Encoding Delete-Free Planning Tasks in Domain-Independent Dynamic Programming

Maria Desteffani

04. 03. 2025

Motivation

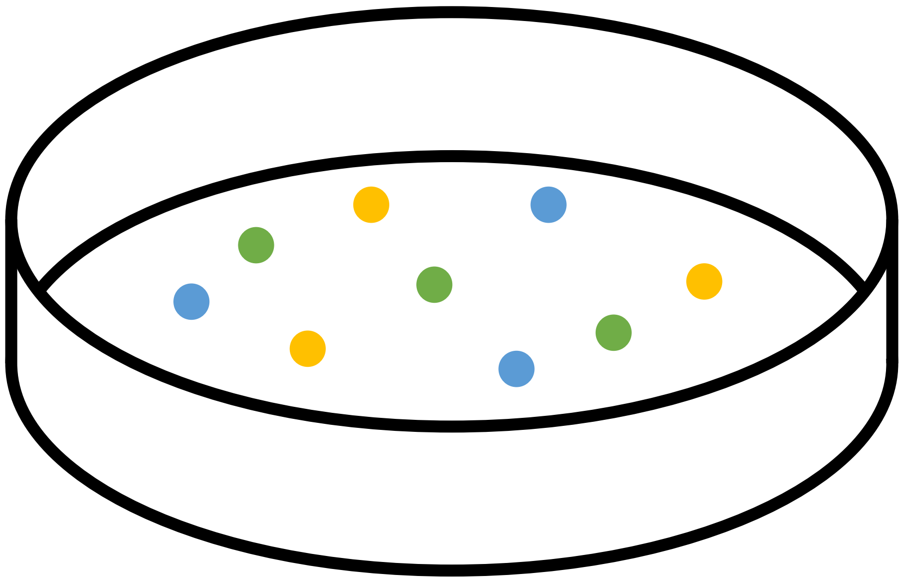
- Bridging Automated Planning and DIDP → out of interest ☺
- DIDP is more powerful than our delete-free planning formalism → possible advantages/disadvantages? Is it worth it?

Delete-Free Planning

- Branch of Automated Planning
- Finding sequences of actions which lead from an initial state to a goal state
- Actions do not remove facts

We use **delete-free STRIPS** as a formalism

The Minimal Seed-Set Problem

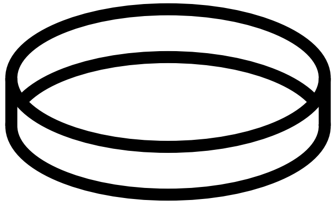


Organism (petri dish) needs to accumulate a certain set of nutrients

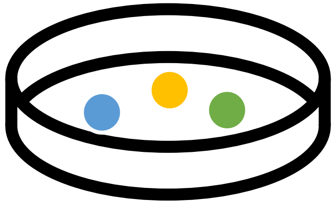
Delete-Free STRIPS

A delete-free STRIPS task is a tuple $\Pi = \langle V,\ I,\ G,\ A \rangle$ containing:

Variables *V*

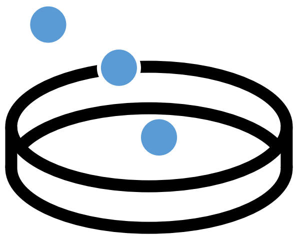


Initial state *I*

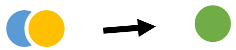


Goal states *G*

Actions *A*



Add nutrients



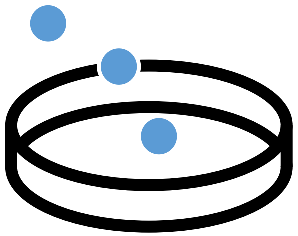
Synthesize nutrients

Delete-Free STRIPS

A delete-free STRIPS task is a tuple $\Pi = \langle V,\ I,\ G,\ A \rangle$ containing:

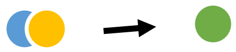
Actions A containing *preconditions*, *add effects*, *cost*

