-s''; sn]$ to the list of unsafe IPCs of \underline{P}' .
- 2''. Remove an unsafe IPC $[s'-p-s''; s''']$ from \underline{P} .
 Make two plans: $\underline{P}' = \underline{P} + s''' < s'$ $\underline{P}'' = \underline{P} + s'' < s'''$
3. Non-deterministically select one of the plans \underline{P}_i from the search queue and Call Refine(\underline{P}_i)

Incremental
Soln. check

Finite-differencing
structures

Tractability
refinement

Many variations on the theme..

Planner	Termination	Goal Sel.	Bookkeeping	Tractability refinements
TWEAK	MTC	MTC	-none-	-none-
UA	MTC	MTC	-none-	pre-order
SNLP / UCPOP	Causal proof	arbitrary	Contrib. Prot	pre-satisfaction
TOCL	Causal proof	arbitrary	Contrib. Prot.	pre-order
McNonlin/ Pedestal	Causal proof	arbitrary	Interval Prot.	pre-satisfaction
SNLP-UA	MTC	MTC	Contrib. Prot.	unambig. ord.

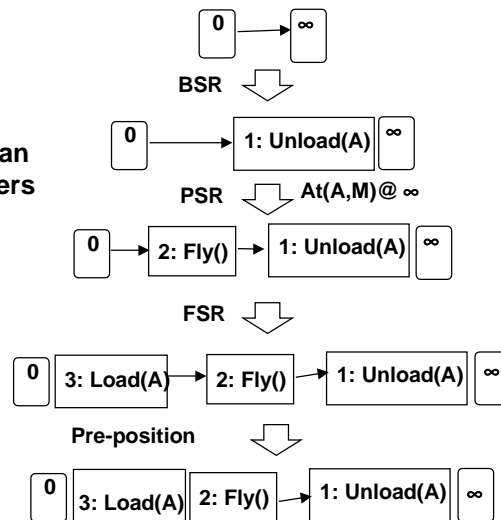
(Kamphampati, Knoblock & Yang, 1995)

MAA: Recent Advances in AI Planning - A Half-Century

Subbarao Kamphampati

Interleaving Refinements

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



(Kamphampati & Srivastava, 1995)

MAA: Recent Advances in AI Planning - A Half-Century

Subbarao Kamphampati

Some implemented conjunctive planners

	Refinement	Heuristics
UCPOP, SNLP [Weld et. al.]	Plan-space	Fail-first Flaw selection
UNPOP [McDermott] HSP [Geffner & Bonet] FF [Hoffman]	Forward state space	Distance heuristics
IxTET [Ghallab et al] RAX [Muscettola et al]	Plan-space	Hand-coded search control
Prodigy [Carbonell et. al.]	Forward state space	Means-ends analysis

MA2: Recent Advances in AI Planning - A Tutorial Version

Subbarao Kambhampati

Conjunctive planners: The State of the Art

- ◇ **Vanilla state-space (FSR/BSR) planners** were known to be less efficient than vanilla plan-space planners
 - Several research efforts concentrated on extending plan-space approaches to non-classical scenarios
- ◇ However, **State-space planners** seem to have better heuristic support than plan-space planners at this time
 - Distance-based heuristics seem particularly useful for state-space planners
- ◇ But, **plan-space planners** are alleged to provide better support for incremental planning, durative actions etc.
 - replanning, reuse and plan modification, temporal reasoning
 - » IXTET, HSTS, RAX....

Charge: *Either develop effective approaches for replanning etc. in state-space planners or develop good heuristics for plan-space planners*

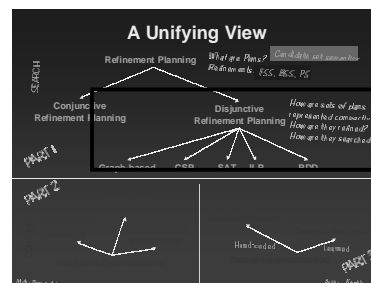
MA2: Recent Advances in AI Planning - A Tutorial Version

Subbarao Kambhampati

Part 1.2

DISJUNCTIVE REFINEMENT PLANNING

1. Refining disjunctive plans
 2. Solution Extraction
 - Direct
 - Compilation
- CSP/SAT/ILP/BDD



MA: Recent Advances in AI Planning - A Unifying View

Subbarao Kambhampati

Disjunctive Planning

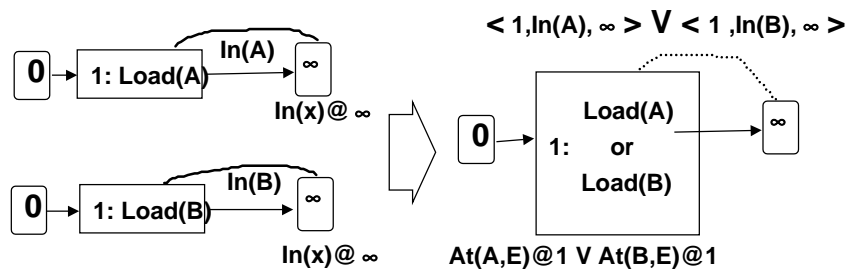
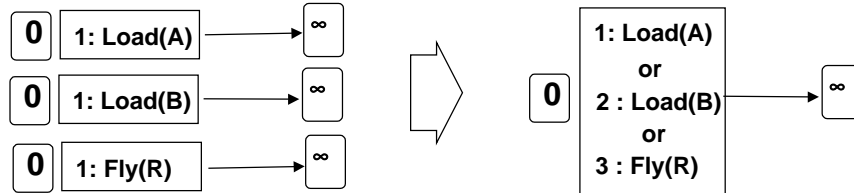
- ❖ **Idea:** Consider Partial plans with disjunctive step, ordering, and auxiliary constraints
- ❖ **Motivation:** Provides a lifted search space, avoids re-generating the same failures multiple times (also, rich connections to combinatorial problems)
- ❖ **Issues:**
 - Refining disjunctive plans
 - » Graphplan (Blum & Furst, 95)
 - Solution extraction in disjunctive plans
 - » Direct combinatorial search
 - » Compilation to CSP/SAT/ILP

MA: Recent Advances in AI Planning - A Unifying View

Subbarao Kambhampati

Disjunctive Representations

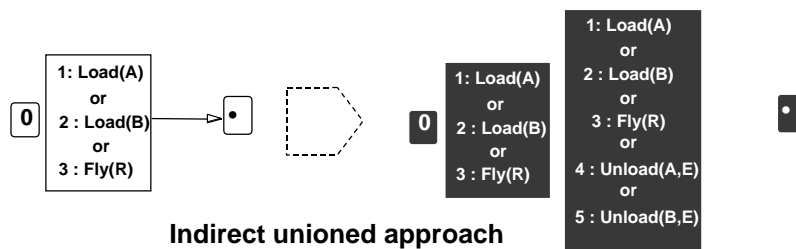
--Allow disjunctive step, ordering and auxiliary constraints in partial plans



MAA: Recent Advances in AI Planning - A Unit-Based View

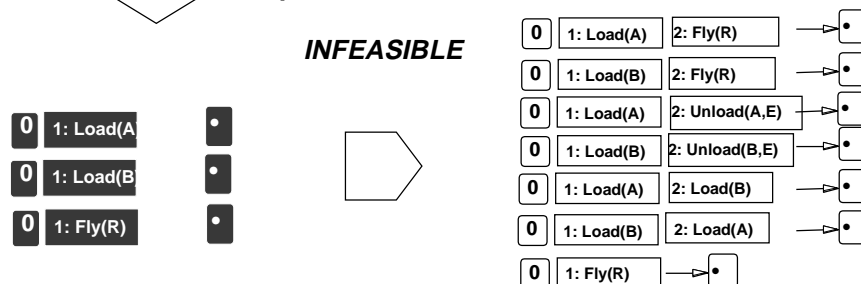
Subbarao Kambhampati

Refining Disjunctive Plans (1)



Indirect unioned approach

+ Maximum pruning power
- Exponential refinement cost



MAA: Recent Advances in AI Planning - A Unit-Based View

Subbarao Kambhampati

Refining Disjunctive plans (2)

Direct naive approach

Put all actions at all levels

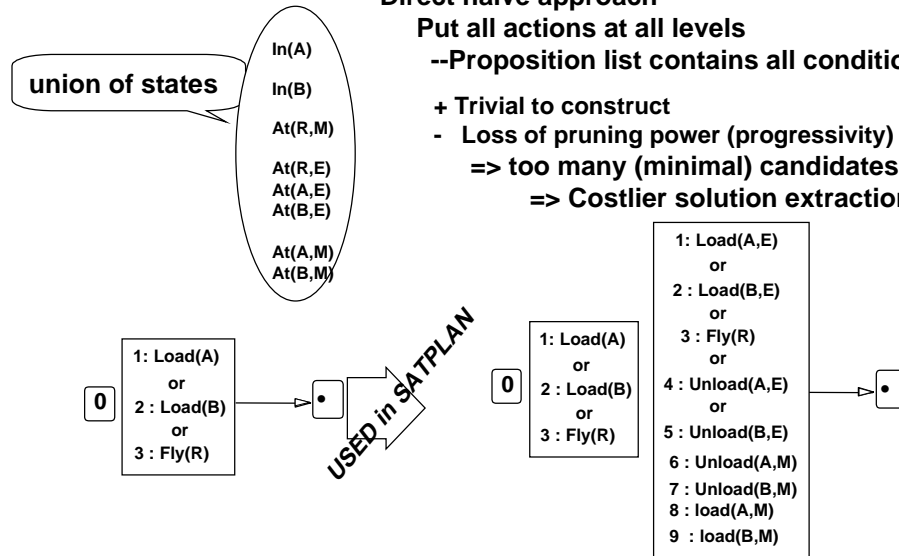
--Proposition list contains all conditions

+ Trivial to construct

- Loss of pruning power (progressivity)

