RePOP: Reviving Partial Order Planning

XuanLong Nguyen & Subbarao Kambhampati

{xuanlong,rao}@asu.edu

Yochan Group*:*

http://rakaposhi.eas.asu.edu/yochan

In the beginning it was all POP.

Then it was cruelly UnPOPped

The good times return with Re(vived)POP

A recent (turbulent) history of planning

1970s-1995

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

↓

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

UCPOP

1995

Advent of CSP style compilation approach:

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

↓

Use of reachability analysis and Disjunctive constraints

1997

Domination of heuristic state search approach:

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

↓

Importance of good Domain-independent heuristics

UNPOP

2000 -

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]

RePOP

RePOP: A revival for partial order planning

- To show that 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
- RePOP, implemented on top of UCPOP
  - Dramatically better than all known partial-order planners
  - Outperforms Graphplan and competitive with

  state search planners in many (parallel) domains

Partial plan representation

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

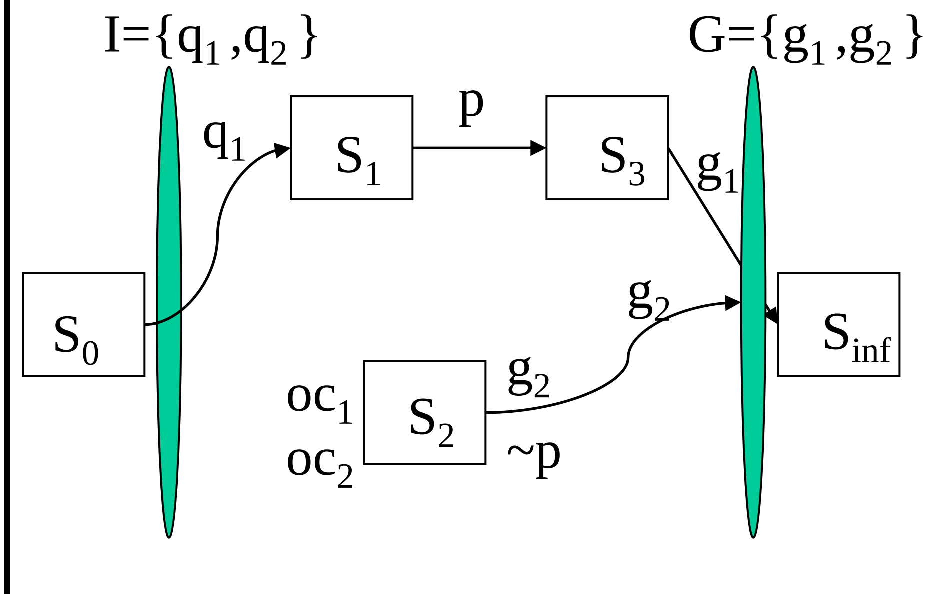
A: set of action steps in the plan $S_0, S_1, S_2 \ldots, S_{inf}$

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



Flaw: Open condition OR unsafe link

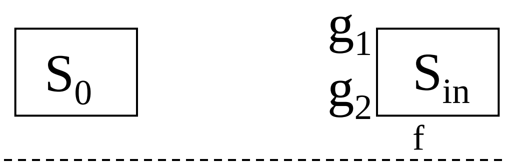
Solution plan: A partial plan with no remaining flaw

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

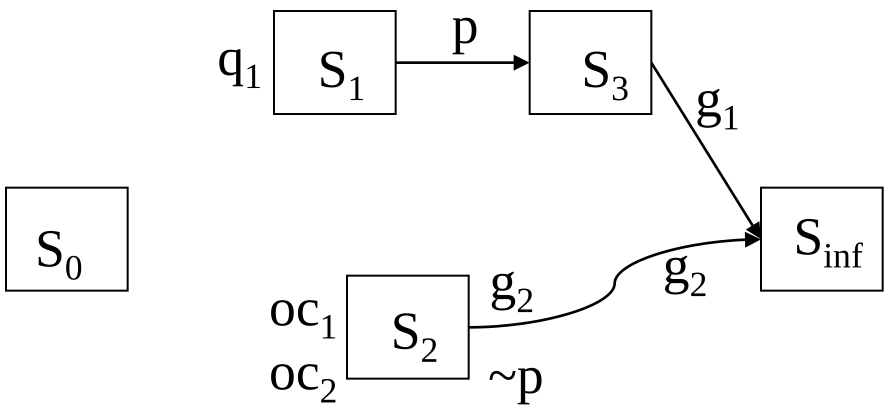
Algorithm

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.

