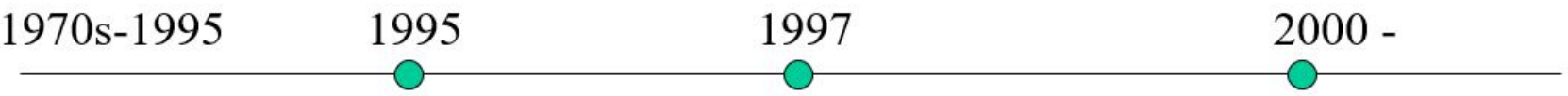
Reviving Partial Order Planning by XuanLong Nguyen & Subbarao Kambhampati

Presented by Fairoz Nower Khan

History of planning



UCPOP
[Penberthy &Weld]
IxTeT
[Ghallab et al]

The whole world believed in POP and was happy to stack 6 blocks!

Advent of CSP style compilation approach:

Graphplan
[Blum & Furst]
SATPLAN
[Kautz & Selman]

Use of reachability analysis and Disjunctive constraints

Domination of heuristic state search approach:

HSP/R [Bonet & Geffner]
UNPOP [McDermott]:
POP is dead!

Importance of good Domain-independent heuristics

Hoffman's FF – a state search planer swept through AIPS-00 competition!

NASA's highly publicized RAX still a POP dinosaur

POP believed to be good framework to handle temporal and resource planning
[Smith et al, 2000]

UCPOP

UNPOP

RePOP

A revival for partial order planning

POP can be made very efficient by exploiting the same ideas that scaled up state search and Graphplan planners

- Effective heuristic search control
- Use of reachability analysis
- Handling of disjunctive constraints

POP review

$P = (A, O, L, OC, UL)$

A: set of action steps in the plan
$S_0, S_1, S_2 \ldots, S_{inf}$

O: set of action ordering $S_i < S_j, \ldots$

L: set of causal links $S_i \xrightarrow{p} S_j$

OC: set of open conditions
(subgoals remain to be satisfied)

UL: set of unsafe links $S_i \xrightarrow{p} S_j$
where p is deleted by some action S_k

<u>Flaw:</u> Open condition OR unsafe link

<u>Solution plan:</u> A partial plan with no remaining flaw

- Every open condition must be satisfied by some action
- No unsafe links should exist (i.e. the plan is consistent)

POP Algorithm

1. Let P be an initial plan
2. Flaw Selection: Choose a flaw f (either open condition or unsafe link)
3. Flaw resolution:

- If f is an open condition,
  choose an action S that achieves f
- If f is an unsafe link,
  choose promotion or demotion
- Update P
- Return NULL if no resolution exist

4. If there is no flaw left, return P, else go to 2.

Main ideas from the paper

1. Ranking partial plans by an effective distance-based heuristic estimator

2. Exploiting reachability analysis by using invariants to discover hidden conflicts in the plan.

3. Resolving threatened links by posting disjunctive ordering constraints into the partial plan to avoid unnecessary and exponential multiplication of failures due to promotion/demotion splitting

1. Ranking partial plans using distance-based Heuristic

1. Ranking Function: $f(P) = g(P) + w\, h(P)$

$g(P)$: number of actions in P

$h(P)$: estimate of number of new actions needed to refine P to become a solution plan

