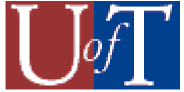


CSC384: Intro to Artificial Intelligence
Planning-II

- Extra Office Hours Tue Nov 28 & Thr Nov 30
- Test4 sample questions posted

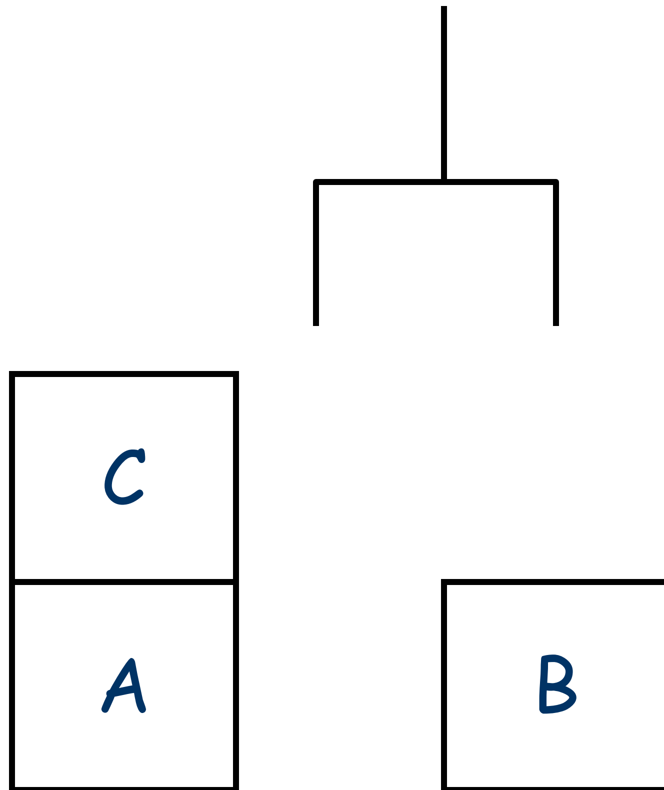
CWA

- "Classical Planning". No incomplete or uncertain knowledge.
- Use the "Closed World Assumption" in our knowledge representation and reasoning.
  - The Knowledge base used to represent a state of the world is a list of positive ground atomic facts.
  - CWA is the assumption that
    a) if a ground atomic fact is not in our list of "known" facts, its negation must be true.
    b) the constants mentioned in KB are all the domain objects.

CWA

- CWA makes our knowledge base much like a database: if employed(John,CIBC) is not in the database, we conclude that $\neg$employed(John, CIBC) is true.

CWA Example

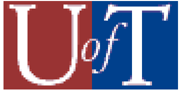


KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

1. clear(c) $\wedge$ clear(b)?
2. $\neg$on(b,c)?
3. on(a,c) $\vee$ on(b,c)?
4. $\exists$X.on(X,c)? (D = {a,b,c})
5. $\forall$X.ontable(X) $\rightarrow$ X = a $\vee$ X = b?

Querying a Closed World KB

- With the CWA, we can evaluate the truth or falsity of arbitrarily complex first-order formulas.
- This process is very similar to query evaluation in databases.
- Just as databases are useful, so are CW KB's.



Querying A CW KB

Query(F, KB) /*return whether or not KB |= F */

if F is atomic
  return($F \in$ KB)

Querying A CW KB

if $F = F_1 \wedge F_2$

return(Query(F_1) && Query(F_2))

if $F = F_1 \vee F_2$

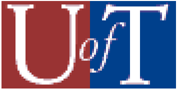
return(Query(F_1) || Query(F_2))

if $F = \neg F_1$

return(! Query(F_1))

if $F = F_1 \rightarrow F_2$

return(!Query(F_1) || Query(F_2))



Querying A CW KB

```
if F = ∃X.F1
  for each constant c ∈ KB
    if (Query(F1{X=c}))
      return(true)
  return(false).

if F = ∀X.F1
  for each constant c ∈ KB
    if (!Query(F1{X=c}))
      return(false)
  return(true).
```

Querying A CW KB

Guarded quantification (for efficiency).

```
if F = ∀X.F1
  for each constant c ∈ KB
    if (!Query(F1{X=c}))
      return(false)
  return(true).
```

E.g., consider checking
$\forall$ X. apple(x) $\rightarrow$ sweet(x)

we already know that the formula is true for all “non-apples”

Querying A CW KB

Guarded quantification (for efficiency).