1. Initial plan:



2. Plan refinement (flaw selection and resolution):



Choice points

- Flaw selection *(open condition? unsafe link?)*
- Flaw resolution *(how to select (rank) partial plan?)*
  - Action selection (backtrack point)
  - Unsafe link selection (backtrack point)

Our approach (main ideas)

1. *Ranking partial plans:*
   use an effective distance-based heuristic estimator

.

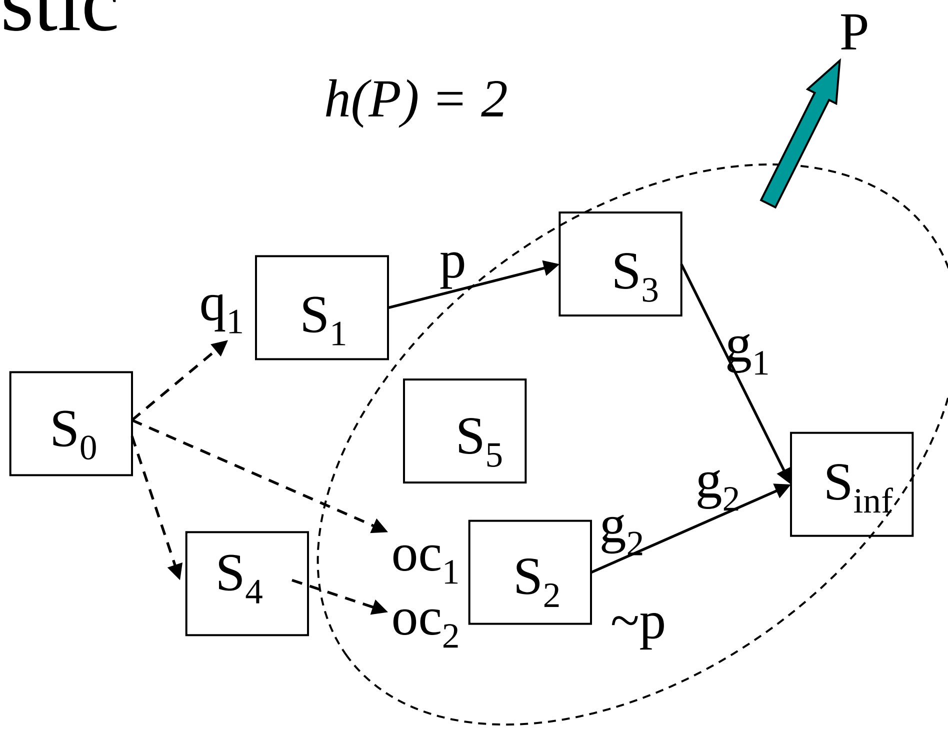
2. *Exploit reachability analysis:*
   use invariants to discover implicit conflicts in the plan.

3. *Unsafe links are resolved by posting disjunctive ordering constraints into the partial plan*:
   avoid unnecessary and exponential multiplication of failures due to promotion/demotion splitting

