Search-Based Planning

Sven Koenig, USC

Russell and Norvig, 3rd Edition, Section 10.2.3

These slides are new and can contain mistakes and typos.
Please report them to Sven (skoenig@usc.edu).

1

Non-Interleaved Planning

In the following, a cost-minimal plan is an action sequence with the smallest number of operators

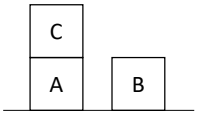
- For each possible ordering of the goal propositions
  - Plan := empty.
  - Consider each goal proposition in the given order
    - Find a cost-minimal plan that achieves the considered goal proposition from the start state and does not delete any of the previously achieved goal propositions.
    - If such a plan does not exist, then start the next iteration of the outer loop.
    - Append the found plan to Plan.
  - Return Plan.
- Return FAILURE.

2

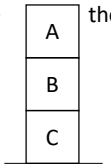
Example: Non-Interleaved Planning

- Blocks world
  This domain consists of a set of cube-shaped blocks sitting on a table. The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block and move it to another position, either on the table or on top of another block. The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it. The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks.

Sussman anomaly



Start state



Goal state

the goal state is specified as follows

On(A,B)
On(B,C)

3

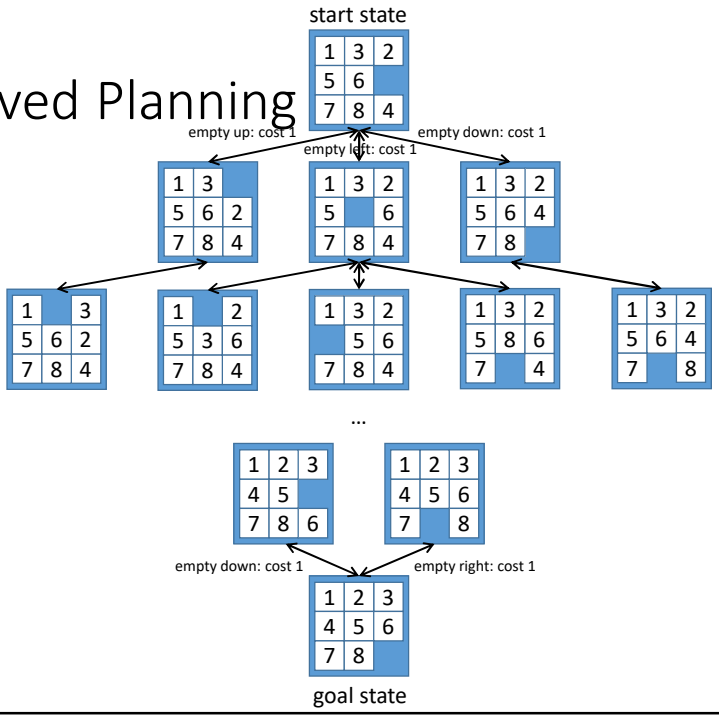
Example: Non-Interleaved Planning

means-end planning:
the plan MoveToTable(C,A), MoveToBlock(A,Table,B)
is a means to the end of achieving On(A,B)

- On(A,B) before On(B,C)
  - Cost-minimal plan for achieving On(A,B): MoveToTable(C,A), MoveToBlock(A,Table,B)
  - Cost-minimal plan for achieving On(B,C) without deleting On(A,B): does not exist
- On(B,C) before On(A,B)
  - Cost-minimal plan for achieving On(B,C): MoveToBlock(B,Table,C)
  - Cost-minimal plan for achieving On(A,B) without deleting On(B,C): does not exist
- FAILURE

4

Interleaved Planning



5

Interleaved Planning (using the Heuristic Search-Based Planner HSP)

- We will use A* to find an action sequence in the state space from the start state to a goal state, resulting in a progression planner (i.e. a planner that searches forward). We could also use A* to implement a regression planner (i.e. a planner that searches backward).
- Often, the branching factor is high and the action sequences are long, meaning that the A* search needs lots of memory and runtime.
- Therefore, one is often content with finding an action sequence of small cost but not necessarily of minimal cost [Suboptimal Search].
- Even then, very informed h-values are necessary to keep the needed amount of memory and runtime sufficiently small [Determining Informed H-Values].

