“STRIPS” Planning

- Set of *operators*, where each operator has
  - Set of *parameters*
  - Set of *preconditions*
  - Set of *effects*, consisting of
    add effects and
    delete effects.

- Set of *objects* to instantiate operator’s parameters

  fully instantiated operator $\equiv$ *action*

- Set of propositions representing *initial state*

- Set of propositions representing *goals*

Planning problem: Find sequence of actions that,
starting in initial state,
achieve all the goals

Approaches to STRIPS planning

- Search through space of *world states*
  - *forward* search,
  - *regression* search
  - *bi-directional* search
  - *means-ends* analysis
  - . . .

- Search through space of *plans*
  - *total order* planning
  - *partial order* planning

- Search through *planning graph*

GraphPlan Approach

1. Construct a "PlanGraph" that contains
   all valid plans
   + other stuff (invalid plans)
   up to a maximum depth

2. Search PlanGraph for valid plan
   ...then return that plan

Simple Cake-Eating Domain

- Initial: `HaveCake` $\wedge$ $\neg$`EatenCake`

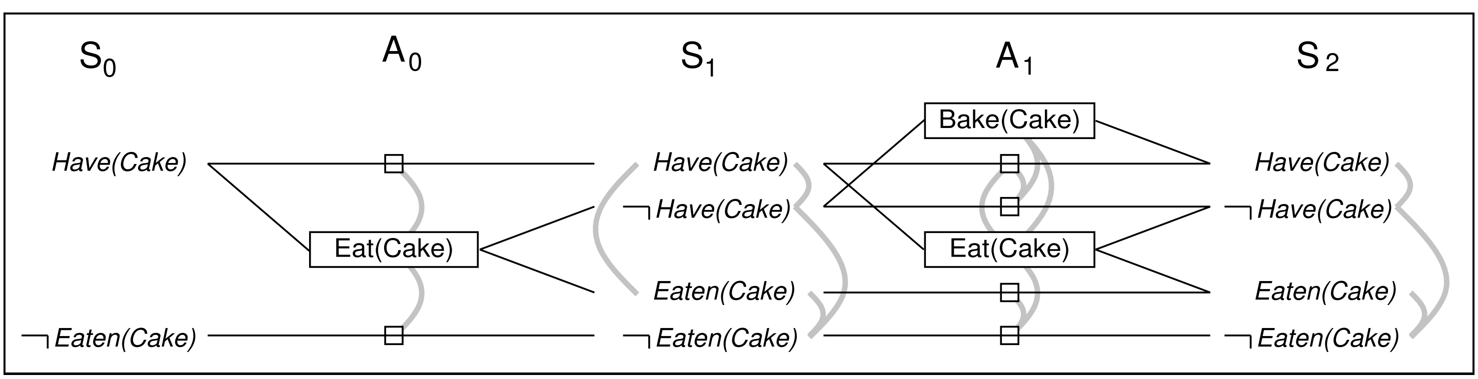
- Goal: `HaveCake` $\wedge$ `EatenCake`

- Actions:

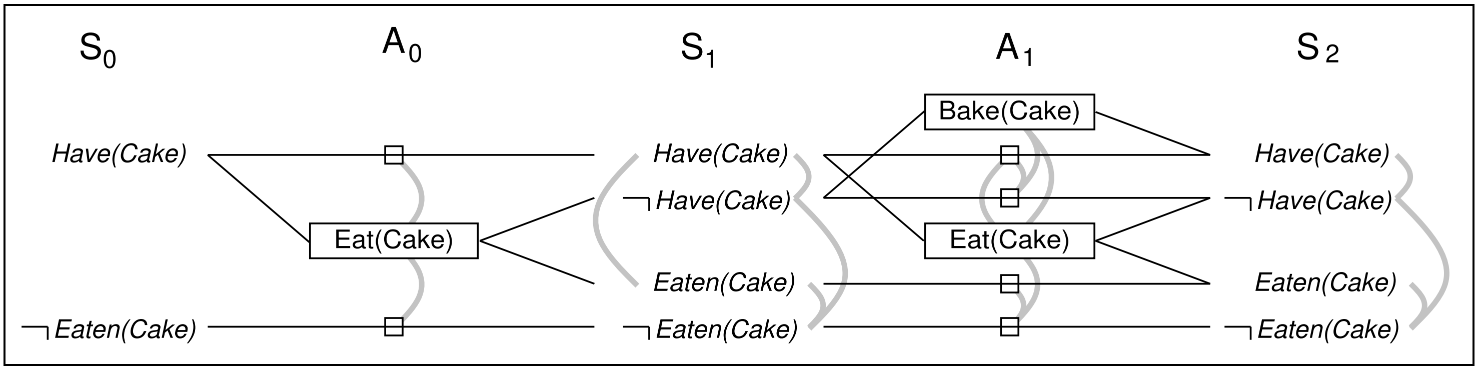
$$\texttt{Op}\begin{pmatrix} \texttt{Eat} & \\ \texttt{PreC:} & \texttt{HaveCake} \\ \texttt{Eff:} & \neg\texttt{HaveCake} \wedge \texttt{EatenCake} \end{pmatrix}$$