Add nutrients



No preconditions
Costs $$

Synthesize nutrients



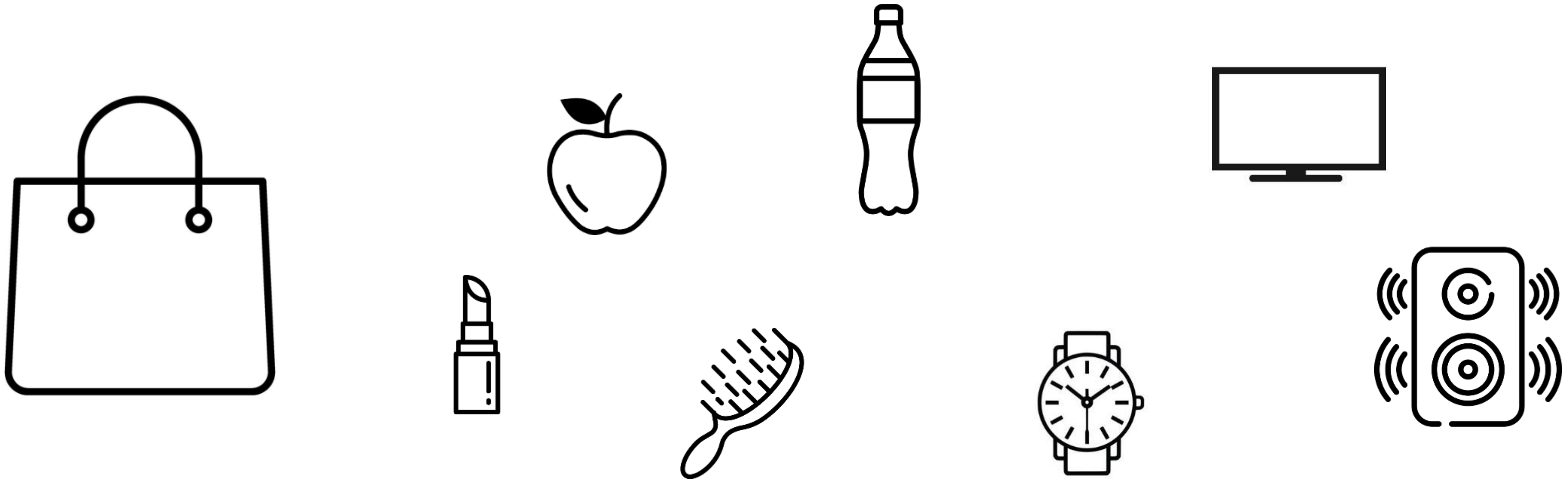
Needs 
Free!

Domain-Independent Dynamic Programming

Dynamic Programming: decompose complex problem into simpler subproblems

e.g.: min. Knapsack problem:

What's the min. weight needed?