=> too many (minimal) candidates

=> Costlier solution extraction



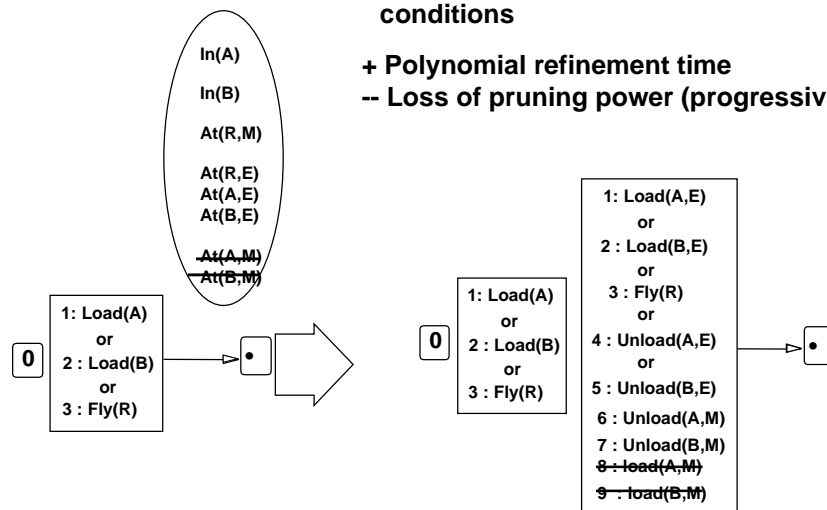
Refining Disjunctive plans (3)

Enforce partial 1-consistency

Proposition list avoids unsupported conditions

+ Polynomial refinement time

-- Loss of pruning power (progressivity)



Refining Disjunctive plans (4)

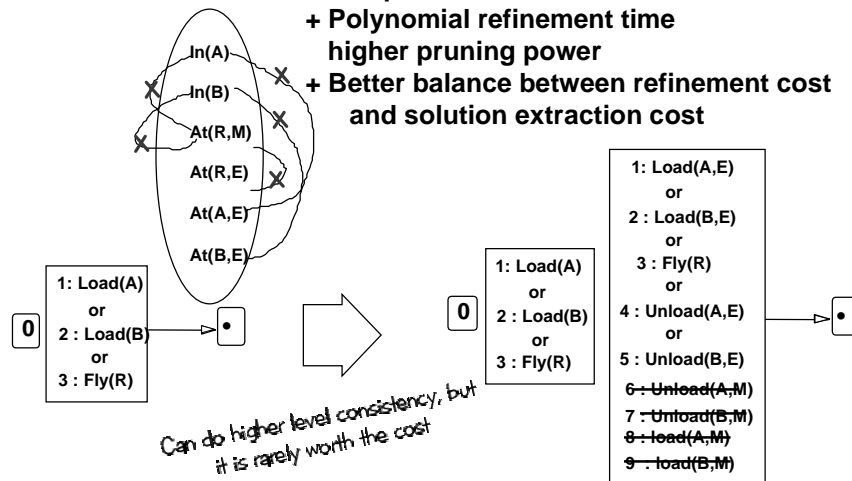
Enforce (partial) 2-consistency

Proposition list maintains interactions
between pairs of conditions

+ Polynomial refinement time

higher pruning power

+ Better balance between refinement cost
and solution extraction cost



Open issues in disjunctive refinement

- ◇ **Directed partial consistency**
 - Mutex propagation is a form of reachability analysis
 - » Relevance analysis?
 - Higher levels of directional consistency?
 - » Typically not cost effective
- ◇ **Supporting refinements other than FSR**
 - Direct naïve refinements are easy to support; enforcing an appropriate level of consistency is harder
 - Some “relevance” based approaches exist for BSR
 - » Inseparability, backward mutex [
 - Kambhampati et. al. ECP, 1997
 - » Can be used in conjunction with reachability analysis
 - Enforcing effective consistency for PSR is still virgin territory...

Solution Extraction in Disjunctive Plans

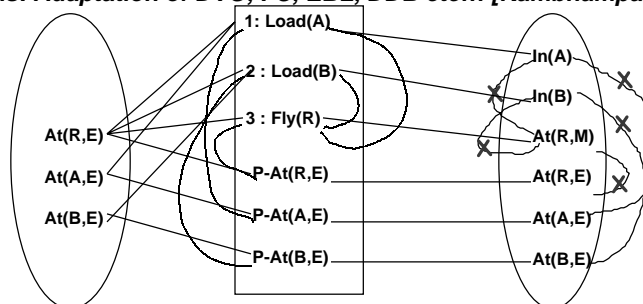
- ◇ **Verify if some minimal candidate of the disjunctive plan is a solution (“Model Finding”)**
 - » Involves (implicit or explicit) use of proof strategies
- ◇ **Direct methods: Can be solved using special techniques customized to the disjunctive structure**
 - » Graphplan backward search; Graphplan local search
- ◇ **Compilation methods: Generate a set of constraints, whose satisfaction ensures that a substructure of the disjunctive structure is a solution. Find a model for the constraints.**
 - » Constraints will correspond to lines of proof
 - Progression proof, regression proof, causal proof
 - » Constraints can be in any standard form
 - e.g. CSP, SAT, ILP, BDD

Graphplan Backward Search (Direct Search I)

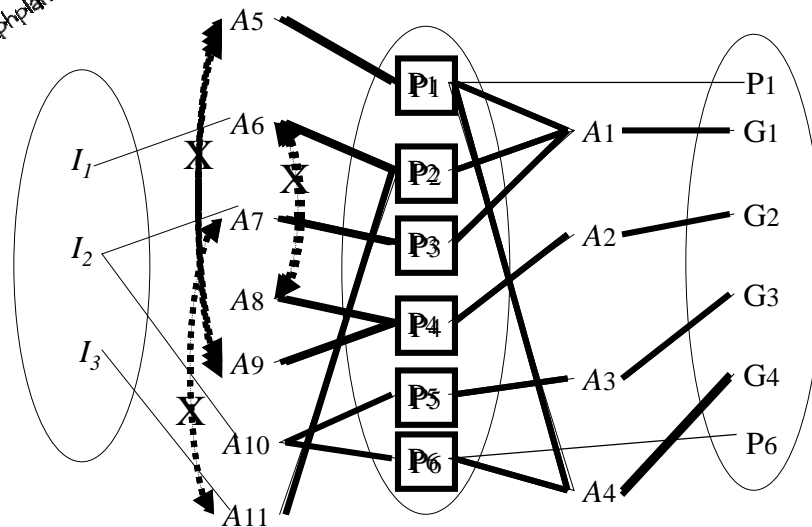
Objective: Find a sub-graph of the plangraph that corresponds to a valid plan.

Method: Start from the goal propositions at the last level
 Select actions to support the goals so that no two are *mutex* (*choice*)
 Recurse on the preconditions of the selected actions
 (recursion ends at the initial state)
(When backtracking over the goals at a level, memoize them)

Optimizations: Adaptation of DVO, FC, EBL, DDB etc... [Kambhampati, JAIR 2000]



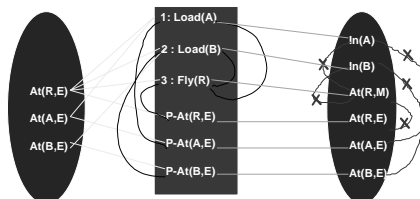
Graphplan Review



MA2: Recent Advances in AI Planning: A Unified View....

Subbarao Kambhampati

- ❖ **Motivation: No compelling reason for making the search for a valid subgraph backward, or systematic...**
- ❖ **Alternatives:**
 - **Forward Search (dynamic programming) [Parker & Kambhampati; 98,99; Blum & Langford 98]**
 - **Systematic Undirectional search [Rintanen, 98]**
 - » *Select an action anywhere in the plan-graph for inclusion in the solution; Propagate consequences (adapts normal CSP Search to plan-graph)*
 - **Local Search [Gerevini et. al., 99]**



MA2: Recent Advances in AI Planning: A Tutorial. J. Edelkamp.

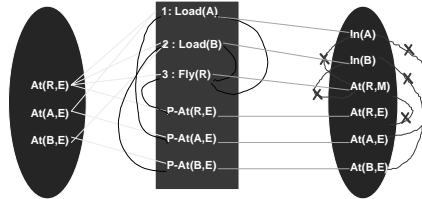
Subbarao Kambhampati

Do & Kamshampati, 2000

Compilation to CSP

Goals: $\text{In}(A), \text{In}(B)$

CSP: Given a set of discrete variables, the domains of the variables, and constraints on the specific values a set of variables can take in combination, FIND an assignment of values to all the variables which respects all constraints



Variables: Propositions ($\text{In-A-1}, \text{In-B-1}, \dots, \text{At-R-E-0} \dots$)

Domains: Actions supporting that proposition in the plan

$\text{In-A-1} : \{ \text{Load-A-1}, \# \}$ $\text{At-R-E-1} : \{ \text{P-At-R-E-1}, \# \}$

Constraints: Mutual exclusion

$\sim [(\text{In-A-1} = \text{Load-A-1}) \ \& \ (\text{At-R-M-1} = \text{Fly-R-1})] ; \text{etc..}$

Activation

$\text{In-A-1} \neq \# \ \& \ \text{In-B-1} \neq \#$ (Goals must have action assignments)

$\text{In-A-1} = \text{Load-A-1} \Rightarrow \text{At-R-E-0} \neq \#, \text{At-A-E-0} \neq \#$

(subgoal activation constraints)

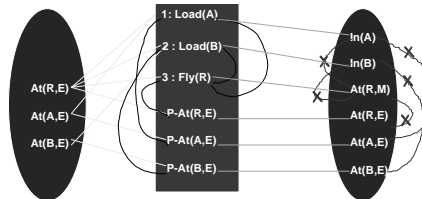
(Corresponds to a regression-based proof)

Katz & Selman

Compilation to SAT

Goals: $\text{In}(A), \text{In}(B)$

SAT is CSP with Boolean Variables



Init: $\text{At-R-E-0} \ \& \ \text{At-A-E-0} \ \& \ \text{At-B-E-0}$

Goal: $\text{In-A-1} \ \& \ \text{In-B-1}$

Graph: “cond at $k \Rightarrow$ one of the supporting actions at $k-1$ ”