State-space idea of distance heuristic

CSP ideas of consistency enforcement

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

2. Estimating h(*P*)

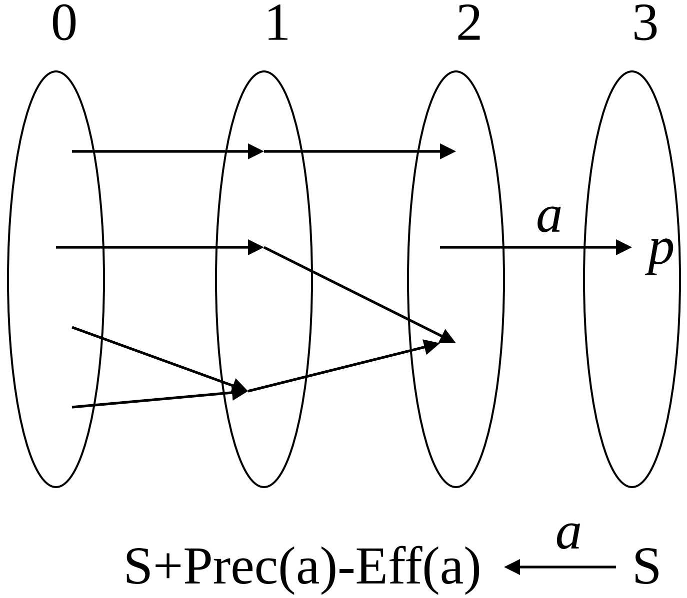
h(P) is estimated by relaxing some constraints present in the partial plan P

Negative effects of actions are relaxed

- *P has no unsafe 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

+ Any state-space heuristic can be adapted
+ Relaxing negative effects makes the estimate inaccurate in serial domains.



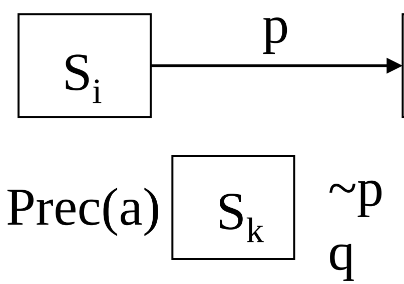
<u>Estimate cost(S)</u>

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

2. Handling unsafe link flaws

1. For each unsafe link $S_i \xrightarrow{p} S_j$
threatened by another step S_k:
Add disjunctive constraint to O

$$S_k < S_i \vee S_j < S_k$$



2. Whenever a new ordering constraint
is introduced to O, perform the constraint propagations:

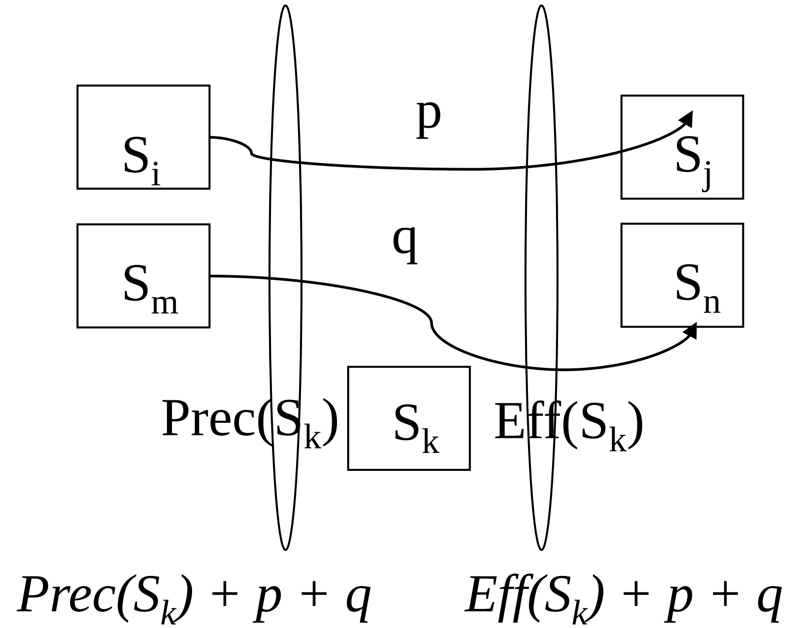
$$S_1 < S_2 \vee S_3 < S_4 \wedge S_4 < S_3 \Rightarrow S_1 < S_2$$
$$S_1 < S_2 \wedge S_2 < S_3 \Rightarrow S_1 < S_3$$
$$S_1 < S_2 \wedge S_2 < S_1 \Rightarrow \text{False}$$