$\forall X{:}[p(X)]\ F_1 \quad \leftrightarrow \quad \forall X{:}\ p(X) \rightarrow F_1$

```
for each constant c s.t. p(c)
  if (!Query(F1{X=c}))
    return(false)
return(true).
```

$\exists X{:}[p(X)]F_1 \quad \leftrightarrow \quad \exists X{:}\ p(X) \wedge F_1$

```
for each constant c s.t. p(c)
  if (Query(F1{X=c}))
    return(true)
return(false).
```

STRIPS representation.

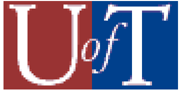
- STRIPS (Stanford Research Institute Problem Solver.) is a way of representing actions.
- Actions are modeled as ways of modifying the world.
  - since the world is represented as a CW-KB, a STRIPS action represents a way of updating the CW-KB.
  - Now actions yield new KB's, describing the new world—the world as it is once the action has been executed.

Sequences of Worlds

- In the situation calculus where in one logical sentence we could refer to two different situations at the same time.
  - $on(a,b,s_0) \wedge \neg on(a,b,s_1)$
- In STRIPS, we would have two separate CW-KB's. One representing the initial state, and another one representing the next state (much like search where each state was represented in a separate data structure).

STRIPS Actions

- STRIPS represents actions using 3 lists.
  1. A list of action preconditions.
  2. A list of action add effects.
  3. A list of action delete effects.
- These lists contain variables, so that we can represent a whole class of actions with one specification.
- Each ground instantiation of the variables yields a specific action.



STRIPS Actions: Example

pickup(X):

Pre: {handempty, clear(X), ontable(X)}

Adds: {holding(X)}

Dels: {handempty, clear(X), ontable(X)}

"pickup(X)" is called a STRIPS operator.

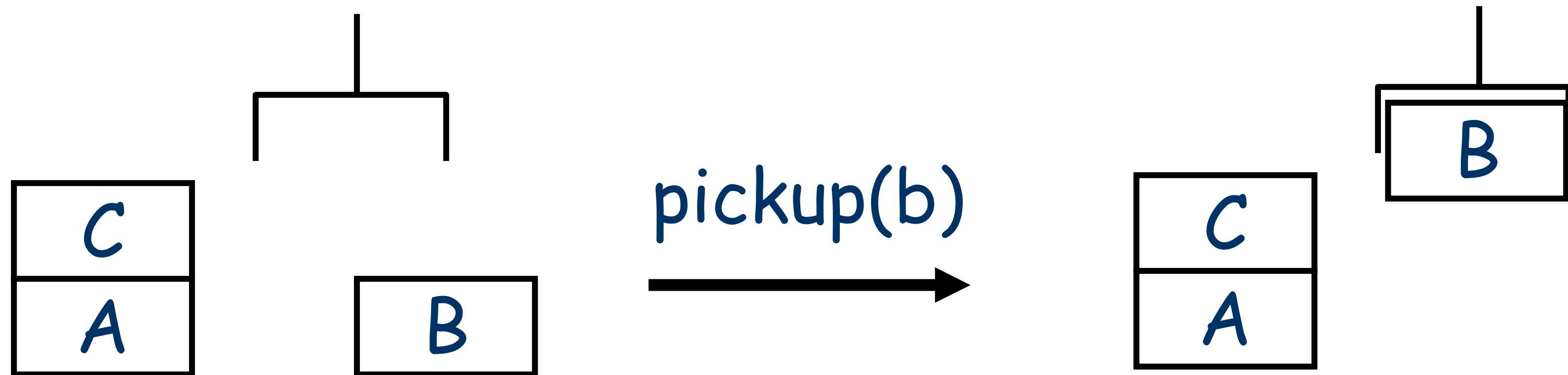
a particular instance e.g.

"pickup(a)" is called an action.

Operation of a STRIPS action.

- For a particular STRIPS action (ground instance) to be applicable to a state (a CW-KB)
  - every fact in its precondition list must be true in KB.
    - This amounts to testing membership since we have only atomic facts in the precondition list.
- If the action is applicable, the new state is generated by
  - removing all facts in Dels from KB, then
  - adding all facts in Adds to KB.

Operation of a Strips Action: Example



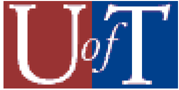
pre = {handmpty,
clear(b),
ontable(b)}

add = {holding(b)}

del = {handmpty,
clear(b),
ontable(b)}

KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a),
ontable(b)}

KB = { holding(b),
clear(c),
on(c,a),
ontable(a)}



STRIPS Blocks World Operators.

- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}
- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}

STRIPS Blocks World Operators.

- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}

STRIPS has no Conditional Effects

- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}

- The table has infinite space, so it is always clear. If we "stack(X,Y)" if Y=Table we cannot delete clear(Table), but if Y is an ordinary block "c" we must delete clear(c).

Conditional Effects

- Since STRIPS has no conditional effects, we must sometimes utilize extra actions: one for each type of condition.
  - We embed the condition in the precondition, and then alter the effects accordingly.

Other Example Domains

- 8 Puzzle as a planning problem
  - A constant representing each position, P1,…,P9

P1	P2	P3
P4	P5	P6
P7	P8	P9

  - A constant for each tile. B,T1, …, T8.

8-Puzzle

- at(T,P) tile T is at position P.

1	2	5
7	8	
6	4	3

at(T1,P1), at(T2,P2), at(T5,P3), …

- adjacent(P1,P2) P1 is next to P2 (i.e., we can slide the blank from P1 to P2 in one move.
  - adjacent(P5,P2), adjacent(P5,P8), …

8-Puzzle

slide(T,X,Y)

Pre: {at(T,X), at(B,Y), adjacent(X,Y)}

Add: {at(B,X), at(T,Y)}

Del: {at(T,X), at(B,Y)}

at(T1,P1), at(T5,P3), at(T8,P5), at(B,P6), …,

1	2	5
7	8	
6	4	3

slide(T8,P5,P6)

at(T1,P1), at(T5,P3), at(B,P5), at(T8,P6), …,

1	2	5
7		8
6	4	3

U of T

Elevator Control

Figure 1: A Miconic-10™ keypad allows passengers to enter their destination before they enter the elevator. A display informs the passenger about the elevator that will offer the most suitable transport.

Elevator Control

- Schindler Lifts.
  - Central panel to enter your elevator request.
  - Your request is scheduled and an elevator is assigned to you.
  - You can't travel with someone going to a secure floor, emergency travel has priority, etc.
- Modeled as a planning problem and fielded in one of Schindler's high end elevators.