$\text{In-A-1} \Rightarrow \text{Load-A-1}$ $\text{In-B-1} \Rightarrow \text{Load-B-1}$

$\text{At-R-M-1} \Rightarrow \text{Fly-R-1}$ $\text{At-R-E-1} \Rightarrow \text{P-At-R-E-1}$

$\text{Load-A-1} \Rightarrow \text{At-R-E-0} \ \& \ \text{At-A-E-0}$

“Actions \Rightarrow preconds”

$\text{Load-B-1} \Rightarrow \text{At-R-E-0} \ \& \ \text{At-B-E-0}$

$\text{P-At-R-E-1} \Rightarrow \text{At-R-E-0h}$

$\sim \text{In-A-1} \vee \sim \text{At-R-M-1} \ \sim \text{In-B-1} \vee \sim \text{At-R-M-1}$ “Mutexes”

Compilation to Integer Linear Programming

ILP: Given a set of real valued variables, a linear objective function on the variables, a set of linear inequalities on the variables, *and a set of integrality restrictions on the variables*, Find the values of the feasible variables for which the objective function attains the maximum value

-- o/i integer programming corresponds closely to SAT problem

✧ Motivations

- Ability to handle numeric quantities, and do optimization
- Heuristic value of the LP relaxation of ILP problems

◆ Conversion

- Convert a SAT/CSP encoding to ILP inequalities
 - » E.g. $X \vee \neg Y \vee Z \Rightarrow x + (1 - y) + z \geq 1$
- Explicitly set up tighter ILP inequalities (Cutting constraints)
 - » If X,Y,Z are pairwise mutex, we can write $x+y+z \leq 1$
(instead of $x+y \leq 1$; $y+z \leq 1$; $z+x \leq 1$)

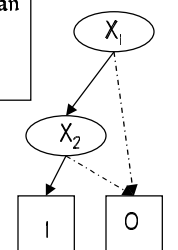
Wasser & Kautz
Vossen et. al
Bockmayr & Dimopoulos

Compilation to Binary Decision Diagrams (BDDs)

BDDs support compact representation *and* direct manipulation of boolean formulae on a finite set of propositions. (Popular in CAD community)
Standard algorithms for converting a boolean formulae into BDDs and for supporting standard boolean operations on them [Bryant et. al.]

- ✧ **Idea: Represent disjunctive plans as BDDs and plan extension as BDD operations**

- **Proposition list at level k is an approximation to the set of states reachable in k steps.**
 - » The set can be represented compactly as BDDs
 - » Plan growth can be modeled as direct manipulations on BDD
 - Operations such as “action projection” need to be modeled as BDD modifications



BDD for X_1 & X_2

Relative Tradeoffs Offered by the various compilation substrates

- ◇ **CSP** encodings support implicit representations
 - More compact encodings [Do & Kambhampati, 2000]
 - Easier integration with Scheduling techniques
- ◇ **ILP** encodings support numeric quantities
 - Seamless integration of numeric resource constraints [Walser & Kautz, 1999]
 - Not competitive with CSP/SAT for problems without numeric constraints
- ◇ **SAT** encodings support axioms in propositional logic form
 - May be more natural to add (for whom ;-)
- ◇ **BDDs** are very popular in CAD community
 - Commercial interest may spur faster algorithms (which we can use)

Direct vs. compiled solution extraction

DIRECT	Compiled
<ul style="list-style-type: none"> ✗ Need to adapt CSP/SAT techniques ✓ Can exploit approaches for compacting the plan ✓ Can make the search incremental across iterations 	<ul style="list-style-type: none"> ✓ Can exploit the latest advances in SAT/CSP solvers ✗ Compilation stage can be time consuming, leads to memory blow-up ✗ Makes it harder to exploit search from previous iterations ✓ Makes it easier to add declarative control knowledge

Disjunctive planners based on BSS and PS refinements?

Graphplan can be seen as a disjunctive planner based on state-space refinement.

- How about planners based on other refinements?
- ◇ Chief difficulty lies in generalizing the refinements to disjunctive plans, while retaining their progressivity
 - The mutex propagation step of Graphplan is what makes it progressive
- ◇ Generalization is however quite easy *If* we sacrifice “progressiveness” of refinements

Basic idea:

- Set up bounded length disjunctive structure
- Search for a substructure that satisfies solution properties
 - » Use lines of proofs

[Mali & Kambhampati, 1999]

Lines of Proof as basis for (*naïve*) encodings

- ◇ Loop from $k=1$, until a solution is found
 - Set up k -length sequence of disjunctive actions ($a_1 \vee a_2 \vee \dots \vee a_n$)
 - » In effect, a direct naïve refinement is used (monotonic, complete, but not progressive)
 - Impose constraints, satisfaction of which ensures that some subsequence of the disjunctive sequence is a solution
 - » The constraints set up lines of proof
 - State-space proofs
 - Graphplan’s backward search can be thought of as setting up backward proof
 - Causal proofs
 - Convert the constraints into your favorite combinatorial substrate and solve

a_1	a_1	a_1	a_1
a_2	a_2	a_2	a_2
a_3	a_3	a_3	a_3
....
a_n	a_n	a_n	a_n
1	2		k

Encodings based on Causal proofs

S0	S1	S2	S3	Sk	S∞
Needs(S0, I1)	a1	a1	a1	a1	Needs(S∞, G1) Needs(S∞, G2)
Needs(S0, I2)	a2	a2	a2	a2	
□	a3	a3	a3	a3	
Needs(S0, In)	
	an	an	an	an	

Each step is mapped to exactly one action
 $Si = A1 \vee Si = A2 \dots ; \sim(Si = Ai \ \& \ Si = Aj)$
 A step inherits the needs, adds and deletes of the action it is mapped to
 $Si = Aj \Rightarrow Adds(Si, Pa) \ \& \ Needs(Si, Pp) \ \& \ Deletes(Si, Pd)$
 A Step get needs, adds and deletes only through mapped actions
 $Adds(Si, Pa) \Rightarrow Si = Aj \vee Si = Ak \dots$
 $(Ai, Ak \text{ add } Pa)$
 Every need is established by some step
 $Needs(Si, Pj) \Rightarrow Estab(S1, Pj, Si) \vee$
 $Estab(S2, Pj, Si) \dots \vee Estab(Sk, Pj, Si)$

Establishment with causal links
Estab(S1,Pj,Si) => Link(S1,Pj,Si)
Link implies addition & precedence
Link(Si,Pj,Sk) => Adds(Si,Pj) &
Precedes(Si,Pj)
Link implies preservation by
intervening steps
Link(Si,Pj,Sk) & Deletes(Sm,Pj)
=> Precedes(Sm, Si) V
Precedes(Sk, Si)

**Precedence is irreflexive,
asymmetric and transitive...**

Tradeoffs between encodings based on different proof strategies

- ◇ **Causal encodings in general have more clauses than state-space encodings**
 - $O(\#actions \times \#actions \times \#fluents)$ for causal link variables
 - » Could be reduced by using white-knight based proofs
 - $O(\#actions \times \#actions \times \#actions)$ clauses for partial ordering
 - » Could be reduced by using contiguous ordering
 - However, the best causal encodings will still be dominated by the backward state-space encodings [Mali & Kambhampati, 99]
- ◇ **Paradoxical given the success of partial order planners in conjunctive planning?**
 - Not really! We are using causal proof which is typically longer than state-based proofs, and are not using the flexibility of step insertion.
 - » Can be helpful in incremental planning & Plan reuse
 - » Are helpful in using causal domain specific knowledge (e.g. HTN schemas)

Some implemented disjunctive planners

	Refinement	Solution Extraction
Graphplan (Blum/Furst) IPP (Koehler) STAN (Fox/Long) GP-EBL (Kambhampati)	<i>Direct Partially 2-consistent refinement with FSR</i>	Direct search on the disjunctive plan
SATPLAN (Kautz/Selman)	<i>Naïve direct refinement with FSR, BSR, PSR</i>	Compilation to SAT
Blackbox (Kautz/Selman)	<i>Same as Graphplan</i>	
GP-CSP (Do/Kambhampati)	<i>Same as Graphplan</i>	Compilation to CSP
IPP Plan (Walser/Kautz)	<i>Naïve direct refinement with FSR</i>	Compilation to ILP
BDDPlan (Hans-Peter) PROPlan (Stoerr)	<i>Naïve direct refinement with FSR</i>	Compilation to BDDs

MA2: Recent Advances in AI Planning - A Tutorial Version

Subbarao Kambhampati

Conjunctive vs. Disjunctive planners

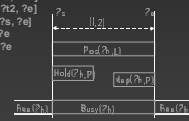
- | | |
|---|---|
| <ul style="list-style-type: none"> ◇ Progress depends on the effectiveness of the heuristics for judging the goodness of a partial plan ◇ Space consumption can be regulated ◇ Better fit with mixed-initiative, incremental planning scenarios(?) | <ul style="list-style-type: none"> ◇ Space consumption is a big issue <ul style="list-style-type: none"> – Creation and storage of disjunctive structures <ul style="list-style-type: none"> » CSPs do better by supporting implicit representations ◇ Connection to combinatorial substrate ◇ Better integration with non-propositional reasoners (?) |
|---|---|

Hybrid planners--Controlled splitting &
Search in the space of disjunctive plans

MA2: Recent Advances in AI Planning - A Tutorial Version

Subbarao Kambhampati

Load(P:package, R:rocket, L:location)



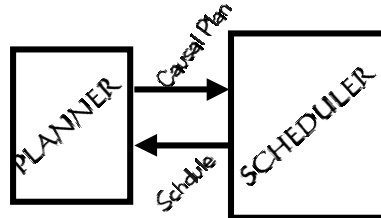
MA2: Recent Advances in AI Planning: A Unified View....