w: increase the greediness of the heuristic search

1. Ranking partial plans using distance-based heuristic

2. Estimating h(P) using Planning Graph

h(P) is estimated by relaxing the negative effects present in the partial plan P

Negative effects of actions are relaxed

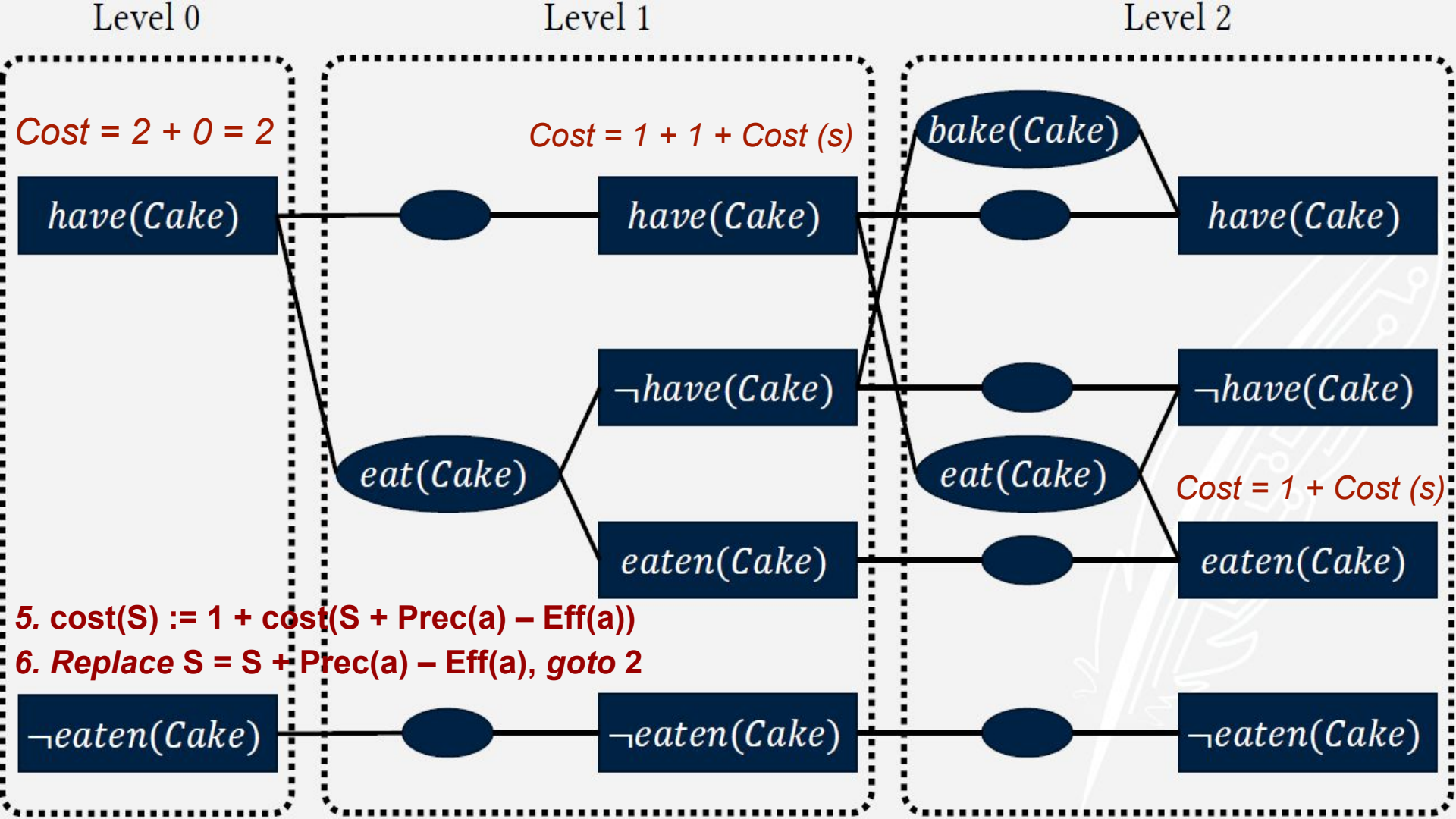
- P has no threatened causal link flaws
- h(P) becomes the number of actions (**cost(S)**) needed to achieve the set of open condition **S** from the initial state

Distance-based heuristic estimate

Estimate cost(S)

