CS5446 AI Planing and Decision Making

Planning Graph

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 1, 2020/21

Outline

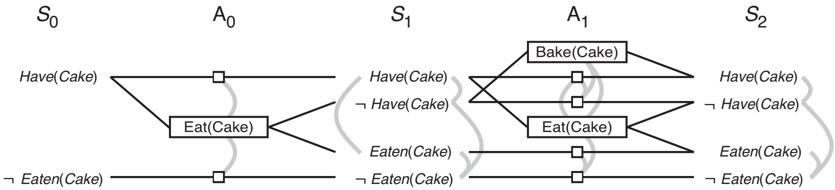
Planning Graphs

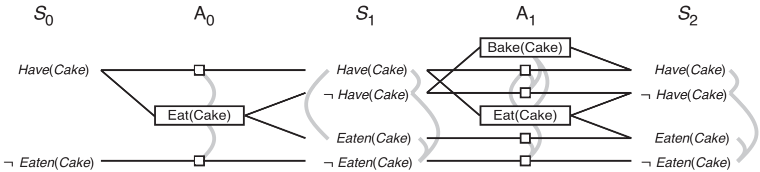
- A **planning graph** is a polynomial sized approximation to the search tree.

- Organized into **levels**: $S_0, A_0, S_1, A_1, \ldots$
  - Roughly, S_i contains all literals that *could* hold at time i and A_i contains all actions that *could* have their preconditions satisfied at time i.
  - Level j at which a literal first appears is a good estimate of how difficult it is to achieve the literal.
- Planning graphs work only on propositional planning problems – need to propositionalize PDDL action schemas.

Have cake and eat it problem and its planning graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake))$

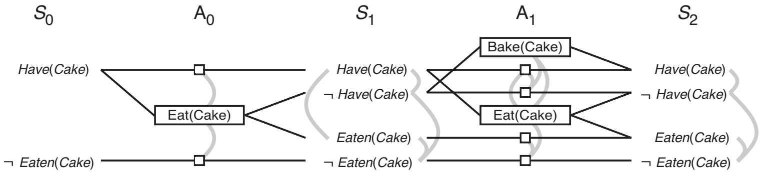




Construct layer by layer, starting from S_0

- Each action at level A_i is connected to its preconditions in S_i and its effects in S_{i+1}.
- A literal appears in S_{i+1} because an action in A_i caused it.

$Init(Have(Cake))$
$Goal(Have(Cake) \land Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \land Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake))$

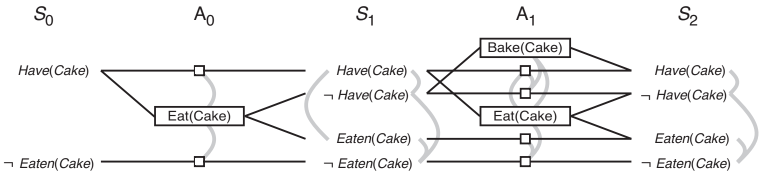


A literal persists if no action negates it – represented as **persistence action** or *no-op*

- For every literal C, we add a persistence action with precond C and effect C.
- Persistence action represented as a box
- Example: Level A_0 in figure, real action $Eat(cake)$ and two persistence action.

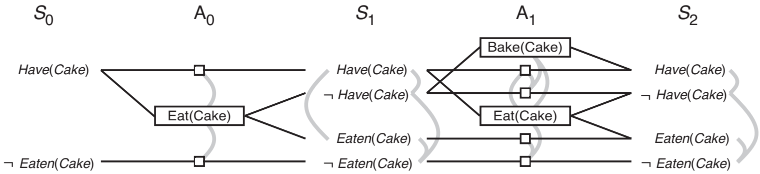
$Init(Have(Cake))$
$Goal(Have(Cake) \land Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg\, Have(Cake) \land Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg\, Have(Cake)$
  EFFECT: $Have(Cake))$

Continue until two consecutive levels identical: **leveled off**.



- Compare to search tree:
  - In search tree, one action at each level – have to branch on the action.
  - In planning graph, all actions that satisfies precond (literals present at time i) run in parallel and produce all possible effects (literals at time $i+1$).
  - First time a goal (or all goals) achieved earlier in planning graph, compared to search tree (optimistic estimate).

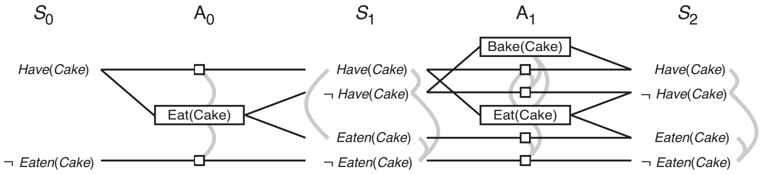
Mutual Exclusion Constraints



- Constraints can be added and propagated in planning graphs to reduce search.
  - Two actions on same level mutually exclusive if no valid plan can contain both.
  - Two propositions on same level mutex if no valid plan can make both true.
- Can put mutual exclusion constraints between every pair of non-persistent actions. For **serial planning graph**.
- Mutex also for *inconsistent effects:* one action negates effect of another. E.g. $Eat(cake)$ and persistence of $Have(cake)$.