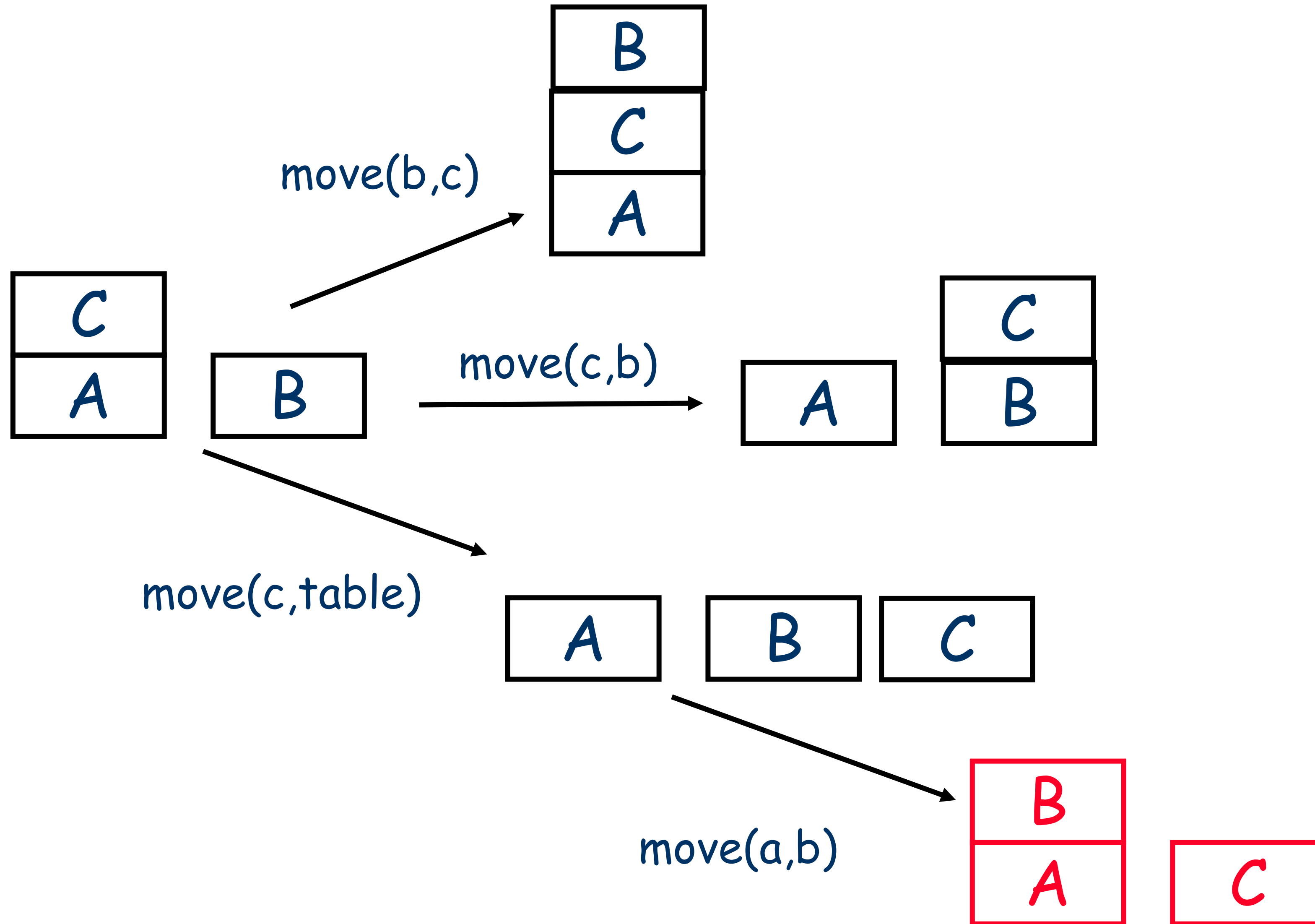
Planning as a Search Problem

- Given a CW-KB representing the initial state, a set of STRIPS or ADL (Action Description Language) operators, and a goal condition we want to achieve (specified either as a conjunction of facts, or as a formula)
  - The planning problem is determine a sequence of actions that when applied to the initial CW-KB yield an updated CW-KB which satisfies the goal.

Planning As Search

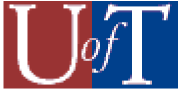
- This can be treated as a search problem.
  - The initial CW-KB is the initial state.
  - The actions are operators mapping a state (a CW-KB) to a new state (an updated CW-KB).
  - The goal is satisfied by any state (CW-KB) that satisfies the goal.

Example.



Problems

- Search tree is generally quite large
  - randomly reconfiguring 9 blocks takes thousands of CPU seconds.
- The representation suggests some structure. Each action only affects a small set of facts, actions depend on each other via their preconditions.
- Planning algorithms are designed to take advantage of the special nature of the representation.

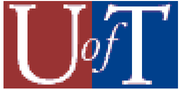


Planning

- We will look at 1 technique
- Relaxed Plan heuristics used with heuristic search.

Reachability Analysis.

- The idea is to consider what happens if we ignore the delete lists of actions.
- This is yields a “relaxed problem” that can produce a useful heuristic estimate.



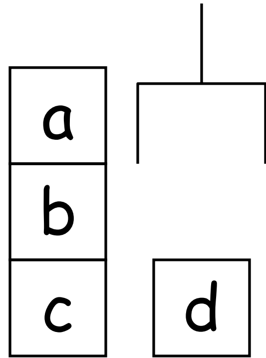
Reachability Analysis

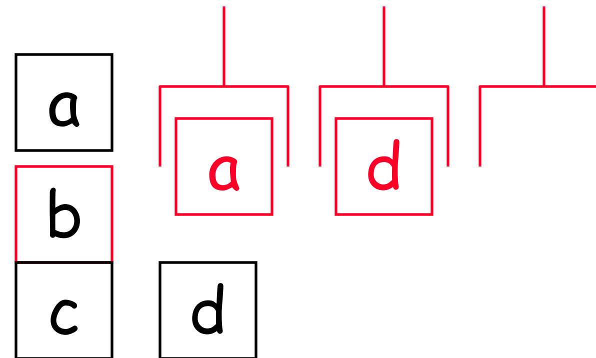
- In the relaxed problem actions add new facts, but never delete facts.
- Then we can do reachability analysis, which is much simpler than searching for a solution.

Reachability

- We start with the initial state S_0.
- We alternate between state and action layers.
- We find all actions whose preconditions are contained in S_0. These actions comprise the first action layer A_0.
- The next state layer consists of all of S_0 as well as the adds of all of the actions in A_0.
- In general
  - A_i is the set of actions whose preconditions are contained in S_i.
  - S_{i+1} is S_i union the add lists of all of the actions in A_i.