$$\texttt{Op}\begin{pmatrix} \texttt{Bake} & \\ \texttt{PreC:} & \neg\texttt{HaveCake} \\ \texttt{Eff:} & \texttt{HaveCake} \end{pmatrix}$$

- PlanGraph



Parts of a PlanGraph



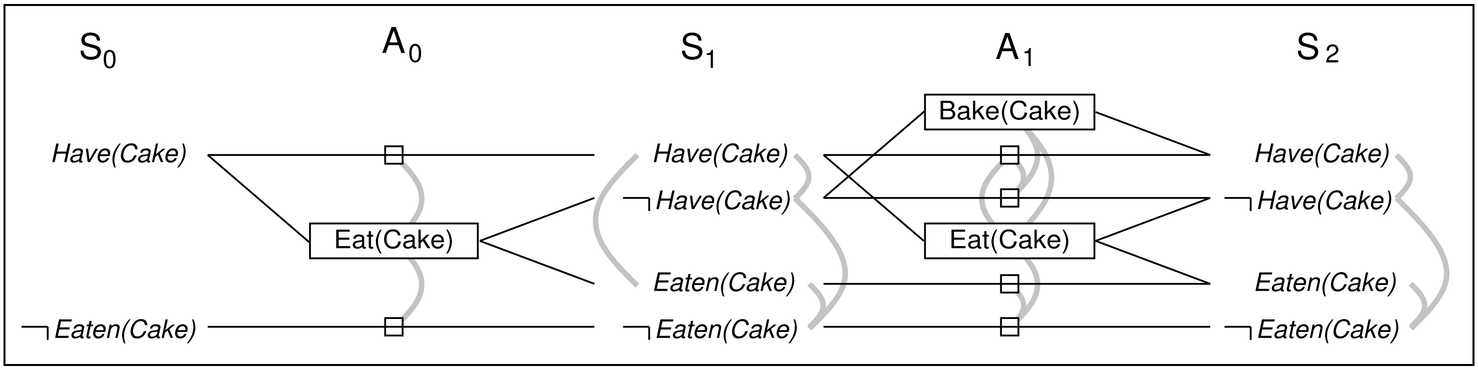
"2-leveled" Graph $\langle \mathcal{S}_0, \mathcal{A}_0, \mathcal{S}_1, \mathcal{A}_1, \ldots \rangle$

- S_0: propositions in initial state
- A_i: each action whose preconditions all occur in level S_{i-1}
- S_i: each prop'n that is ADDed/DELETEd by
  - $\star$ an action in level A_i
  - $\star$ a "No-Op" (persistance)

- **Mutex** links
  - $\star$ between actions in level A_i
  - $\star$ between propositions in level S_i

  "**mut**ually **ex**clusive"
  "cannot occur in same plan"

Mutex Conditions#1: Actions



Between 2 actions O_1 and O_2, same level A_i:

- *Inconsistent effects*

  O_1:*Eff* negates O_2:*Eff*

  `EatenCake`, `NoOp(HaveCake)` disagree wrt "`HaveCake`"

`EatenCake:Eff`	=	¬HaveCake
`NoOp(HaveCake):Eff`	=	HaveCake

- *Interference*

  O_1:*Eff* negates O_2:*PreC*

  `EatenCake` interfers with `NoOp(HaveCake)`:

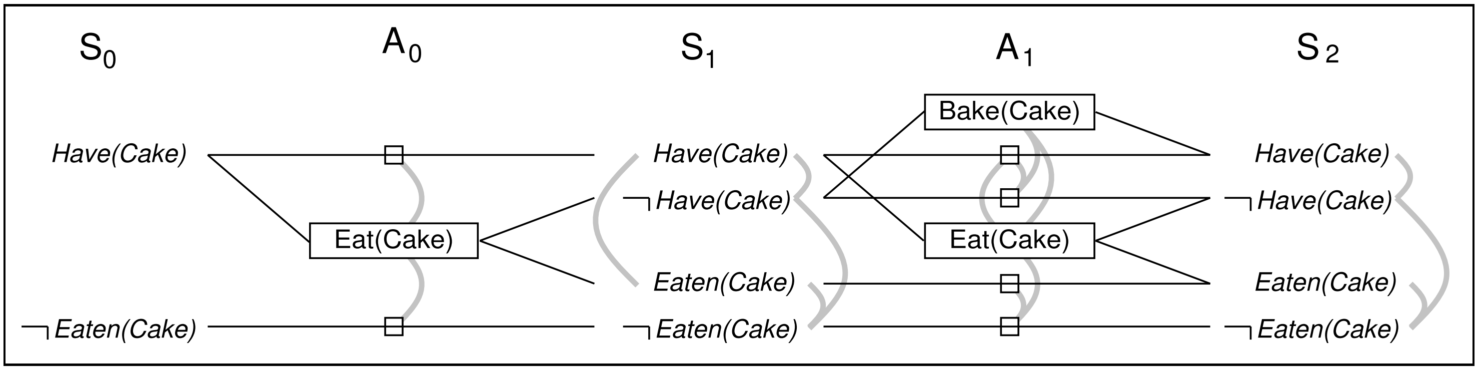
`EatenCake:Eff`	=	¬`HaveCake`
`NoOp(HaveCake):PreC`	=	`HaveCake`

- *Competing Needs*

  O_1:*PreC* negates O_2:*PreC*

`Bake:PreC`	=	¬`HaveCake`
`Eat:PreC`	=	`HaveCake`

Mutex Conditions#2: Propositions



Between 2 propositions ρ_1 and ρ_2, same level S_i:

- *Negation*
  $\rho_1 = \neg\rho_2$

- *Inconsistent Support*
  Every action achieving ρ_1 (from S_{i-1})
  is mutex with every action achieving ρ_2

  In S_1: `HaveCake` mutex `EatenCake` as
  only way to achieve `HaveCake`:
  `NoOp(HaveCake)`
  is mutex with only way to achieve `EatenCake`:
  `Eat`

  N.b.: Not mutex at S_2 !

Planning Graphs

- A *valid plan* is "2-leveled" graph
  - two kinds of nodes
    (propositions, actions)
    alternates: proposition level, action level
  - 5 kinds of edges
    - ⋆ precondition $(S_i \to A_i)$
    - ⋆ add effect $(A_i \to S_{i+1})$
    - ⋆ delete effect $(A_i \to S_{i+1})$
    - ⋆ mutex-action $(A_i \leftrightarrow A_i)$
    - ⋆ mutex-prop $(S_i \leftrightarrow S_i)$
  - Include action O at action-level A_i
    if all preconditions at proposition-level S_i
  - Include proposition ρ at proposition-level S_i
    if it is add/delete effect of action $O \in A_{i-1}$
    (including *no-op* actions)

  Restriction:
  Allow actions $O_1,\ O_2$ at same time t
  ONLY if don't interfere with each other

- *PlanningGraph* $\approx$ *valid plan* but
  without no-interfere restriction

GraphPlan **Algorithm**

```
function Graphplan( problem ) returns solution or failure
   graph ← Initial-Planning-Graph(problem)
   goals ← Goals[problem]
   loop do
      if goals all non-mutex in last level of graph then do
         solution ← Extract-Solution(graph, goals, Length(graph))
         if solution ≠ failure then return solution
         else if No-Solution-Possible(graph) then return failure
      graph ← Expand-Graph(graph, problem)
   end
```

Flat-Tire Domain

Fl= Flat; Sp= Spare; Ax= Axel; Tr= Trunk; Gr= Ground

- Initial: At(Fl, Ax) $\wedge$ At(Sp, Tr)
- Goal: At(Sp, Ax)
- Actions:

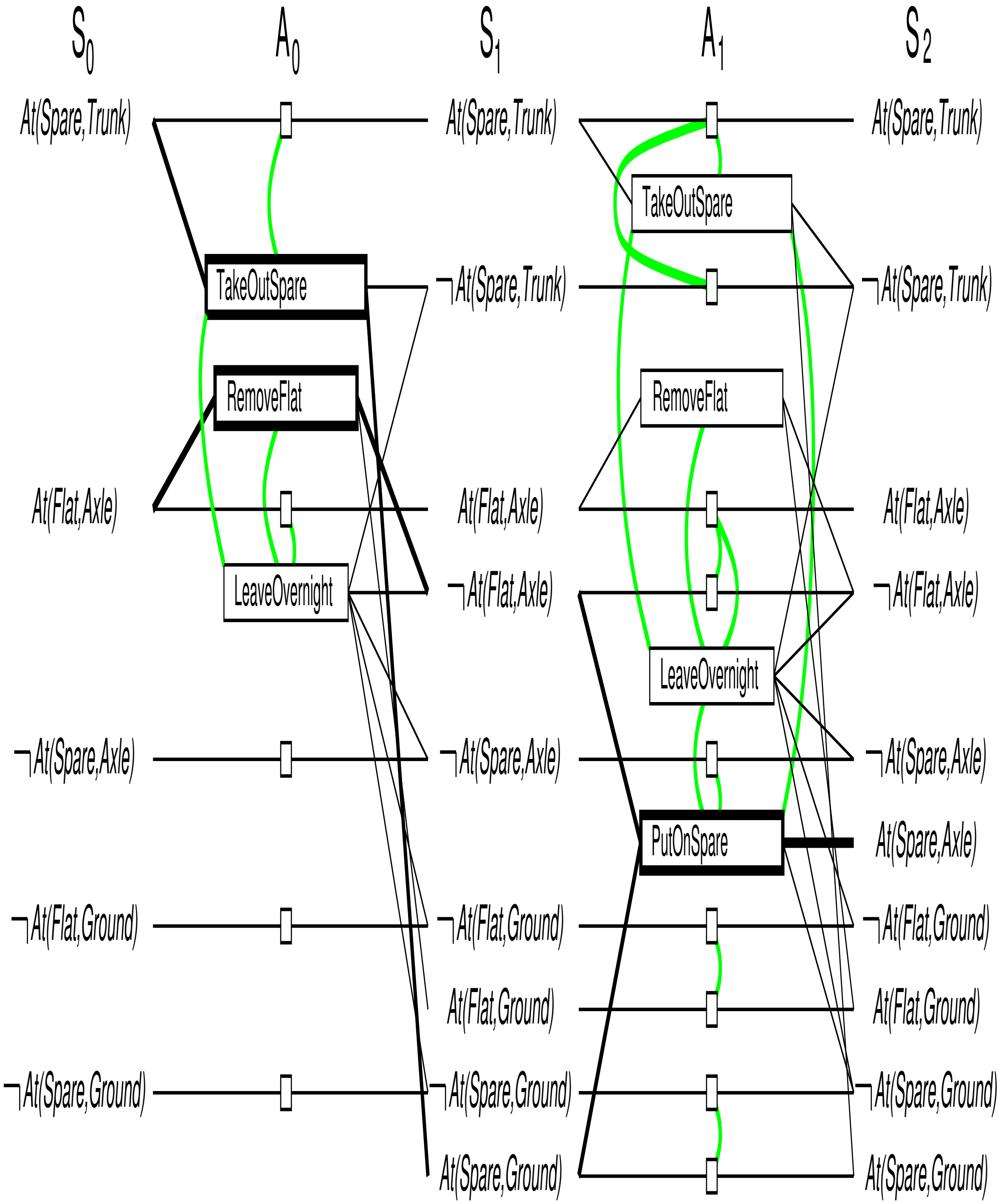
$$\text{Op}\begin{pmatrix} \texttt{TakeOutSpare} & \\ \texttt{PreC:} & \texttt{At(Sp, Tr)} \\ \texttt{Eff:} & \neg\texttt{At(Sp, Tr)} \wedge \texttt{At(Sp, Gr)} \end{pmatrix}$$

$$\text{Op}\begin{pmatrix} \texttt{RemoveFlat} & \\ \texttt{PreC:} & \texttt{At(Fl, Ax)} \\ \texttt{Eff:} & \neg\texttt{At(Fl, Ax)} \wedge \texttt{At(Fl, Gr)} \end{pmatrix}$$