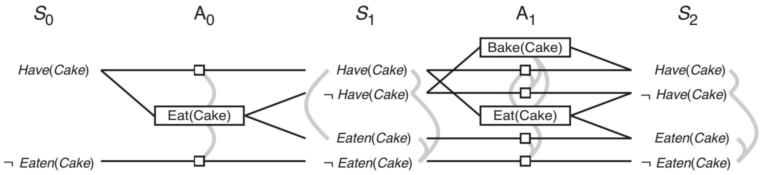
Parallel Plans

- Blum and Furst [1] introduced a natural type of parallel plans.
    - At each time step, have such a set of actions instead of a single action.
    - The set of actions can be carried out in any order or at the same time.
- Planning graph can be used to search for this type of plan, as well as the usual plan (serial planning graph).



Instead of allowing a single action (all pairs of mutex), for parallel plans, only add mutex for the following two conditions.

- *Interference:* Effect of one action is negation of precondition of another. E.g. $Eat(cake)$ interferes with persistence of $Have(cake)$ by negating its precond.
- *Competing needs:* Precond for one action is mutually exclusive with precond of the other. E.g. $Bake(cake)$ and $Eat(cake)$ compete for $Have(cake)$ precond.



- Mutex holds between two literals if one is negation of the other, or possible pairs of actions that could achieve the two literals are mutually exclusive: *inconsistent support*.
  - E.g. $Have(cake)$ and $Eaten(cake)$ are mutex in S_1 – the only way to achieve $Have(cake)$, through persistence action of $Have(cake)$, is mutex with the only way to achieve $Eaten(cake)$, through $Eat(cake)$.
  - But in S_2, $Have(cake)$ and $Eaten(cake)$ are not mutex – new ways of achieving them, e.g. $Bake(cake)$ and persistence of $Eaten(cake)$ are not mutex.
- For a set of goals to be achieved, it is necessary that there is no mutex between any of the goal literals.

For a planning graph with ℓ literals and a actions:

- Each S_i has at most ℓ nodes and ℓ^2 mutex.
- Each A_i has at most $a + \ell$ nodes (including no-ops), $(a + \ell)^2$ mutex, and $2(a\ell + \ell)$ precond and effect links.
- Entire graph with n levels has size $O(n(a + \ell)^2)$. Same complexity for time to build graph.

Planning Graph for Heuristic Estimation

- Estimate the cost of achieving any goal literal with the level at which it first appears: **level cost**.
  - Admissible for single goal.
  - Serial planning graph gives a better estimate.
- For a conjunction of goals
  - **Max-level** heuristic, maximum level cost of any goals, is admissible.
  - **Level sum** heuristic, returns the sum of level costs of goals. Can be inadmissible, but works well in practice when problems are mostly decomposible.
  - **Set level** heuristic, finds the level at which all the conjunctive goals appear without any pair being mutex. Dominates max-level heuristic.

The GraphPlan Algorithm

```
function GRAPHPLAN(problem) returns solution or failure
  graph ← INITIAL-PLANNING-GRAPH(problem)
  goals ← CONJUNCTS(problem.GOAL)
  nogoods ← an empty hash table
  for tl = 0 to ∞ do
     if goals all non-mutex in S_t of graph then
        solution ← EXTRACT-SOLUTION(graph, goals, NUMLEVELS(graph), nogoods)
        if solution ≠ failure then return solution
     if graph and nogoods have both leveled off then return failure
     graph ← EXPAND-GRAPH(graph, problem)
```

- Uses hash table, *nogoods* to store all failed subproblems – when encountered again, just return fail.
- Keep expanding levels until both graph and nogoods have leveled off.
- Can formulate Extract-Solution as a constraint satisfaction problem, or as a backward search problem.

Termination of GraphPlan

Sketch: termination can be shown based on the following properties

- Literals increase monotonically, because of persistence actions.
- Actions increase monotonically, because literals increase monotonically.
- Mutex decreases monotonically, because actions increase monotonically.
- No-goods decreases monotonically, something that is not achievable at current level, cannot be achievable at prev levels, otherwise just use persistence actions.

Proof follows from finiteness of number of actions and literals, and that there cannot be fewer mutex and no-goods than zero.

Reading

- AIAMA Chapter 10.3

Acknowledgement: Figures from AIAMA

References I

[1] Avrim L Blum and Merrick L Furst. "Fast planning through planning graph analysis". In: *Artificial intelligence* 90.1-2 (1997), pp. 281–300.