Example





on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d)

A_0

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state!

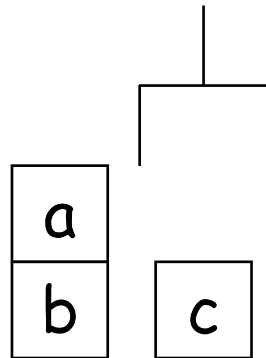
Example

on(a,b),	putdown(a),
on(b,c),	putdown(d),
ontable(c),	stack(a,b),
ontable(d),	stack(a,a),
clear(a),	stack(d,b),
clear(d),	stack(d,a),
handempty,	pickup(d),
holding(a),	…
clear(b),	unstack(b,c)
holding(d)	…
S_1	A_1

Reachabilty

- We continue until the goal G is contained in the state layer, or until the state layer no longer changes.
- Intuitively, the actions at level A_i are the actions that could be executed at the i-th step of some plan, and the facts in level S_i are the facts that could be made true after some i-1 step plan.
- Some of the actions/facts have this property. But not all!

Reachability



on(a,b),
on(b,c),
ontable(c),
ontable(b),
clear(a),
clear(c),
handempty

unstack(a,b)
pickup(c)

on(a,b),
on(b,c),
ontable(c),
ontable(b),
clear(a),
clear(c),
handempty,
holding(a),
clear(b),
holding(c)

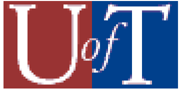
stack(c,b)
...

...
on(c,b),
...

and on(c,b) needs 4 actions

but stack(c,b) cannot be executed after one step

S_0 A_0 S_1 A_1



Heuristics from Reachability Analysis

Grow the levels until the goal is contained in the final state level S[K].

- If the state level stops changing and the goal is not present. The goal is unachievable. (The goal is a set of positive facts, and in STRIPS all preconditions are positive facts).

- Now do the following

Heuristics from Reachability Analysis

CountActions(G,S_K):

/* Compute the number of actions contained in a relaxed plan achieving the goal. */

- Split G into facts in S_{K-1} and elements in S_K only. These sets are the previously achieved and just achieved parts of G.
- Find a minimal set of actions A whose add-effects cover the just achieved part of G. (The set contains no redundant actions, but it might not be the minimum sized set.)
- Replace the just achieved part of G with the preconditions of A, call this updated G, NewG.
- Now return CountAction(NewG,S_{K-1}) + number of actions needed to cover the just achieved part of G.

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$S_2 = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$G = \{f_6, f_5, f_1\}$

We split G into G_P and G_N:

CountActs(G,S_2)

$G_P = \{f_5, f_1\}$ //already in S1

$G_N = \{f_6\}$ //New in S2

$A = \{a_3\}$ //adds all in G_N

//the new goal: $G_P \cup Pre(A)$

$G_1 = \{f_5, f_1, f_2, f_4\}$

Return

1 + CountActs(G_1,S_1)

Example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_1 = \{f_5,f_1,f_2,f_4\}$

We split G1 into G_P and G_N:

CountActs(G_1,S_1)

$G_P = \{f_1,f_2\}$ //already in S0

$G_N = \{f_4,f_5\}$ //New in S1

$A = \{a_1,a_2\}$ //adds all in G_N

//the new goal: $G_P \cup Pre(A)$

$G_2 = \{f_1,f_2\}$

Return

2 + CountActs(G_2,S_0)

= 2 + 0

So, in total CountActs(G,S2)=1+2=3

Using the Heuristic

1. To use CountActions as a heuristic, we build a layered structure from a state S that reaches the goal.
2. Then we CountActions to see how many actions are required in a relaxed plan.
3. We use this count as our heuristic estimate of the distance of S to the goal.
4. This heuristic tends to work better as a best-first search, i.e., when the cost of getting to the current state is ignored.

Admissibility

- An optimal length plan in the relaxed problem (actions have no deletes) will be a lower bound on the optimal length of a plan in the real problem.
- However, CountActions does NOT compute the length of the optimal relaxed plan.
- The choice of which *action set* to use to achieve G_P ("just achieved part of G") is not necessarily optimal.
- In fact it is NP-Hard to compute the optimal length plan even in the relaxed plan space.
- So CountActions will not be admissible.

Empirically

- However, empirically refinements of CountActions performs very well on a number of sample planning domains.