Subbarao Kambhampati

[illegible]

- Customized consistency enforcement techniques
 - » ARC-B consistency
 - » Edge-finding
- Customized variable/value ordering heuristics
 - » Contention-based
 - » Slack-based
- MaxCSP; B&B searches

Adapting to Metric/Temporal Planning



Schedulers already routinely handle resources and metric/temporal constraints.

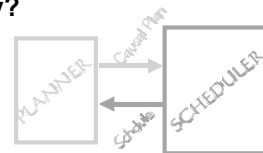
- Let the “planner” concentrate on causal reasoning
- Let the “scheduler” concentrate on resource allocation, sequencing and numeric constraints for the generated causal plan

Need better coupling to avoid inter-module thrashing....

Issues in handling time and resources

- Answer!
- ◇ Monolithic or loosely-coupled?
 - ◇ How can the Planner keep track of consistency?
 - Low level constraint propagation
 - » Loose path consistency on TCSPs
 - » Bounds on resource consumption,
 - » LP relaxations of metric constraints
 - Pre-emptive conflict resolution

The more aggressive you do this, the less need for a scheduler..
 - ◇ How do the modules interact?
 - Failure explanations; Partial results
 - ◇ Which Planners are best suited for time and resources?*



What planners are good for handling resources and time?

- ❖ State-space approaches have an edge in terms of ease of monitoring resource usage
 - *Time-point based representations are known to be better for multi-capacity resource constraints in scheduling*
- ❖ Plan-space approaches have an edge in terms of durative actions and continuous change
 - Notion of state not well defined in such cases (Too many states)
 - *PCP representations are known to be better for scheduling with single-capacity resources*
- ❖ Disjunctive compilation approaches may provide better basis for interacting with schedulers and other external constraint reasoners
 - Conjunctive approaches better for *monolithic* architectures (disjunctive ones may lead to very large encodings)

[Adapted from David Smith's Invited Talk at AIPS-2000]

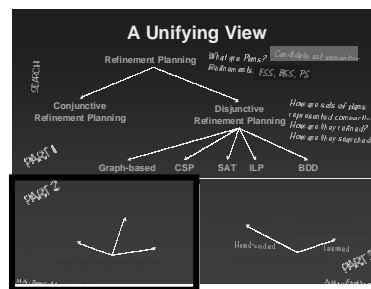
Tradeoffs in the current implementations

	Multi-capacity Resources	Metric Quantities	Optimization	Continuous Time	Speed
Graphplan	hard?	hard IPP	hard?	hard TGP	good
SAT Compilation	moderate?	moderate LPSAT	moderate?	very hard?	good
ILP Compilation	easy?	easy ILP-plan	easy ILP-plan	very hard?	fair?
Conjunctive PS	moderate IxTeT	moderate Zeno	moderate?	easy HSTS	fair?

Work on loosely-coupled architectures is just starting... [Srivastava, 2000]

PART 2

HEURISTICS & OPTIMIZATIONS



MA: Recent Advances in AI Planning - A Unifying View

Subbarao Kamthampati

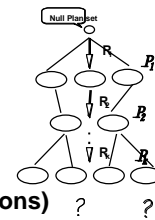
Distance Heuristics from Relaxed Problems

Problem: Estimating the distance of a (partial) state from the initial (or goal) state

Solution: Relax the problem by assuming that all subgoals are independent (ignore +ve / -ve interactions between actions)
Solve the relaxed problem and use the length of the solution as part of the heuristic

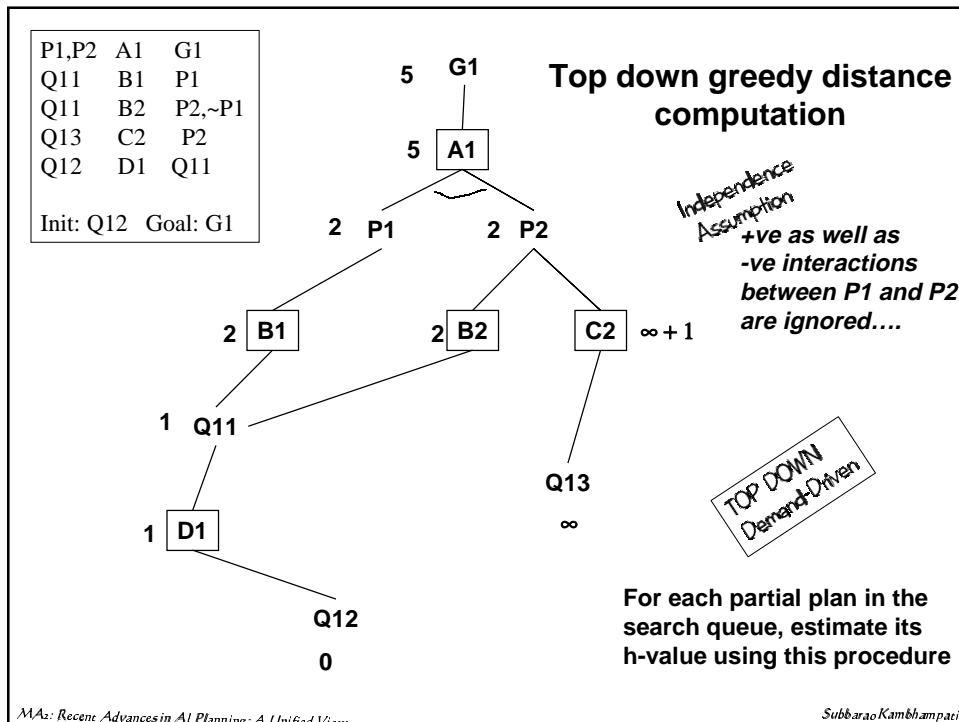
Properties: The heuristic is neither a lower bound (-ve interactions) nor an upper-bound (+ve interactions).
--leads to inoptimal solutions (in terms of plan length)
>>Possible to reduce inoptimality by considering interactions

History: First used in IxTET [Ghallab et. al. 1995]
hinted at by Smith's Operator Graphs [Smith et. al., 1995]
independently re-discovered in UNPOP [McDermott, 1996]
independently re-re-discovered in HSP [Bonet & Geffner, 1997]



MA: Recent Advances in AI Planning - A Unifying View

Subbarao Kamthampati



Bottom-up Distance computation

- ◇ Each condition (state-variable & value combo) has a distance
 - Initialized to 0 for all conditions in the initial state
 - ∞ for everything else
- ◇ Repeat until reaching fix-point (until no distances change)
 - Select an action A to be applied. Let $P_1 \dots P_m$ be its preconditions and $E_1 \dots E_n$ be the conditions it adds.
 - Reset the distance values of E_i as follows

$$Dist(E_i) = \min \left(\underbrace{Dist(E_i)}_{\text{Old Dist}}, Dist(\{P_1 \dots P_i\}) + 1 \right)$$

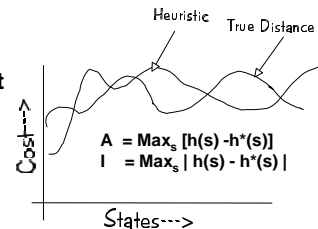
$$Dist(\{P_1 \dots P_i\}) = \sum_i Dist(P_i)$$

Independence assumption

Bottom-up

Improving the Heuristic

- ◇ Subgoal independence assumption leads to
 - Inadmissibility (when subgoals interact positively; A is +ve)
 - Un-informedness (when subgoals interact negatively; I is large)
- ◇ Informedness improved by accounting for -ve interactions
 - Use information about infeasible subgoal sets (“Mutexes”)
 - » No plan of length k can achieve p and q
 - » No plan can ever achieve p and q
- ◇ Admissibility can be improved by accounting for +ve interactions
 - Solve the planning problems ignoring -ve interactions between actions to get lower bound on plan length (distance)



MAA: Recent Advances in AI Planning - A Tutorial Volume

Subbarao Kamphampati

Long & Kamphampati,
AAAI 2000

Using the Planning Graph to account for +ve/-ve Interactions

- ◇ Mutex relations can help account for -ve interactions
 - Grow the PG until *level-off* (Polynomial operation)
 - Distance of a set $\{P_1, P_2, \dots, P_n\}$ is the *index* of the first level of the PG where P_1, \dots, P_n are present without any mutexes
 - » Easy to generalize to n -ary mutexes
 - » Informedness can be improved by partitioning P_1, \dots, P_n into some k partitions, S_1, \dots, S_k and *summing* the distances of S_i
- ◇ +ve interactions can be accounted for by extracting a plan from the PG *ignoring* mutex relations
 - Run Graphplan algorithm, but do not do any mutex propagation.

Insight: Ideas of disjunctive planning help
in generating heuristics for conjunctive planners!!

MAA: Recent Advances in AI Planning - A Tutorial Volume

Subbarao Kamphampati

Plan Graph produces a large spectrum of effective heuristics with differing tradeoffs

Problem	Graphplan	Sum-mutex	set-lev	partition-1	partition-2	adj-sum	combo	adj-sum2
bw-large-b	18/ 379.25	18/ 132.50	18/ 10735.48	-	18/ 79.18	22/ 65.62	22/ 63.57	18/ 87.11
bw-large-c	-	-	-	-	-	30/ 724.63	30/ 444.79	28/ 738.00
bw-large-d	-	-	-	-	-	-	-	36/ 2350.71
rocket-ext-a	-	36/ 40.08	-	32/ 4.04	32/ 10.24	40/ 6.10	34/ 4.72	40/ 43.63
rocket-ext-b	-	34/ 39.61	-	32/ 4.93	32/ 10.73	36/ 14.13	32/ 7.38	36/ 554.78
att-log-a	-	69/ 42.16	-	63/ 10.13	-	63/ 16.97	63/ 11.96	56/ 36.71
att-log-b	-	67/ 56.08	-	69/ 20.05	-	67/ 32.73	67/ 19.04	61/ 53.28
gripper-20	-	59/ 90.68	-	59/ 39.17	-	59/ 20.54	59/ 20.92	59/ 38.18
8-puzzle1	31/ 2444.22	33/ 196.73	31/ 4658.87	35/ 80.05	47/ 172.87	39/ 78.36	39/ 119.54	31/ 143.59
8-puzzle2	30/ 1545.66	42/ 224.15	30/ 2411.21	38/ 96.50	38/ 105.40	42/ 103.70	48/ 50.45	30/ 348.27
8-puzzle3	20/ 50.56	20/ 202.54	20/ 68.32	20/ 45.50	20/ 54.10	24/ 77.39	20/ 63.23	20/ 62.56
travel-1	9/ 0.32	9/ 5.24	9/ 0.48	9/ 0.53	9/ 0.62	9/ 0.42	9/ 0.44	9/ 0.53
grid3	16/ 3.74	-	16/ 14.09	16/ 55.40	16/ 46.79	18/ 21.45	19/ 18.82	16/ 15.12
grid4	18/ 21.30	-	18/ 32.26	18/ 86.17	18/ 126.94	18/ 37.01	18/ 37.12	18/ 30.47
airs-grid1	14/ 311.97	-	14/ 659.81	14/ 870.02	14/ 1010.80	14/ 679.36	14/ 640.47	14/ 739.43
mrtime-1	4/ 17.48	-	4/ 743.66	4/ 78.730	4/ 622.67	4/ 76.98	4/ 79.55	4/ 722.55

Table 2: Number of actions/ Total CPU Time in seconds. The dash (-) indicates that no solution was found in 3 hours or 250MB.