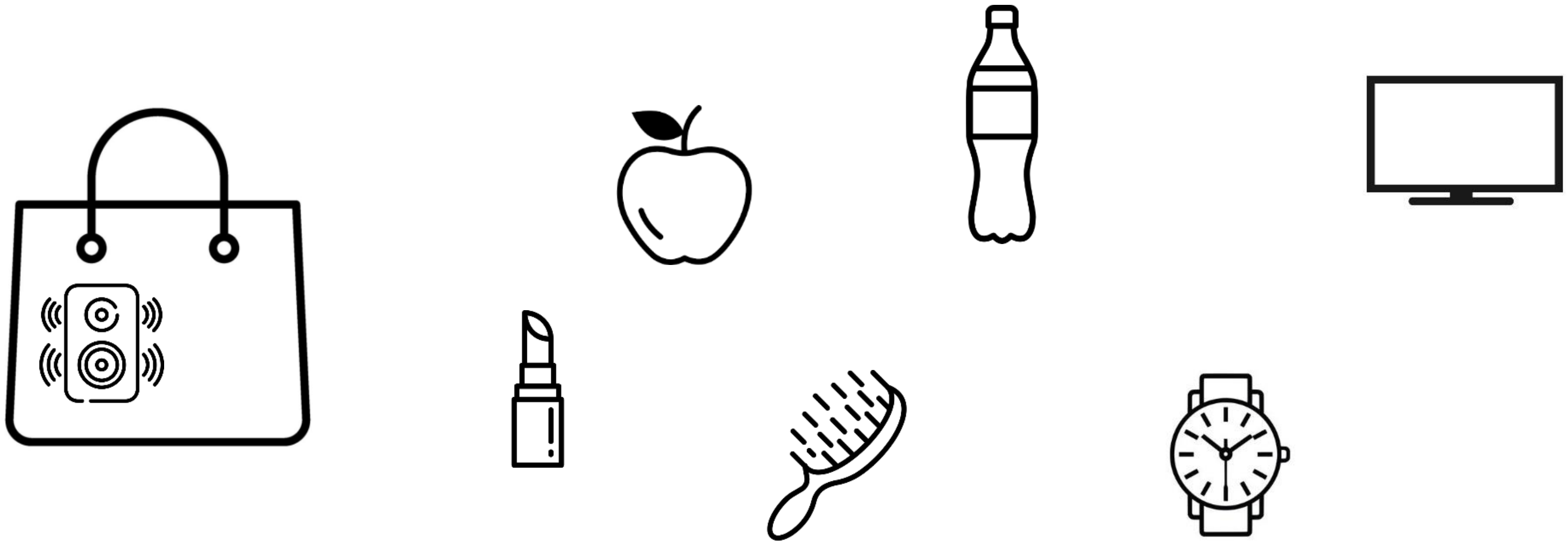


Domain-Independent Dynamic Programming

Dynamic Programming: decompose complex problem into simpler subproblems

e.g.: min. Knapsack problem:

What's the min. weight needed?

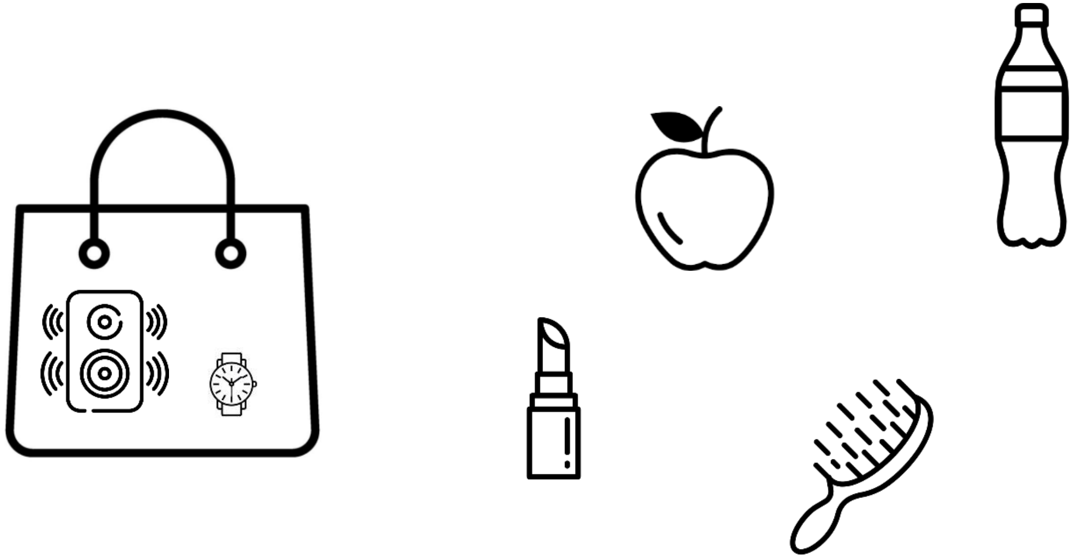


Domain-Independent Dynamic Programming

Dynamic Programming: decompose complex problem into simpler subproblems

e.g.: min. Knapsack problem:

What's the min. weight needed?