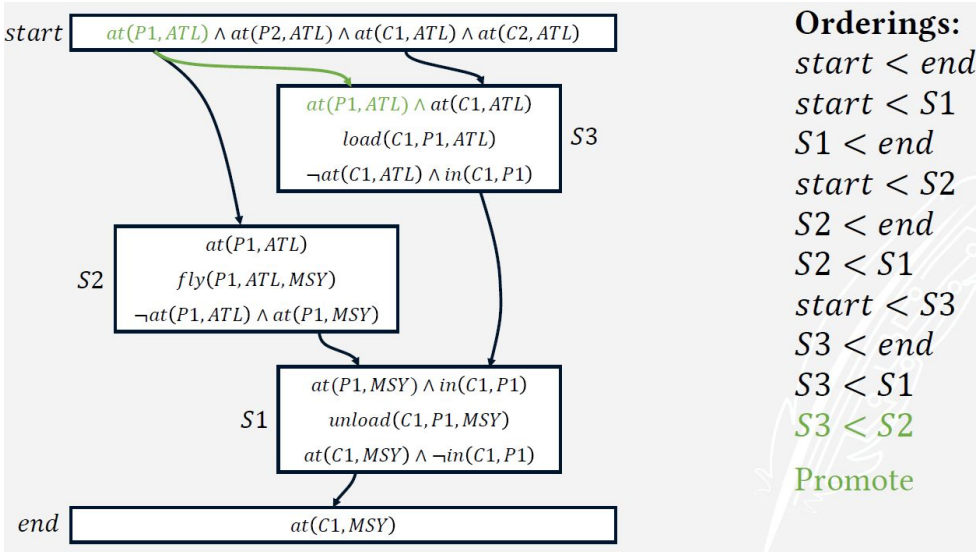
1. Build a planning graph PG from the initial state.
2. Cost(S) := 0 if all subgoals in S are in level 0.
3. Let p be a subgoal in S that appears last in PG.
4. Pick an action a in the graph that first achieves p
5. Update

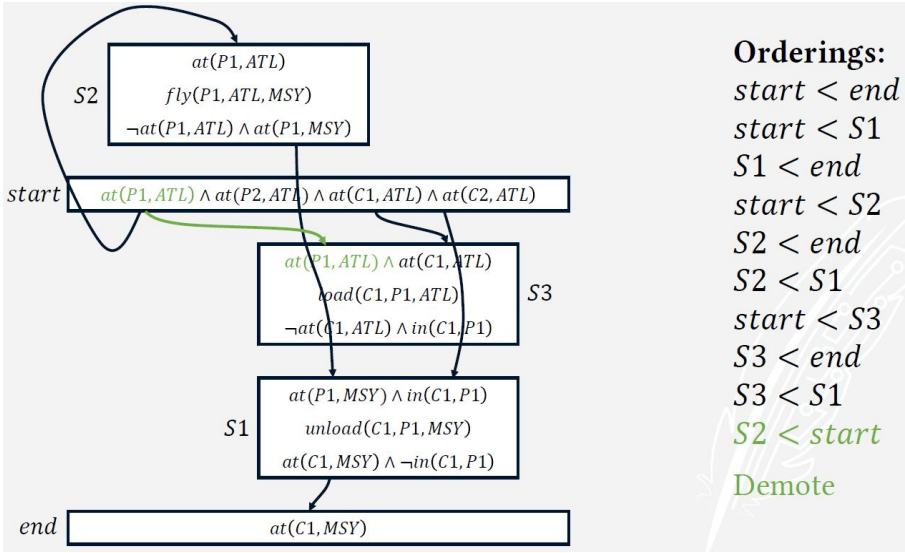
   cost(S) := 1 + cost(S + Prec(a) – Eff(a))
6. Replace S = S + Prec(a) – Eff(a), goto 2


Level 0
Level 1
Level 2
Cost = 2 + 0 = 2
Cost = 1 + 1 + Cost (s)
have(Cake)
have(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
¬have(Cake)
eat(Cake)
eat(Cake)
Cost = 1 + Cost (s)
eaten(Cake)
eaten(Cake)
5. cost(S) := 1 + cost(S + Prec(a) – Eff(a))
6. Replace S = S + Prec(a) – Eff(a), goto 2
¬eaten(Cake)
¬eaten(Cake)
¬eaten(Cake)