[Long & Kambhampati,
AAAI 2000]

Using Distance Heuristics

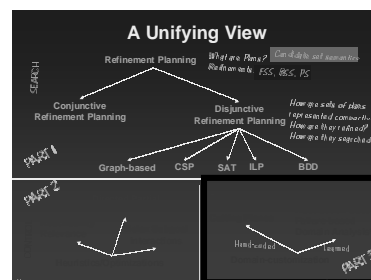
- ❖ **Conjunctive FSS planners need to recompute the heuristic as the init state changes**
 - UNPOP [McDermott], HSP [Bonet & Geffner], FF [Hoffman]
 - » GRT [Refanidis & Vlahavas] elaborates the goal state and computes heuristic in the reverse direction
- ❖ **Conjunctive BSS planners can get by with a single computation of the heuristic**
 - HSP-R [Bonet & Geffner], AltAlt [Long & Kambhampati &..]
- ❖ **Conjunctive PS planners can use distance heuristics and mutex information to rank partial plans and flaw resolution choices**
 - IxTET [Ghallab et. al.]; AltAlt-UC [Long & Kambhampati]
- ❖ **Disjunctive planners can use distance heuristics as the basis for variable and value ordering heuristics**
 - GP-HSP [Kambhampati & Nigenda], GP-CSP [Do & Kambhampati]

Other notable optimizations

- ◇ **Reduce the size of the disjunctive plan**
 - Bi-level Representations [Fox & Long, 97][Smith & Weld, 97]
 - Lifted representations [Parke & Ginsberg, 2000]
 - Relevance based pruning
 - » Backward plan growth [Kambhampati et. al., 97]
 - » Preprocessing to remove irrelevant actions and conditions [RIFO, Nebel et. al. 97]
- ◇ **Increase the consistency level of the disjunctive plan**
 - Learn (or input) higher-order mutexes (invariants) [Gerevini, 98] [Fox & Long, 99][Rintanen, 2000]
- ◇ **Improve the solution extraction process**
 - Exploit Symmetry [Fox et. al. 99, Srivastava et. al. 99]
 - + the usual retinue of search improvements from CSP, ILP etc. [Kambhampati, 2000]

PART 3 CUSTOMIZING PLANNERS WITH DOMAIN SPECIFIC KNOWLEDGE

1. User-assisted customization
2. Automated customization



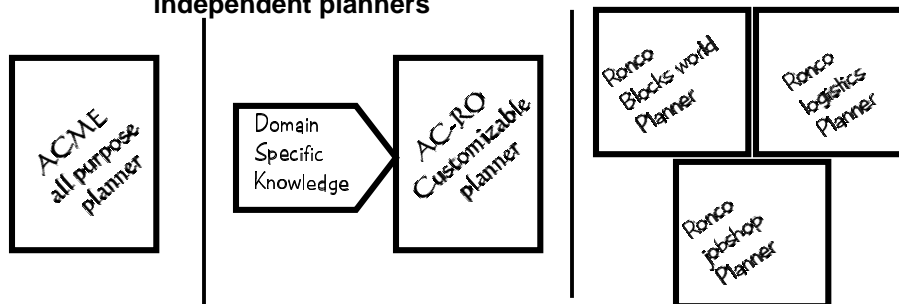
Improving Performance through Customization

- ◇ **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

User-Assisted Customization (using domain-specific Knowledge)

- ◇ Domain independent planners tend to miss the regularities in the domain
- ◇ Domain specific planners have to be built from scratch for every domain

An “Any-Expertise” Solution: Try adding domain specific control knowledge to the domain-independent planners



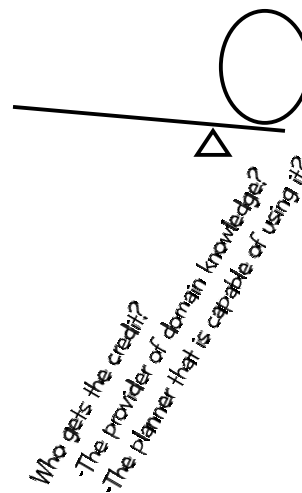
Many User-Customizable Planners

- ◇ Conjunctive planners
 - HTN planners
 - » SIPE [Wilkins, 85-]
 - » NONLIN/O-Plan [Tate et. al., 77-]
 - » NOAH [Sacerdoti, 75]
 - » Also SHOP (Nau et. al., IJCAI-99)
 - State-space planners
 - » TLPlan [Bacchus & Kabanza, 95; 99]
 - » TALPlan [Kvarnstrom & Doherty, 2000]
 - Customization frameworks
 - » CLAY [Srivastava & Kambhampati, 97]
- ◇ Disjunctive planners
 - HTN SAT [Mali & Kambhampati, 98]
 - SATPLAN+Dom [Kautz & Selman, 98]

How do they relate?
What are the tradeoffs?

With right domain knowledge any level of performance can be achieved...

- ◇ HTN-SAT, SATPLAN+DOM beat SATPLAN...
 - Expect reduction schemas, declarative knowledge about inoptimal plans
- ◇ TLPLAN beats SATPLAN, GRAPHPLAN
 - But uses quite detailed domain knowledge
- ◇ SHOP beats TLPLAN...(but not TALPlan)
 - Expects user to write a “program” for the domain in its language
 - » Explicit instructions on the order in which schemas are considered and concatenated



Types of Guidance

- ◇ **Declarative knowledge about desirable or undesirable solutions and partial solutions (SATPLAN+DOM; Cutting Planes)**
- ◇ **Declarative knowledge about desirable/undesirable search paths (TLPlan & TALPlan)**
- ◇ **A declarative grammar of desirable solutions (HTN)**

(largely) independent of the details of the specific planner

(affinities do exist between specific types of guidance and planners)

Planner specific. Expert needs to understand the specific details of the planner's search space

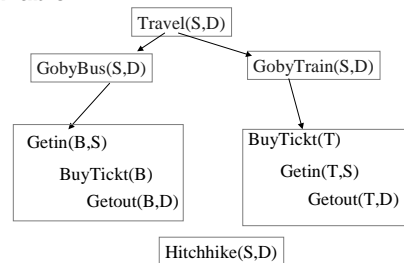
- ◇ **Procedural knowledge about how the search for the solution should be organized (SHOP)**
- ◇ **Search control rules for guiding choice points in the planner's search (NASA RAX)**

Ways of using the Domain Knowledge

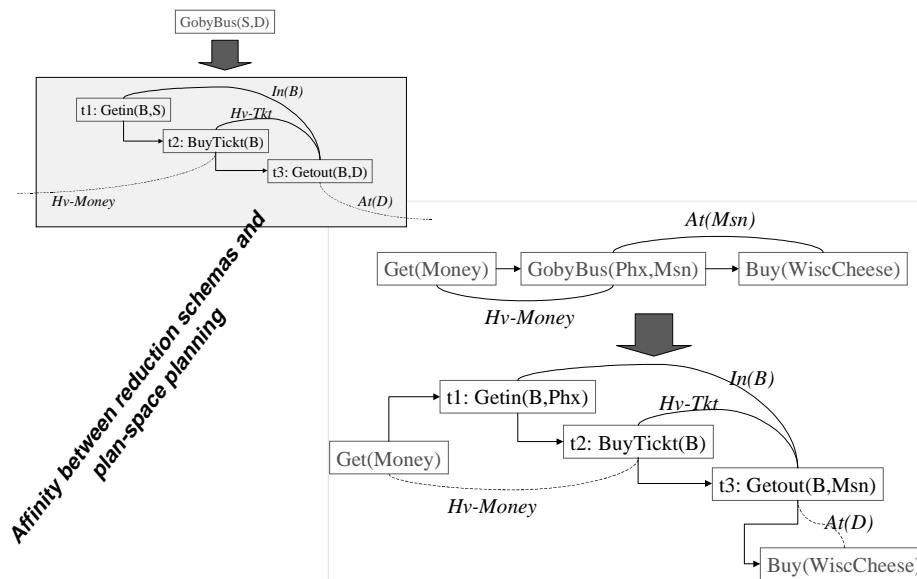
- ◇ **As search control**
 - HTN schemas, TLPlan rules, SHOP procedures
 - *Issues of Efficient Matching*
- ◇ **To prune unpromising partial solutions**
 - HTN schemas, TLPlan rules, SHOP procedures
 - *Issues of maintaining multiple parses*
- ◇ **As declarative axioms that are used along with other knowledge**
 - SATPlan+Domain specific knowledge
 - Cutting Planes (for ILP encodings)
 - *Issues of domain-knowledge driven simplification*
- ◇ **Folded into the domain-independent algorithm to generate a new domain-customized planner**
 - **CLAY**
 - *Issues of Program synthesis*

Task Decomposition (HTN) Planning

- ❖ The OLDEST approach for providing domain-specific knowledge
 - Most of the fielded applications use HTN planning
- ❖ Domain model contains non-primitive actions, and schemas for reducing them
- ❖ Reduction schemas are given by the designer
 - Can be seen as encoding user-intent
 - » *Popularity of HTN approaches a testament of ease with which these schemas are available?*
- ❖ Two notions of completeness:
 - Schema completeness
 - » (Partial Hierarchization)
 - Planner completeness



Modeling Action Reduction



Dual views of HTN planning

- ◇ Capturing hierarchical structure of the domain
 - Motivates top-down planning
 - » Start with abstract plans, and reduce them
- ◇ Many technical headaches
 - Respecting user-intent, maintaining systematicity and minimality
 - [Kambhampati et. al. AAAI-98]
 - » Phantomization, filters, promiscuity, downward-unlinearizability..
- ◇ Capturing expert advice about desirable solutions
 - Motivates bottom-up planning
 - » Ensure that each partial plan being considered is “legal” with respect to the reduction schemas
 - » Directly usable with disjunctive planning approaches
 - [Mali & Kambhampati, 98]
 - ◇ Connection to efficiency is not obvious

Relative advantages are still unclear...