6

STRIPS Example

Start state:

At(Home)
Unequal(Home, SM) Unequal(Home,HW)
Unequal(SM,Home) Unequal(SM, HW)
Unequal(HW,Home) Unequal(HW,SM)
Sells(SM,Milk) Sells(SM,Bananas)
Sells(HSW,Drill)

Goal state:

At(Home)
Have(Milk)
Have(Bananas)
Have(Drill)

Operator schemata:

Go(x,y)
PRECOND: At(x), Unequal(x,y)
EFFECT: At(y), NOT At(x)

Buy(t,x)
PRECOND: Sells(x,t), At(x) ← To keep it simple, we do not model money.
EFFECT: Have(t)

7

Determining Informed H-Values

- Determine informed h-values via problem relaxation
  - Delete some preconditions of some operator schemata
  - Delete some delete effects of some operator schemata
- Here: We delete all delete effects of all operator schemata

8

Determining Informed H-Values

Start state:

At(Home)
Unequal(Home, SM) Unequal(Home,HW)
Unequal(SM,Home) Unequal(SM, HW)
Unequal(HW,Home) Unequal(HW,SM)
Sells(SM,Milk) Sells(SM,Bananas)
Sells(HSW,Drill)

Goal state:

~~At(Home)~~
Have(Milk)
Have(Bananas)
Have(Drill)

Operator schemata:

Go(x,y)
PRECOND: At(x), Unequal(x,y)
EFFECT: At(y), ~~NOT At(x)~~

Buy(t,x)
PRECOND: Sells(x,t), At(x) ← To keep it simple, we do not model money.
EFFECT: Have(t)

9

Determining Informed H-Values

- Determine the ground(ed) predicates (= propositions) that are in all states that can be reached from the start state by operator applications. These ground(ed) predicates are:
  - Unequal(Home, SM) Unequal(Home,HW), Unequal(SM,Home), Unequal(SM, HW), Unequal(HW,Home), Unequal(HW,SM), Sells(SM,Milk), Sells(SM,Bananas), Sells(HSW,Drill)
- Instantiate all operator schemata (with all delete effects removed) and eliminate those operators whose preconditions are not satisfied in any state reachable from the start state.
- Delete the ground(ed) predicates from Step 1 from the start state, the goal state and all remaining operators.

10

Determining Informed H-Values

- The remaining operators are:
  - Go'(Home,SM), Go'(Home,HWS), Go'(SM,Home), Go'(SM,HSW), Go'(HSW,Home), Go'(HSW,SM), Buy'(Milk,SM), Buy'(Bananas,SM), Buy'(Drill,HWS)
- Examples:
  - Go'(Home,SM)
    PRECOND: At(Home)
    EFFECT: At(SM)
  - Buy'(Milk,SM)
    PRECOND: At(SM)
    EFFECT: Have(Milk)

11

Determining Informed H-Values

- gd(start state) = 6
  since the following action sequence solves the planning problem:
  Go(Home,SM), Buy(Milk,SM), Buy(Bananas,SM), Go(SM,HWS), Buy(HWS,Drill), Go(HWS,Home)
- h(start state) = 5
  since the following action sequence solves the relaxed planning problem:
  Go'(Home,SM), Buy'(Milk,SM), Buy'(Bananas,SM), Go'(SM,HWS), Buy'(HWS,Drill)

12

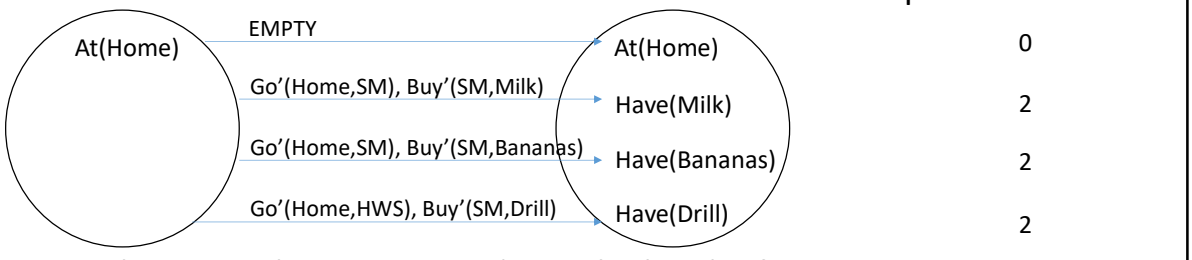
Determining Informed H-Values

- Unfortunately, finding minimum-cost plans for the relaxed planning problems is NP-hard. However, we need to solve a relaxed planning problem each time the A* search needs to calculate an h-value (i.e. many times) and thus we need to solve them fast.
- We therefore simplify the relaxed planning problems further by not taking interactions of the add effects of the operators into account.