- *Avoid the unnecessary exponential multiplication of failing partial plans*

3. Detecting indirect conflicts using reachability analysis

1. Reachability analysis to detect invariant:
   - *on(a,b) and clear(b)*
   - *How to get state information in a partial plan*

3. Cutset: Set of literals that must be true at some point during execution of plan
   For each action a, $S_i \xrightarrow{p} S_j$



$pre\text{-}C(S_k) = Prec(S_k) \cup \{p \mid S_i \xrightarrow{p} S_j$ is a link and $S_i < S_k < S_j\}$

$post\text{-}C(S_k) = Eff(S_k) \cup \{p \mid$ is a link and $S_i < S_k < S_j\}$

<u>4. If exists a cutset that violates of a variant, the partial plan is invalid and should be pruned</u>

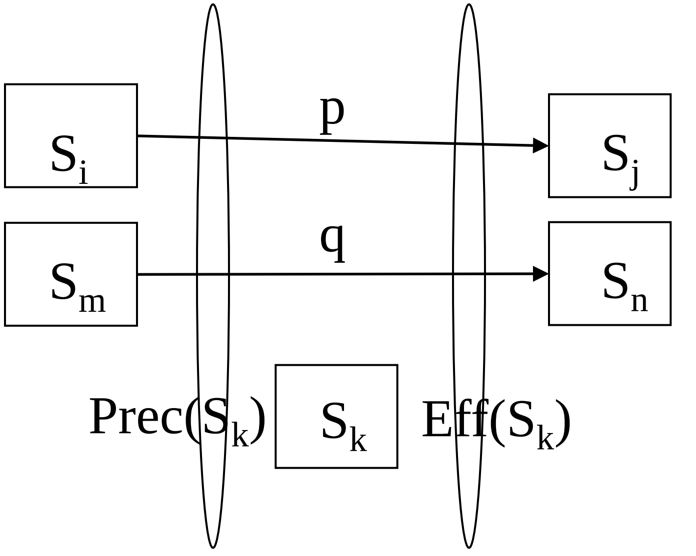
Disadvantage:
- *Inconsistency checking is passive and maybe expensive*

Detecting indirect conflicts using reachability analysis

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)

   $S_k < S_i \vee S_i < S_j$



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

Experiments on RePOP

- RePOP is implemented on top of UCPOP planner using the three presented ideas
  - Written in Lisp, runs on Linux, 500MHz, 250MB
- Compare RePOP against UCPOP, Graphplan and AltAlt in a number of benchmark domains
  - Time
  - Solution quality

Comparing planning time (summary)

Repop vs. UCPOP
Graphplan
AltAlt

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

Comparing planning time (time in seconds)

Repop vs. UCPOP
Graphplan
AltAlt

Problem	UCPOP	RePOP	Graphplan	AltAlt
Gripper-8	-	1.01	66.82	.43
Gripper-10	-	2.72	47min	1.15
Gripper-20	**-**	**81.86**	**-**	**15.42**
Rocket-a	-	8.36	75.12	1.02
Rocket-b	**-**	**8.17**	**77.48**	**1.29**
Logistics-a	-	3.16	306.12	1.59
Logistics-b	-	2.31	262.64	1.18
Logistics-c	-	22.54	-	4.52
Logistics-d	**-**	**91.53**	**-**	**20.62**
Bw-large-a	45.78	(5.23) -	14.67	4.12
Bw-large-b	-	(18.86) -	122.56	14.14
Bw-large-c	**-**	**(137.84) -**	**-**	**116.34**