[Barrett, 97]

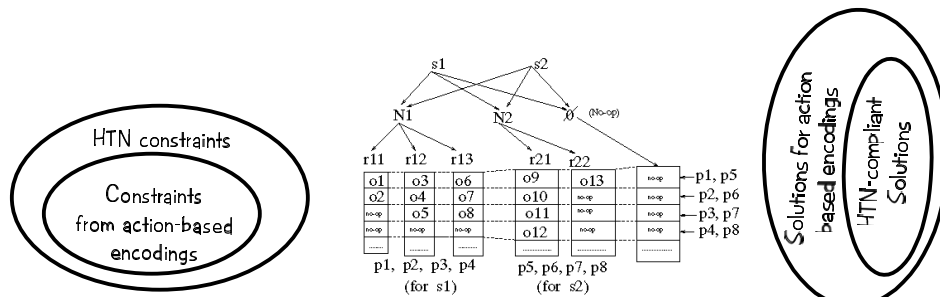
MAA: Recent Advances in AI Planning - A Half-Century

Subbarao Kambhampati

[Mali & Kambhampati, AIPS-98]

SAT encodings of HTN planning

- ◇ Abstract actions can be seen as disjunctive constraints
 - K-step encoding has each of the steps mapped to a disjunction of the non-primitive tasks
 - If a step s is mapped to a task N , then one of the reductions of N must hold (**The heart of encoding setup**)
 - + The normal constraints of primitive action-based encoding
 - » Causal encodings seem to be a natural fit (given the causal dependencies encoded in reduction schemas)



MAA: Recent Advances in AI Planning - A Half-Century

Subbarao Kambhampati

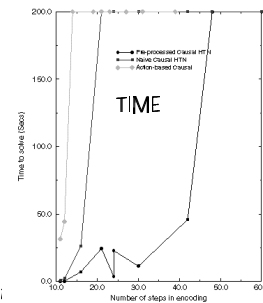
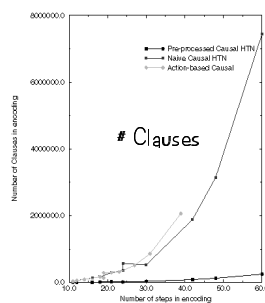
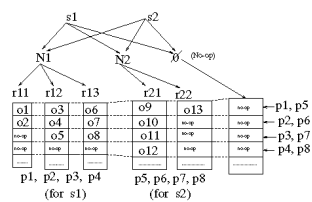
Solving HTN Encodings

Puzzle: How can increasing encoding sizes lead to efficient planning?

Abstract actions and their reductions put restrictions on the amount of step-action disjunction at the primitive level.

**--Reduction in step-action disjunction propagates
e.g. Fewer causal-link variables, Fewer exclusion clauses...**

Savings won't hold if each non-primitive task has MANY reductions



MA: Recent Advances in AI Planning - A Half-Century

Subbarao Kambhampati

[Nau et. al., 99]

Full procedural control: The SHOP way

Shop provides a
"high-level" programming
language in which the
user can code his/her
domain specific planner

-- Similarities to HTN
planning
-- Not declarative (?)

The SHOP engine can be
seen as an interpreter
for this language

```
(:method (travel-to ?y)
  (:first (at ?x)
    (at-taxi-stand ?t ?x)
    (distance ?x ?y ?d)
    (have-taxi-fare ?d))
  `(!hail ?t ?x) (!ride ?t ?x ?y)
  (pay-driver ,(+ 1.50 ?d)))
  ((at ?x) (bus-route ?bus ?x ?y))
  `(!wait-for ?bus ?x)
  (pay-driver 1.00)
  (!ride ?bus ?x ?y)))
```

Travel by bus only if going by taxi doesn't work out

Blurs the domain-specific/domain-independent divide

How often does one have this level of knowledge about a domain?

MA: Recent Advances in AI Planning - A Half-Century

Subbarao Kambhampati

Non-HTN Declarative Guidance

Invariants: *A truck is at only one location*

$$at(truck, loc1, I) \ \& \ loc1 \neq loc2 \Rightarrow \sim at(truck, loc2, I)$$

Optimality: *Do not return a package to the same location*

$$at(pkg, loc, I) \ \& \ \sim at(pkg, loc, I+1) \ \& \ I < J \Rightarrow \sim at(pkg, loc, j)$$

Simplifying: *Once a truck is loaded, it should immediately move*

$$\sim in(pkg, truck, I) \ \& \ in(pkg, truck, I+1) \ \& \ at(truck, loc, I+1) \Rightarrow \\ \sim at(truck, loc, I+2)$$

Once again, additional clauses first increase the encoding size
but make them easier to solve after simplification
(unit-propagation etc).

Rules on desirable State Sequences: TLPlan approach

TLPlan [Bacchus & Kabanza, 95/98] controls a
forward state-space planner

Rules are written on state sequences
using the linear temporal logic (LTL)

LTL is an extension of prop logic with temporal modalities

U	until	[]	always
O	next	<>	eventually

Example:

If you achieve on(B,A), then preserve it until On(C,B) is achieved:

$$[] (on(B,A) \Rightarrow on(B,A) \ U \ on(C,B))$$

TLPLAN Rules can get quite baroque

Good towers are those that do not violate any goal conditions

$$\begin{aligned} \text{goodtower}(x) &\triangleq \text{clear}(x) \wedge \text{goodtowerbelow}(x) \\ \text{goodtowerbelow}(x) &\triangleq (\text{ontable}(x) \wedge \neg \text{GOAL}(\exists [y:\text{on}(x, y)] \vee \text{holding}(x))) \\ &\vee \exists [y:\text{on}(x, y)] \neg \text{GOAL}(\text{ontable}(x) \vee \text{holding}(x)) \wedge \neg \text{GOAL}(\text{clear}(y)) \\ &\wedge \forall [z:\text{GOAL}(\text{on}(x, z))] z = y \wedge \forall [z:\text{GOAL}(\text{on}(z, y))] z = x \\ &\wedge \text{goodtowerbelow}(y) \end{aligned}$$

Keep growing “good” towers, and avoid “bad” towers

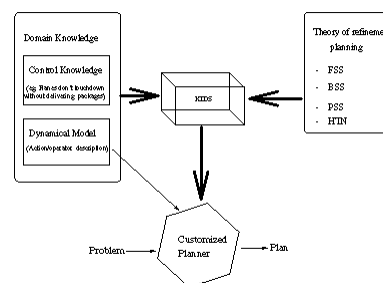
$$\begin{aligned} \Box & \left(\forall [x:\text{clear}(x)] \text{goodtower}(x) \Rightarrow \bigcirc \text{goodtowerabove}(x) \right. \\ & \wedge \text{badtower}(x) \Rightarrow \bigcirc (\neg \exists [y:\text{on}(y, x)]) \\ & \left. \wedge (\text{ontable}(x) \wedge \exists [y:\text{GOAL}(\text{on}(x, y))] \neg \text{goodtower}(y)) \right. \\ & \left. \Rightarrow \bigcirc (\neg \text{holding}(x)) \right) \end{aligned}$$

How “Obvious”
are these rules?

The heart of TLPlan is the ability to *incrementally*
and *effectively* evaluate the truth of LTL formulas.

Folding the Control Knowledge into the planner: CLAY approach

- ❖ Control knowledge similar to TLPlan’s
- ❖ Knowledge is folded using KIDS semi-automated software synthesis tool into a generic refinement planning template
 - Use of program optimizations such as
 - » Finite differencing
 - » Context-dependent & independent simplification
- ❖ Empirical results demonstrate that folding can be better than interpreting rules



Caveat: Current automated software
synthesis tools
have a very steep learning curve

Conundrums of user-assisted customization

- ◇ Which planners are easier to control?
 - Conjunctive planners are better if you have search control knowledge
 - » Forward State Space (according to TLPlan)
 - » Plan-space planners (according to HTN approaches)
 - Disjunctive planners are better if your knowledge can be posed as additional constraints on the valid plans
 - » Which SAT encoding?
 - HTN knowledge is easier to add on top of causal encodings
- ◇ Which approach provides the best language for expressing domain knowledge for the lay user?
 - *(Mine--no, no, Mine!)*
- ◇ What type of domain knowledge is easier to validate?
- ◇ When does it become “cheating”/ “wishful-thinking”
 - Foolish not to be able to use available knowledge
 - Wishful to expect deep procedural knowledge...