$$\text{Op}\begin{pmatrix} \texttt{PutOnSpare} & \\ \texttt{PreC:} & \texttt{At(Sp, Gr)} \wedge \neg\texttt{At(Fl, Ax)} \\ \texttt{Eff:} & \neg\texttt{At(Sp, Gr)} \wedge \texttt{At(Sp, Ax)} \end{pmatrix}$$

$$\text{Op}\begin{pmatrix} \texttt{LeaveOverNight} & \\ \texttt{PreC:} & \{\} \\ \texttt{Eff:} & \neg\texttt{At(Sp, Gr)} \wedge \neg\texttt{At(Sp, Ax)} \wedge \neg\texttt{At(Sp, Tr)} \\ & \wedge \neg\texttt{At(Fl, Gr)} \wedge \neg\texttt{At(Fl, Ax)} \end{pmatrix}$$

Flat-Tire in GraphPlan



Trace of GraphPlan Algorithm #1

- S_0: initial facts (include ¬facts)

- As `At(Sp,Ax)` $\notin S_0$
  do not call Extract-Solution

- Expand-Graph forms A_0 with
  - ⋆ 3 "real" actions
  - ⋆ 5 no-op actions;

  S_1 is effects

  Expand-Graph then finds
  - ⋆ 4 action-mutex within A_0
  - ⋆ 4 prop-mutex within S_1

- As `At(Sp,Ax)` $\notin S_1$
  do not call Extract-Solution

- Expand-Graph forms A_1 with
  - ⋆ 4 "real" actions
  - ⋆ 7 no-op actions

  S_2 is effects

Mutex wrt FlatTire

- *Inconsistent Effects*

  `RemoveSpare` + `LeaveOvernight`

  `RemoveSpare:Eff` = `At(Sp,Gr)`

  `LeaveOvernight:Eff` = ¬`At(Sp,Gr)`

- *Inteference*

  `RemoveFlat` + `LeaveOvernight`

  `RemoveFlat:PreC` = `At(Sp,Ax)`

  `LeaveOvernight:Eff` = ¬`At(Sp,Ax)`

- *Competing Needs*

  `RemoveFlat` + `PutOnSpare`

  `RemoveFlat:PreC` = `At(Fl,Ax)`

  `PutOnSpare:Eff` = ¬`At(Fl,Ax)`

- *Inconsistent Support*

  `At(Sp,Ax)` + `At(Fl,Ax)` in S_2

  `At(Sp,Ax)` by `PutOnSpare`

  `At(Fl,Ax)` by `NoOp[At(Fl,Ax)]`

  and

  `PutOnSpare` mutex `NoOp[At(Fl,Ax)]`

  (Can't put 2 objects in same place at same time)

Trace of GraphPlan Algorithm #2

- "All" goal literals, `At(Sp, Ax)`, in S_2
  none are mutex . . .

- So there MAY be solution
  . . . call Extract-Solution

Extract-Solution(. . .)
  Let G_n be the GOAL at last level, S_n
  For each $i = n..1$
    ⋆ Let H_i be a conflict-free subset of A_{i-1},
      that covers G_i (in S_i)
    ⋆ Let G_{i-1} be preconditions of H_i
  . . . until reach state in S_0 satisfying all goals

Action-set H is "conflict-free"

$\equiv$

no pair of H are mutex, and

no pair of preconditions (in G) are mutex

Trace of Extract-Solution

- $G_2 \quad = \quad \{\ \texttt{At(Sp,Ax)}\ \}$
  $H_2 \quad = \quad \{\ \texttt{PutOnSpare}\ \}$

- $G_1 \quad = \quad \{\ \texttt{At(Sp,Gr)},\ \neg\texttt{At(Fl,Ax)}\ \}$
  What is H_1?
  - Achieve `At(Sp,Gr)` by `TakeOutSpare`
  - Achieve ¬`At(Fl,Ax)` by
    #1. `LeaveOvernight`
    #2. `RemoveFlat`

    But not #1, as
    `LeaveOvernight` is mutex with `TakeOutSpare`

  $\Rightarrow H_1 \quad = \quad \{\ \texttt{TakeOutSpare},\ \texttt{RemoveFlat}\ \}$

- $G_0 = \{\ \texttt{At(Sp,Tr)},\ \texttt{At(Fl,Ax)}\ \}$
  As in $G_0 \subset S_0$, DONE!