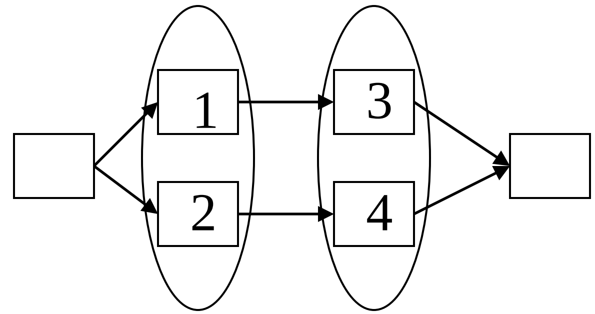
Some solution quality metrics

Repop vs. UCPOP
Graphplan
AltAlt

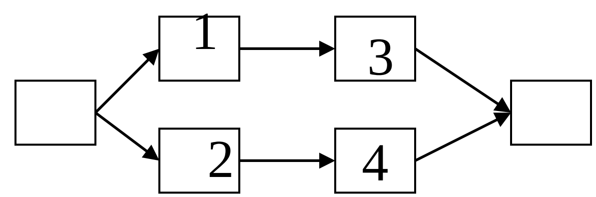
1. Number of actions

2. Makespan:
minimum completion time
(number of time steps)

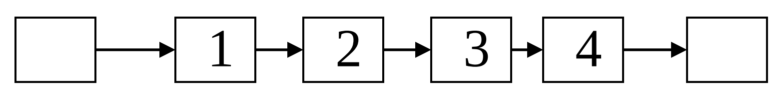
3. Flexibility:
Average number of actions
that do not have ordering
constraints with other actions



Num_act=4
Makespan=2
Flex = 1



Num_act=4
Makespan=2
Flex = 2



Num_act=4
Makespan=4
Flex = 0

Comparing solution quality (summary)

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

Comparing solution quality

	Number of actions/ time steps				Flexibility degree		
Problem	RePOP	Graphplan	AltAlt		RePOP	Graphplan	AltAlt
Gripper-8	21/ 15	23/ 15	21/ 21		.57	.69	0
Gripper-10	27/ 19	29/ 19	27/ 27		.59	.61	0
Gripper-20	**59/ 39**	**-**	**59/ 59**		**.68**	**-**	**0**
Rocket-a	35/ 16	40/ 7	36/ 36		2.46	7.15	0
Rocket-b	**34/15**	**30/ 7**	**34/ 34**		**7.29**	**4.80**	**0**
Logistics-a	52/ 13	80/ 11	64/ 64		20.54	6.58	0
Logistics-b	**42/ 13**	**79/ 13**	**53/ 53**		**20.0**	**5.34**	**0**
Logistics-c	50/ 15	-	70/ 70		16.92	-	0
Logistics-d	**69/ 33**	**-**	**85/ 85**		**22.84**	**-**	**0**
Bw-large-a	(8/5) -	11/ 4	9/ 9		2.75	2.0	0
Bw-large-b	(11/8) -	18/ 5	11/ 11		3.28	2.67	0
Bw-large-c	(17/ 10) -	-	19/ 19		5.06	-	0

Ablation studies

CE: Consistency enforcement techniques (reachability analysis and disjunctive constraint handling

HP: Distance-based heuristic

Problem	UCPOP	+ CE	+ HP	+CE+HP (RePOP)
Gripper-8	*	6557/ 3881	*	1299/ 698
Gripper-10	*	11407/ 6642	*	2215/ 1175
Gripper-12	*	17628/ 10147	*	3380/ 1776
Gripper-20	*****	*****	*****	**11097/ 5675**
Rocket-a	*****	*****	**30110/ 17768**	**7638/ 4261**
Rocket-b	*	*	85316/ 51540	28282/ 16324
Logistics-a	*	*	411/ 191	847/ 436
Logistics-b	*	*	920/ 436	542/ 271
Logistics-c	*	*	4939/ 2468	7424/ 4796
Logistics-d	*****	*****	*****	**16572/ 10512**

Conclusion

- Developed effective techniques for improving partial-order planners:
  - Ranking partial plan heuristics,
  - Disjunctive representation for unsafe links,
  - Use of reachability analysis
- Presented and evaluated RePOP
  - Brings POP to the realm of effective planning algorithms
  - Can now exploit the flexibility of POP without too much efficiency penalty

Future Work

- Extend RePOP to deal with time and resource constraints
- Extend RePOP to deal with partially instantiated actions
- Improve the efficiency of RePOP in serial domains
  - Serial domains inherent weakness of POP?
  - Real-world domains tend to admit partially ordered plans
- Devising effective admissible heuristics for POP