Automated Customization of Planners

- ◇ Domain pre-processing
 - Invariant detection; Relevance detection; Choice elimination, Type analysis
 - » STAN/TIM, DISCOPLAN etc.
 - » RIFO; ONLP
- ◇ Abstraction
 - » ALPINE; ABSTRIPS, STAN/TIM etc.
- ◇ Learning Search Control rules
 - » UCPOP+EBL,
 - » PRODIGY+EBL, (Graphplan+EBL)
- ◇ Case-based planning (plan reuse)
 - » DerSNLP, Prodigy/Analogy

Most of the work has been done in the context of Conjunctive Refinement Planners

Symmetry & Invariant Detection

- ◇ **Generate potential invariants and test them**
 - DISCOPLAN [Gerevini et. al.]
 - » Allows detection of higher-order mutexes
 - Rintanen's planner
 - » Uses model-verification techniques
 - STAN/TIM
 - » Type analysis of the domain is used to generate invariants
 - ONLP (Peot & Smith)
 - » Use operator graph analysis to eliminate non-viable choices

Abstraction

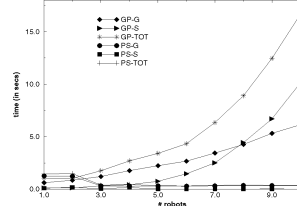
- ◇ **Idea**
 - Abstract some details of the problem or actions.
 - Solve the abstracted version.
 - Extend the solution to the detailed version
- ◇ **Precondition Abstraction**
 - Work on satisfying *important* preconditions first
 - » Importance judged by:
 - Length of plans for subgoals [ABSTRIPS, PABLO]
 - Inter-goal relations [ALPINE]
 - Distribution-based [HighPoint]
 - Strong abstractions (with downward refinement property) are rare
 - Effectiveness is planner-dependent
 - » Clashes with other heuristics such as “*most constrained first*”

Example: Abstracting Resources

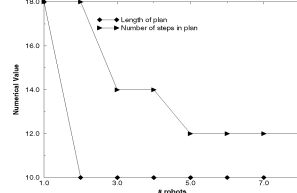
- ◇ Most planners thrash by addressing planning and scheduling considerations together
 - Eg. Blocks world, with multiple robot hands
- ◇ Idea: Abstract resources away during planning
 - Plan assuming infinite resources
 - Do a post-planning resource allocation phase
 - Re-plan if needed

(with Biplav Srivastava)

Performance of Graphplan v/s Planning+Scheduling in the 6-Block Shuffle Problem in Blocks World



A Look into Plans by Graphplan in the Blocks World domain



MA: Recent Advances in AI Planning - A Unit&od V...

Subbarao Kambhampati

Learning Search Control Rules with EBL

Explain leaf level failures

Regress the explanations to compute interior node failure explanations

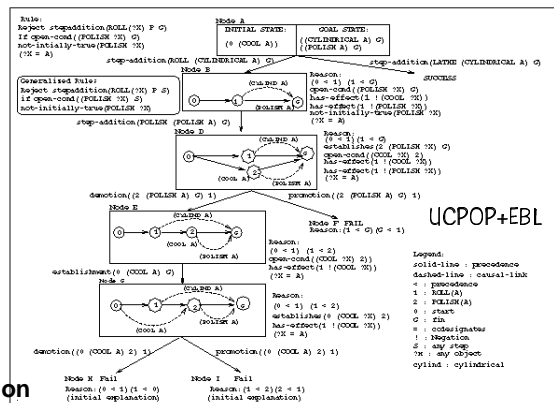
Use failure explanations to set up control rules

Problems:

- Most branches end in depth-limits
 - >No analytical explanation
 - >Use preference rules?
- THE Utility problem
 - >Learn general rules
 - >Keep usage statistics & prune useless rules

(Kambhampati, Katukam, Qu, 95)

MA: Recent Advances in AI Planning - A Unit&od V...



If Polished(x)@S &
~Initially-True(Polished(x))
Then
REJECT
Stepadd(Roll(x),Cylindrical(x)@s)

Subbarao Kambhampati

Example Rules (Learned)

UCPOP

If Polished(x)@S &
 ~Initially-True(Polished(x))
Then
REJECT
 Stepadd(Roll(x),Cylindrical(x)@s)

Pruning rule

Prodigy

If (and (current-node *node*)
 (candidate-goal *node*
 (shape *obj* Cyl))
 (candidate-goal *node*
 (surface-condition *obj*
 polished)))
Then: Prefer (shape *obj* cyl) to
 (surface-condition *obj* polished)

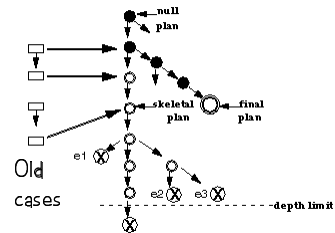
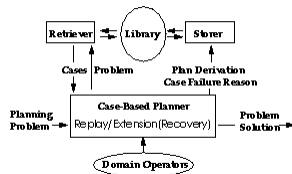
Preference rule

Case-based Planning Macrops, Reuse, Replay

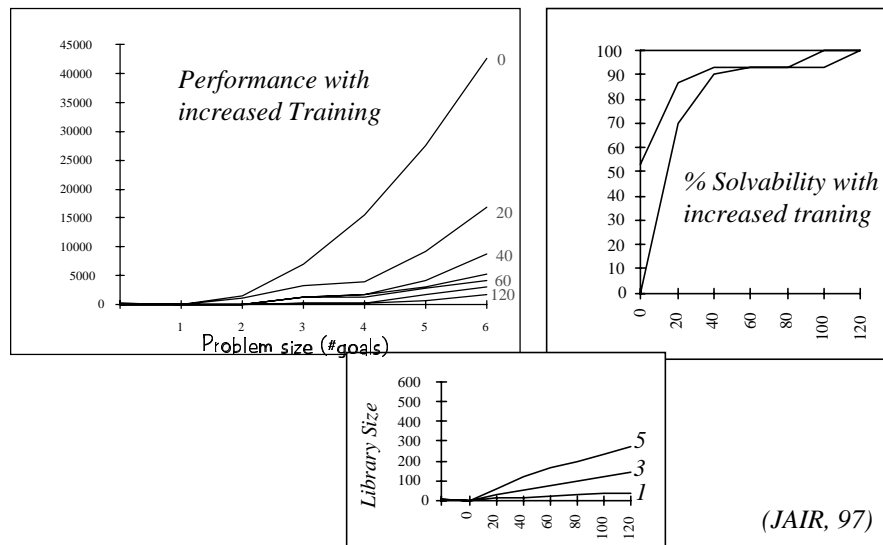
- ◇ Structures being reused
 - Opaque vs. Modifiable
 - Solution vs. Solving process (derivation)
- ◇ Acquisition of structures to be reused
 - Human given vs. Automatically acquired
- ◇ Mechanics of reuse
 - Phased vs. simultaneous
- ◇ Costs
 - Storage & Retrieval costs; Solution quality

Case-study: DerSNLP

- ◇ Modifiable derivational traces are reused
- ◇ Traces are automatically acquired during problem solving
 - Analyze the interactions among the parts of a plan, and store plans for *non-interacting* subgoals separately
 - » Reduces retrieval cost
 - Use of EBL failure analysis to detect interactions
- ◇ All relevant trace fragments are retrieved and replayed before the control is given to from-scratch planner
 - Extension failures are traced to individual replayed traces, and their storage indices are modified appropriately
 - » Improves retrieval accuracy



DerSNLP: Results



Reuse in Disjunctive Planning

- ◇ **Harder to make a disjunctive planner commit to extending a specific plan first**
- ◇ **Options:**
 - Support opaque macros along with primitive actions
 - » Increases the size of k-step disjunctive plan
 - » But a solution may be found at smaller k
 - Modify the problem/domain specification so the old plan's constraints will be respected in any solution (Bailotti et. al.)
 - MAX-SAT formulations of reuse problem
 - » Constrain the encoding so that certain steps may have smaller step-action mapping and ordering choices
 - » Causal encodings provide better support

[with Amol Mali]

Summary

- ◇ **Focus on “Neo-classical” planning**
 - Refinement planning provides a unifying view
 - Conjunctive Refinement Planners
 - Disjunctive Refinement Planners
 - » Refinement
 - » Solution Extraction
 - Direct vs. compilation to CSP/SAT
- ◇ **Heuristics/Optimizations**
 - Relaxed Problem heuristics
 - Consistency enforcement
- ◇ **Customization of planners**
 - User-assisted
 - Automated
- ◇ **Related approaches to non-classical planning**

But who is winning???

(Using AIPS-2000 Competition as a benchmark)

- ◇ 15 planners took part in AIPS-2000 competition
 - Representatives from all types of planners we discussed
- ◇ Competition focussed on pure classical problems
 - Separate tracks for domain-independent and user-customized planners
- ◇ Domain independent:
 - FF (Forward state-space planning; Heuristics based on planning graphs, local search)
- ◇ Customized:
 - TALPlan (control knowledge about good and bad state sequences, applied to a forward state-space planner)
- ◇ Million Dollar question: Will the line-up be same for neo-classical planning (durative actions, continuous change etc?)

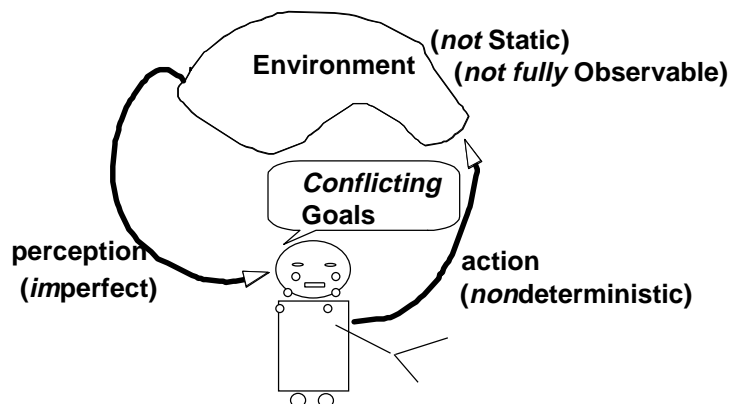
Status

- ◇ Exciting times...
 - Many approaches with superior scale-up capabilities
 - » Broadened views of planning
 - Many influences (CSP; OR; MDP; SCM)
- ◇ Ripe for serious applications
 - VICAR [JPL]; DeepSpace monitoring [NASA/AMES]
 - Plant monitoring [Ayslett et. al.]
 - Manufacturing Process Planning [Nau et. al.; Kambhampati et. al.]
 - Supply chain management/ Logistics
 - » Industrial “Planning” does not have to the optimal scheduling of an inoptimal action selection!