Extending PlanGraph

Add action level A_i:

ForEach action$^{(*)}$ O

If O's preconditions all true in prop-level S_i,
and NOT mut-ex,

Then add O to level A_i
include precondition-links
create mutex (O:actions-I-am-exclusive-of)

Add prop-level S_{i+1}:

ForEach effect ρ of each action in action-level A_i

Add ρ to prop-level S_{i+1}

Add $S \leftarrow \rho$ add- or delete- links

Mark ρ_1, ρ_2 as mutex if
each way of generating ρ_1 is mutex to
each way of generating ρ_2

$^{(*)}$ each instantiation of each operator; including "no-op"s

Correctness

Graphplan is sound and complete:

* any plan Graphplan finds is a legal plan
* if $\exists$ legal plan then Graphplan will find one.

Theorem:	If $\exists$ valid plan using $\leq t$ time steps, then plan is subgraph of (depth-t) Planning Graph.

+

If Goals not satisfiable by any valid plan,
then GraphPlan will halt, w/failure, in finite time.

(extends most partial-order planners)

Leveling Off

- GraphPlan ≈ Iterative deepening
  When to stop??

- **Lemma:** If no valid plan exists, then $\exists$ a prop-level S_n s.t. all future proposition levels are identical to S_n
  - Identical ≡ same propositions, mutual exclusions
  - graph has "*leveled off* after S_n"

- **Corollary:** No solution exists if
  - a goal does not appear in S_n or
  - S_n has mutually exclusive goals

- Subtlety:
  { `on(A,B)`, `on(B,C)`, `on(C,A)` }

Termination Condition

- Let S_i^t denote set of memoized goal sets at level i after an unsuccessful stage t

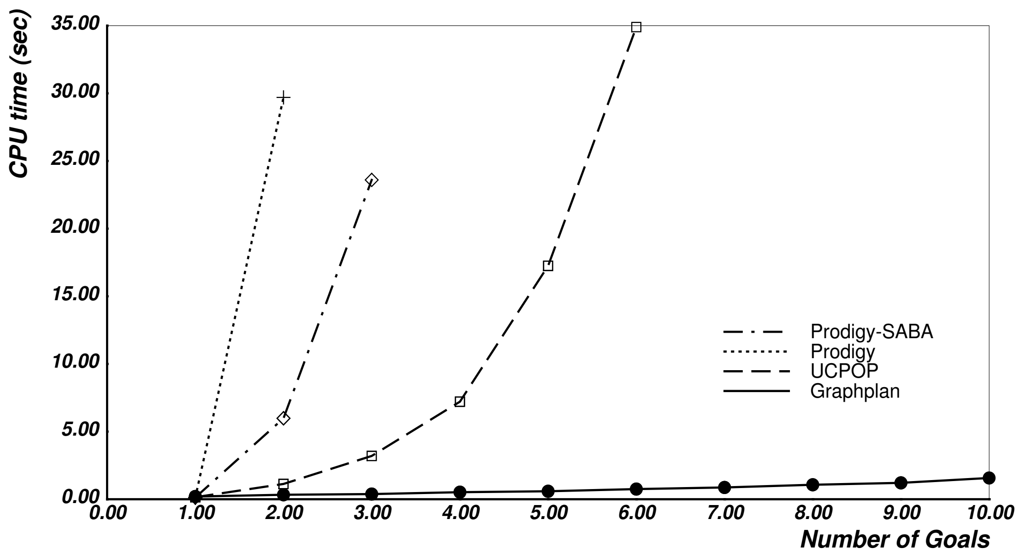
- **Theorem:** If the graph has leveled off at level n and stage t has passed in which $|S_n^{t-1}| = |S_n^t|$, then no valid plan exists