Domain-Independent Dynamic Programming (DIDP)

General framework for dynamic programming problems

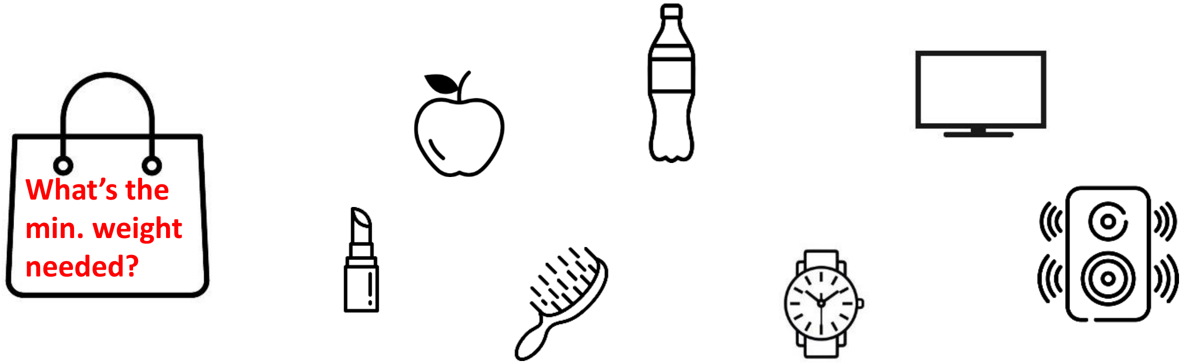
→Allows for a separation of modeling and solving

We use **Dynamic Programming Description Language (DyPDL)** as a modeling formalism

DyPDL

A DyPDL task is a tuple $D = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$ containing:

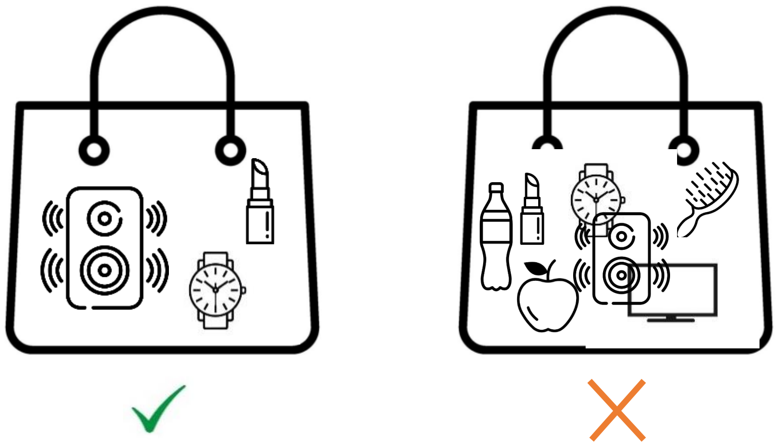
- State variables $\mathcal{V}$ - define states
  - Element variables
  - Set variables
  - Numerical variables
- Target state s^0 - state whose value we want to compute



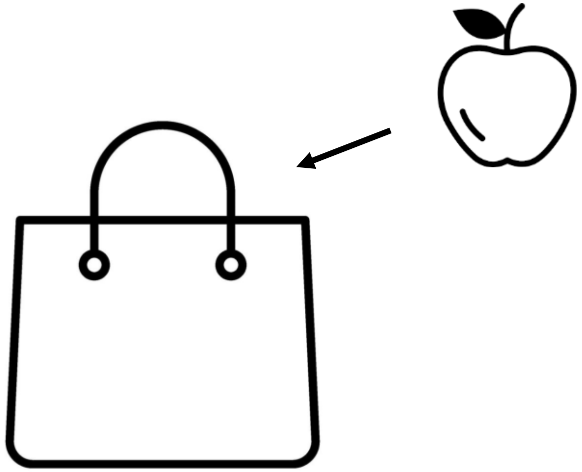
DyPDL

A DyPDL task is a tuple $D = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$ containing:

- Constants $\mathcal{K}$ - state independent variables
- Base cases $\mathcal{B}$ - simplest subproblem, value immediately known