Resources

- ◇ **Mailing Lists**
 - Planning list digest
 - » <http://rakaposhi.eas.asu.edu/planning-list-digest>
 - U.K. P & S List
 - » <http://www.salford.ac.uk/planning/home.html>
- ◇ **Special Conferences**
 - Intl. Conf. on AI Planning & Scheduling
 - » <http://www.isi.edu/aips> (April 2000, Breckenrdige, CO)
 - European Conference on Planning
 - » <http://www.informatik.uni-ulm.de/ki/ecp-99.html>
 - Also, AAAI, IJCAI, ECAI, Spring/Fall Symposia
- ◇ **Courses**
 - ASU Planning Seminar Online Notes (2000,1999, 1997, 1995,1993)
 - » <http://rakaposhi.eas.asu.edu/planning-class.html>

Adapting to Incompleteness, Uncertainty and Dynamism



The most efficient approaches to all these *ills* are still based on classical planning ideas...

Incomplete Information

- ◇ **PROBLEM:** Values of some state variables are unknown;
There are actions capable of sensing (some) of them.
 - If k boolean state variables are unknown, then we are in one of 2^k initial states
 - Two naïve approaches
 - » **PLAN/SENSE/EXECUTE** : Solve each of the 2^k problems separately; At the execution time sense the appropriate variables, and execute the appropriate plan
 - » **SENSE/PLAN/EXECUTE**: First sense the values of the variables. Solve the problem corresponding to the sensed values
 - Problems with naïve approaches
 - » Solving the 2^k problems separately is wasteful
 - Shared structure (Tree structured plans)
 - » Not all variables may be observable (or worth observing)
 - Conformant planning
 - (Find non-sensing plans that work in all worlds)
 - Irrelevant variables (Goal directed planning)

Incomplete Information: Some Implemented Approaches

- ◇ **Conjunctive planners**
 - CNLP [Peot & Smith; 92] CASSANDRA [Pryor & Collins, 96] add sensing actions to UCPOP; support tree-shaped plans
 - SADL/PUCCINI [Golden & Weld; 96-98] integrates planning and sensing in the context of a UCPOP-like planner
- ◇ **Disjunctive planners**
 - CGP [Smith & Weld, 98] supports conformant planning on Graphplan
 - SGP [Weld et. al., 98] supports conditional planning on Graphplan
 - » One plan-graph per possible world/ Interactions among plangraphs captured through induced mutexes
 - [Rintanen, 99] converts conditional planning to QBF encodings

Dynamic Environments

- ◇ **PROBLEM:** The world doesn't sit still. Blind execution of a "correct" plan may not reach goals

- ◇ **APPROACHES:**

- **PLAN/MONITOR/REPLAN:** Monitor the execution; when the observed state differs from the expected one, REPLAN
 - » Replanning is like reuse except there is added incentive for minimal modification
 - Easy to support with conjunctive plan-space planners
 - PRIAR [Kambhampati; 92]; DerSNLP [Ihrig & Kambhampati, 97]
 - Possible to support with disjunctive causal encodings
 - [Mali, 1999]
 - **MONITOR/REACT/LEARN:**
 - » Policy construction (Universal plans)...
 - **MODEL OTHER AGENTS CAUSING CHANGE:**
 - » Collaborative/Distributed planning

Stochastic Actions

- ◇ **PROBLEM:** Action effects are stochastic
 - Actions transform *state-distributions* to state-distributions
 - Maximize "probability" of goal satisfaction
 - Plan assessment itself is hard
- ◇ **APPROACHES:**
 - Conjunctive planners
 - » BURIDAN [Hanks et. al., 95] uses UCPOP techniques to put candidate plans together and assesses them
 - *Multiple /redundant supports*
 - Disjunctive planners
 - » Pgraphplan [Blum & Langford, 98] modifies Graphplan to support some forms of stochastic planning
 - Forward search; value propagation; Envelope extension
 - » Maxplan [Majercik & Littman, 98] uses EMAJSAT encodings to solve stochastic planning problems
 - Chance variables & Choice variables. Equivalence classes of models that have the same values of choice variables. Find the equivalence class with maximum probability mass.

Complex & Conflicting Goals

❖ Problems & Solutions:

- Goals that have temporal extent (stay alive)
 - » UCPOP, TLPlan, TGP [Smith & Weld, 99]
- Goals that have mutual conflicts (Sky-dive & Stay Alive)
- Goals that take cost of achievement into account
- Goals that admit degrees of satisfaction (Get rich)
 - » Branch & Bound approaches; MAXSAT approaches
 - Pyrrhus [Williamson & Hanks; 92]

Decision Theoretic Approaches:

Model goals in terms of factored reward functions
for Markov Decision Processes

--Can utilize tricks and insights from classical planning
[Boutilier, Hanks, Dean; JAI 99]

CSP/SAT/TCSP Review

(very) Quick overview of CSP/SAT concepts

Constraint Satisfaction Problem (CSP)

– Given

- » A set of discrete variables
- » Legal domains for each of the variables
- » A set of constraints on values groups of variables can take

- Find an assignment of values to all the variables so that none of the constraints are violated

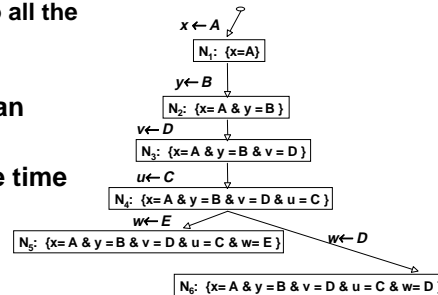
SAT Problem = CSP with boolean variables

TCSP = CSP where variables are time points and constraints describe allowed distances

$x, y, u, v: \{A, B, C, D, E\}$
 $w: \{D, E\} \mid \{A, B\}$
 $x=A \Rightarrow w \neq E$
 $y=B \Rightarrow u \neq D$
 $u=C \Rightarrow l \neq A$
 $v=D \Rightarrow l \neq B$

A solution:

$x=B, y=C, u=D, v=E, w=D, l=B$



Important ideas in solving CSPs

Variable order heuristics:

Pick the most constrained variable

- Smallest domain, connected to most other variables,
- causes most unit propagation,
- causes most resource contention,
- has the most distance etc...

Value ordering heuristics

Pick the least constraining value

Consistency enforcement

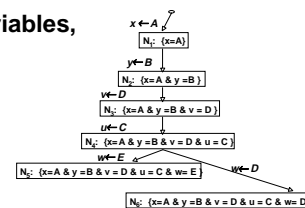
k-consistency; adaptive consistency. (pre-processing)

Forward Checking, unit propagation during search (dynamic)

Search/Backtracking

DDB/EBL: Remember and use interior node failure explanations

Randomized search

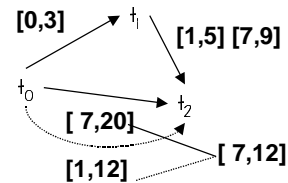


$x, y, u, v: \{A, B, C, D, E\}$
 $w: \{D, E\} \mid \{A, B\}$
 $x=A \Rightarrow w \neq E$
 $y=B \Rightarrow u \neq D$
 $u=C \Rightarrow l \neq A$
 $v=D \Rightarrow l \neq B$

[Dechter, Schwab, AU 1997]

Temporal Constraint Satisfaction Problem (TCSP)

- ◇ Variables correspond to time points
- ◇ Constraints restrict allowable distances between time points
- ◇ Constraints can be represented as sets of intervals
 - Simple temporal problem (STP)
 - » All constraints have at most one interval (TRACTABLE!!!)
- ◇ Solution of general TCSP involves enforcing *path-consistency*
 - Approximations such as loose path consistency can be used to detect inconsistencies



$$(0 \leq t_1 - t_0 \leq 3)$$

$$(1 \leq t_2 - t_1 \leq 5) \vee (7 \leq t_2 - t_1 \leq 9)$$

$$c_{02} = (t_0 t_1) \circ (t_1 t_2) = [1,8][7,12] = [1,12]$$

$$c_{02} \cap (t_0 t_2) = [7,20] \cap [1,12] = [7,12]$$

MA: Recent Advances in AI Planning - A Half-Century

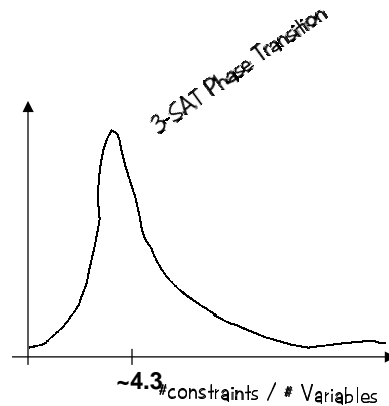
Subbarao Kambhampati

What makes CSP problems hard?

Assignments to individual variables that seem locally consistent are often globally infeasible, causing costly backtracking.

The difficulty of a CSP/SAT problem depends on

- Number of variables (propositions)
- Number of constraints (clauses)
- Arity of constraints
- Degree of local consistency



MA: Recent Advances in AI Planning - A Half-Century

Subbarao Kambhampati

Hardness & Local Consistency

- ◇ An n-variable CSP problem is said to be k-consistent iff every consistent assignment for (k-1) of the n variables can be extended to include any k-th variable
 - » Directional consistency: Assignment to first k-1 variables can be extended to the k-th variable
 - » Strongly k-consistent if it is j-consistent for all j from 1 to k
- ◇ Higher the level of (strong) consistency of problem, the lesser the amount of backtracking required to solve the problem
 - A CSP with strong n-consistency can be solved without any backtracking
- ◇ We can improve the level of consistency of a problem by explicating implicit constraints
 - Enforcing k-consistency is of $O(n^k)$ complexity
 - » Break-even seems to be around $k=2$ (“arc consistency”) or 3 (“path consistency”)
 - » Use of directional and partial consistency enforcement techniques