13

Determining Informed H-Values

- We now determine the h-value of the start state.



- How do we combine 0, 2, 2 and 2 to the h-value?
- Note that we count Go'(Home,SM) once too many. Thus, we could be inadmissible when adding 0, 2, 2 and 2. We obtain an admissible h-value by taking the maximum of 0, 2, 2 and 2, yielding h(start state) = 2.

14

Determining Informed H-Values

- Formally: Calculate the h-value of state s as h(s) = h_s(goal state), where
  - h_s(set of propositions s') is the estimated number of operator executions to achieve all propositions in s' from s
    - h_s(set of propositions s') := $\max_{p \text{ in } s'} h_s(p)$ [that is what we calculated on the previous slide]
  - h_s(single proposition p) is the estimated number of operator executions to achieve proposition p from s
    - h_s(single proposition p) := 0 if p is in s
    - h_s(single proposition p) := 1 + $\min_{\text{operator o with p as add effect}} h_s$(precondition list of o) if p is not in s

15

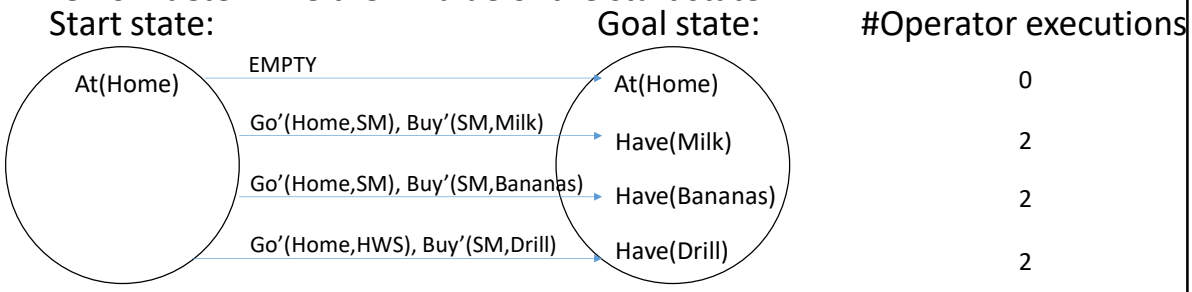
Suboptimal Search

- Give up admissibility of the h-values to result in a suboptimal A* search with less memory and runtime.
- Trick 1: Make the h-values more informed (i.e. get them closer to the goal distances) even if they might become inadmissible.

16

Suboptimal Search

- We now determine the h-value of the start state.



- How do we combine 0, 2, 2 and 2 to the h-value?
- We obtain an h-value by taking the sum of 0, 2, 2 and 2, yielding h(start state) = 6. This is closer to the goal distance of the start state than 2.

17

Suboptimal Search

- Formally: Calculate the h-value of state s as h(s) = h_s(goal state), where
  - h_s(set of propositions s') is the estimated number of operator executions to achieve all propositions in s' from s
    - h_s(set of propositions s') := $\Sigma_{p \text{ in } s'}$ h_s(s') [that is what we calculated on the previous slide]
  - h_s(single proposition p) is the estimated number of operator executions to achieve proposition p from s
    - h_s(single proposition p) := 0 if p is in s
    - h_s(single proposition p) := 1 + $\min_{\text{operator o with p as add effect}}$ h_s(precondition list of o) if p is not in s

18

Suboptimal Search

- Give up admissibility of the h-values to result in a suboptimal A* search with less memory and runtime.
- Trick 2: Make the h-values even larger.
  - Greedy best-first search: f(s) = 0 g(s) + h(s)
  - A*: f(s) = 1 g(s) + h(s)
  - Weighted A*: f(s) = 1/w g(s) + h(s) or, equivalently, f(s) = g(s) + w h(s) for w ≥ 1
    - A* search for w = 1 (typically more node expansions, optimal paths)
    - greedy best-first search for w = ∞ (typically few node expansions, suboptimal paths)
    - the length of the path is at most w times longer than the cost-minimal path
    - the number of node expansions typically decreases as w increases
    - the length of the found path typically increases as w increases

19