Termination Proof

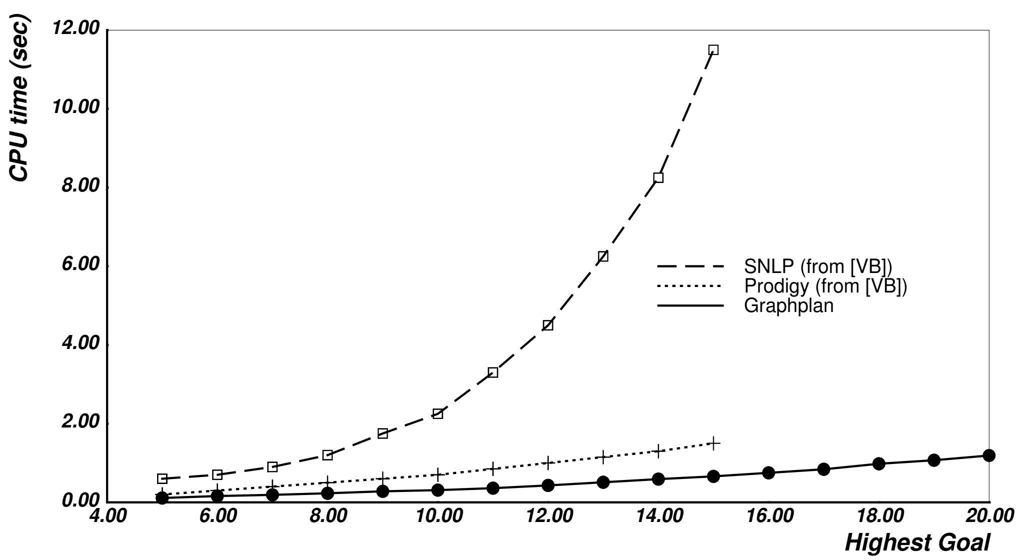
- As PlanGraph gets deeper. . .
  - Literals increase monotonically
  - Actions increase monotonically
  - Mutex decrease monotonically
    - ⋆ If O_1 and O_2 are mutex in A_k,
      then mutex in A_i $i = 1..k$ provided $O_1, O_2 \in A_i$
    - ⋆ If ρ_1 and ρ_2 are mutex in S_k,
      then mutex in S_i $i = 1..k$ provided $\rho_1, \rho_2 \in S_i$

- Only finite # of actions/literals,
  planning graph must eventually "level off"

Experimental Results



"2 Rockets Problem"



"Link-Repeat Problem"

Accounting for Graphplan's Efficiency

- Mutual exclusions

  (Most constraints are pair-wise mut-ex's;
  Propagating constraints prunes large part of space.)

- Consideration of parallel plans

  (Valid parallel plans are short, wrt total plan
  $\Rightarrow$ reduces cost of constructing pgraph, search)

- Memoizing

  (Many goal-sets appear > 1)

- Low-level costs

  (Graphplan avoid cost of instantiation during search)

Efficiency
Size of Planning Graph

Theorem: Consider planning problem with
n objects,
p propositions in initial state,
m operators,
each w/constant number of parameters
Let l be length of longest add list.
Then size of a t-level planning graph, and
time needed to create the graph,
are polynomial in n, m, p, l, and t.

- Empirically: exclusion relations most expensive part of graph creation

  Graph creation only significant in simple problems

⇒ As graph is small,
"finding mut-ex" is hard as planning. . . PSpace-hard

Comments

- PlanGraph $\neq$ StateGraph
  plan $\equiv$ *path* in StateGraph but
  plan $\equiv$ *flow* in PlanGraph

- Like "Traditional TotalOrder Planner":
  considers action at *FIXED* time

  Like "Partial Order Planner"
  generates partially-ordered plans

- *Parallel Plan*: can execute many actions at once
  if no conflicts
  (eg, load all items at once)

- Guaranteed to find **SHORTEST** plan

- $\approx$ *Not* sensitive to given order of goals

Final Comments

- Planning $\equiv$ Searching

  $\Rightarrow$ GraphPlan

  ...a new approach to Planning

- Future work
  - ∗ Learning (from one plan to next)
  - ∗ Two-way search (fact$\mapsto$goal, goal$\mapsto$fact)
  - ∗ beyond "Strips"-like domains

    creating objects, $\forall$, ...
  - ∗ incorporating other types of constraints
  - ∗ Why guarantee SHORTEST path?

- `http://www.cs.cmu.edu/~avrim/graphplan.html`

SatPlan

- Convert plan-situation
  (Operators, Initial/Final Conditions, . . .)
  to SAT

  (Up to fixed length)

- Run WalkSat to find
  satisficing assignment $\equiv$ plan. . .

  . . . iterative deepening

- Plays to *SatPlan*'s strength,
  as $\exists$ satisfying assignment. . .