2. Handling unsafe link flaws

- Previously, we know that a threatened causal link is resolved by either promotion or demotion
- Splitting the current partial plan into two partial plans





2. Handling unsafe link flaws

- For each unsafe link $S_i \xrightarrow{p} S_j$ threatened by another step S_k:

  Add disjunctive constraint to O: $S_k < S_i \text{ V } S_i < S_j$

- The disjunctive ordering constraint captures both the promotion and demotion possibilities without prematurely splitting a single failing plan unnecessarily multiplied into many descendant plans

- Whenever a new ordering constraint is introduced to O, perform the constraint propagations

2. Handling unsafe link flaws

- $(a_1 \prec a_2) \in \mathcal{O} \wedge (a_2 \prec a_3) \in \mathcal{O} \Rightarrow \mathcal{O} \leftarrow \mathcal{O} \cup (a_1 \prec a_3)$
- $(a_1 \prec a_2) \in \mathcal{O} \wedge (a_2 \prec a_1) \in \mathcal{O} \Rightarrow \text{False}$
- $(a_1 \prec a_2) \in \mathcal{O} \wedge (a_2 \prec a_1 \vee a_3 \prec a_4) \in \mathcal{O} \Rightarrow$
  $\mathcal{O} \leftarrow \mathcal{O} \cup (a_3 \prec a_4)$
  $\mathcal{O} \leftarrow \mathcal{O} - (a_2 \prec a_1 \vee a_3 \prec a_4)$

- If action a1 comes before a2, and a2 comes before a3, then a1 must come before a3.
- If a1 comes before a2 and a2 comes before a1, it's impossible, so it's false.
- If a1 comes before a2 and a2 comes before a1 or a3 comes before a4, then a3 must come before a4

3. Detecting indirect conflicts using reachability analysis

Reachability analysis to detect invariant:

- To detect hidden conflicts in plans, we need to understand what states the plan can reach. This type of information, known as reachability information, has been crucial in improving state space planners. It helps to identify conflicts and inconsistencies within plans.
- Graphplan's planning graph structure is applied to identify mutex constraints. Mutexes are conditions or actions that cannot occur together. At the final level of the planning graph, the mutexes present there are known as state invariants.

3. Detecting indirect conflicts using reachability analysis

Reachability analysis to detect invariant:

- Cutset: Set of literals that must be true at some point during execution of plan. Cutsets are sets of conditions that must be true before and after a specific action is taken in a plan.
- If there's a cutset that contains a mutex, it means the plan is definitely invalid and cannot lead to a successful solution. This allows us to efficiently prune or remove such plans from consideration.

3. Detecting indirect conflicts using reachability analysis

- An action is ak is said to have conflict with a causal link ai $\xrightarrow{p}$ aj :
- if a set of disjunctive ordering constraints (O) is consistent and if either the preconditions (Prec) or effects (Eff) of action ak contain a mutex involving the condition p.

1. *Generalizing unsafe link*: S_k threatens $S_i \xrightarrow{p} S_j$ iff p is mutually exclusive (mutex) with either $Prec(S_k)$ or $Eff(S_k)$

2. *Unsafe link is resolved* by posting disjunctive constraints (as before)
   $S_k < S_i$ V $S_i < S_j$

- Detects indirect conflicts early
- Derives more disjunctive constraints to be propagated

Experiments on RePOP

- Compared RePOP against UCPOP, Graphplan and AltAlt in a number of benchmark domains

  – Time and Solution quality
- RePOP is very good in parallel domains (gripper, logistics, rocket, parallel blocks world)
  - Outperforms Graphplan in many domains
  - Competitive with AltAlt
  - Completely dominates UCPOP
- RePOP still inefficient in serial domains:
  Travel, Grid, 8-puzzle

Experiments on RePOP

RePOP generates partially ordered plans

- *Number of actions:* RePOP typically returns shortest plans
- *Number of time steps (makespan):*
  - Graphplan produces optimal number of time steps
  - RePOP comes close
- *Flexibility*:
  - RePOP typically returns the most flexible plans

Thank You!