- Transitions $\mathcal{T}$ containing:
  - Preconditions
  - Effects
  - Costs



DyPDL

A DyPDL task is a tuple $D = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$ containing:

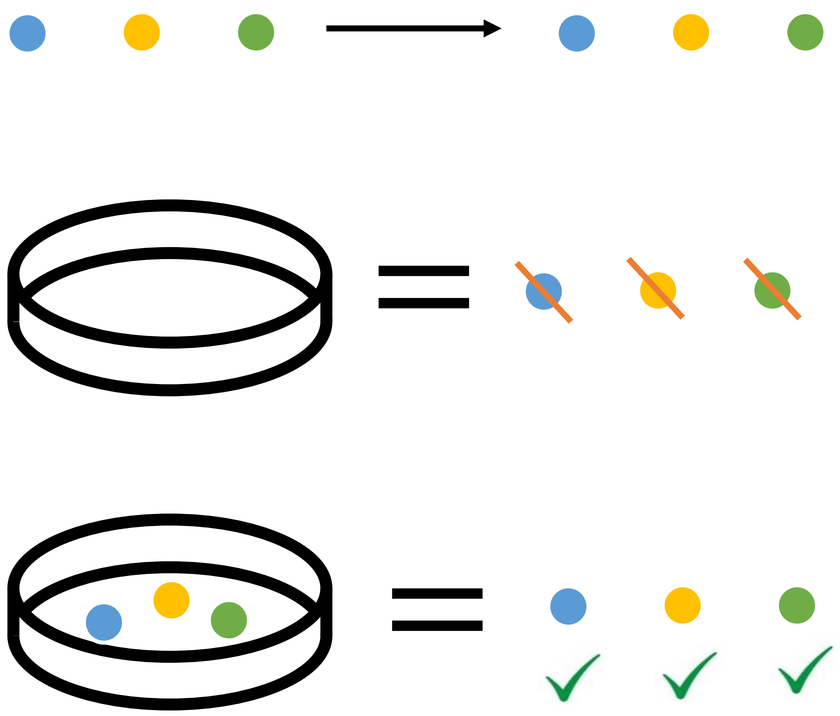
- State Constraint $\mathcal{C}$
- Dual bound h = heuristics

DyPDL – Variable-to-Variable Encoding

$\Pi = \langle V, I, G, A \rangle \quad \rightarrow \quad \mathrm{D}_{\mathrm{VAR}} = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$

We define:

- A DyPDL element variable for each delete-free STRIPS variable
- Target state s^0 expressing the same as I
- Base cases $\mathcal{B}$ expressing the same as G



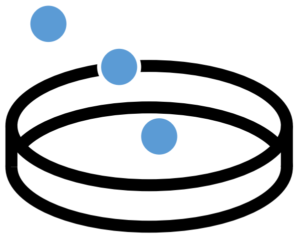
DyPDL – Variable-to-Variable Encoding

$$\Pi = \langle V, I, G, A \rangle \quad \rightarrow \quad \mathrm{D}_{\mathrm{VAR}} = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$$

We define:

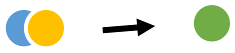
- A transition for each action which:
  - express the same preconditions
  - Have equal cost
  - Effects express the same as add effects

Add nutrients



No preconditions
Costs $$

Synthesize nutrients



Needs 
Free!

DyPDL – Variable-to-Variable Encoding

$\Pi = \langle V, I, G, A \rangle \quad \rightarrow \quad \mathrm{D}_{\mathrm{VA}R} = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$

- Constants $\mathcal{K}$ contain costs of actions
- No dual bound *h* or state constraint $\mathcal{C}$

Motivation for the Second Encoding

Use a set variable instead of many element variables

Possible advantages:

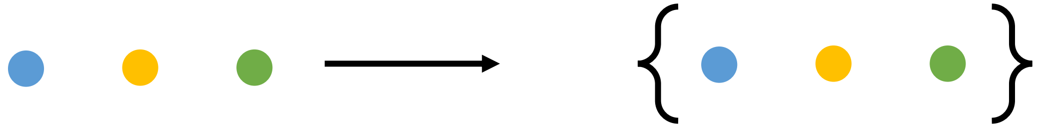
- Internal representation as a bitset possible → less memory used
- Operations & comparisons on bitset → faster

DyPDL – Variable-to-Set Encoding

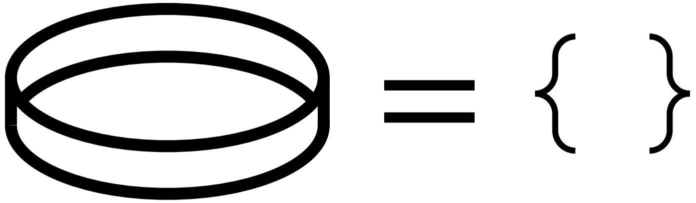
$\Pi = \langle V, I, G, A \rangle \quad \rightarrow \quad \mathrm{D_{SET}} = \langle \mathcal{V},\ s^0,\ \mathcal{K},\ \mathcal{T},\ \mathcal{B},\ \mathcal{C},\ h \rangle$

Difference to the VAR encoding:

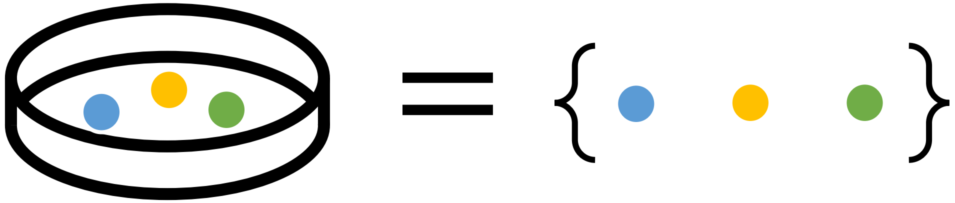
- A DyPDL set variable representing all delete-free STRIPS variables



- Target state s^0 expressing the same as *I*



- Base cases $\mathcal{B}$ expressing the same as *G*



What's the difference between DyPDL and delete-free STRIPS?

Share some inherent similarities, BUT they are two different models:

- States in delete-free STRIPS are sets of variables, in DyPDL they're variable assignments
- Goals are variable sets in delete-free STRIPS, but base cases consist of conditions and values in DyPDL
- Cost gets computed differently
- Delete-free STRIPS cannot support state constraints or built in heuristics (= dual bounds)
- …

Adding a Dual Bound

Heuristics can be added as dual bounds, are a part of the model

→ Maps each state *s* to the estimated cost to the nearest goal

Zero heuristic h^0: $$h^0(s) = 0$$

Mod. Goal count heuristic h^g: $$h^g(s) = \frac{\#\, unsatisfied\ goal\ vars}{\max\{\#\, affected\ vars\}}$$

Force Modification

Actions get forced as soon as their preconditions are satisfied 

We immediately need to do one of the following:

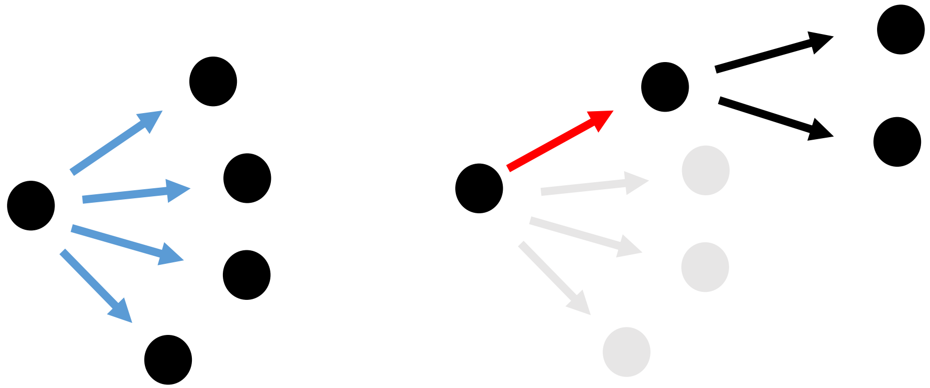
→Apply the action 

→Discard the action 

Actions can NEVER be forced again

Force Modification – possible Advantage

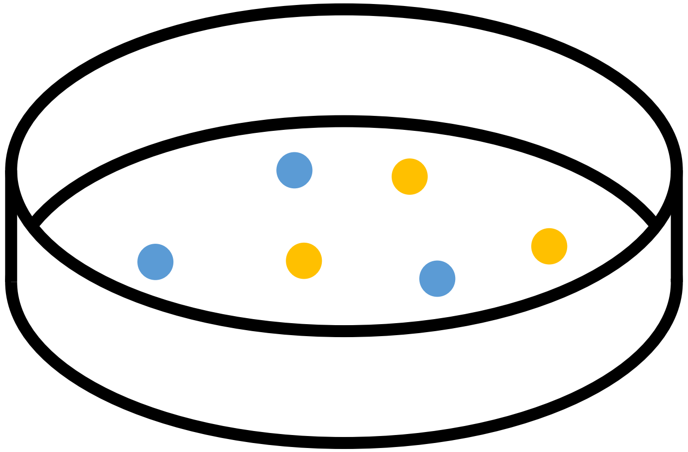
Only a single action can be applied in any state instead of all applicable ones → reduces branching factor



Chain of transitions will get longer though as applying/discarding a single action uses two transitions (forcing + applying/discarding)

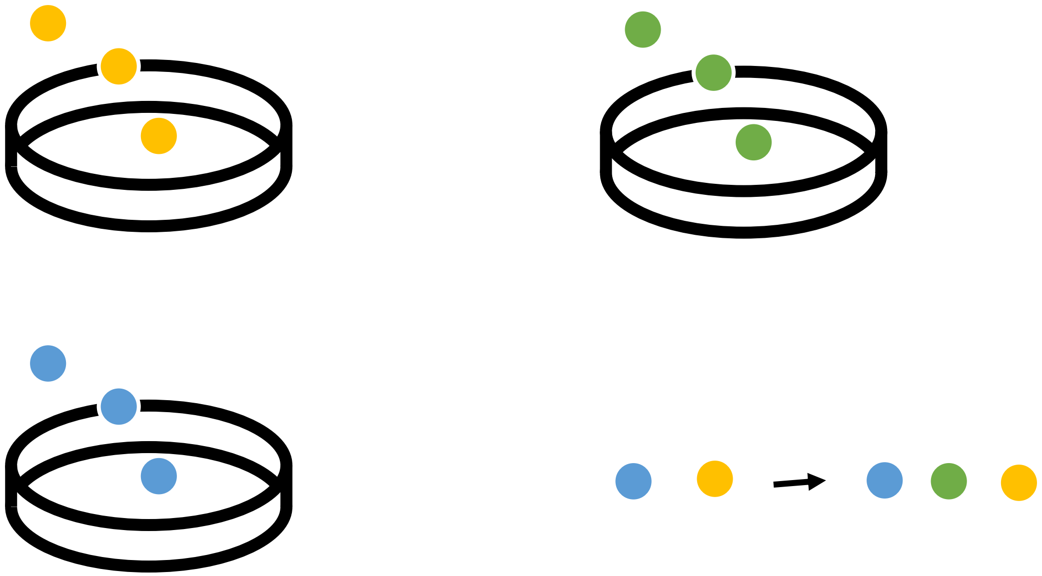
Ignore Modification

We never consider actions which do not add new facts



Ignore

consider



We explicitly enforce this using a precondition instead of relying on the solver!

Experiments – Setup

- Use problems of optimal-strips suite from downward-benchmarks repository
- Use the Fast Downward translator to preprocess and relax problems
- Use the DIDPPy Python interface to generate models, CAASDy solver
- We measure:
  - Finished runs
  - Runs terminated due to memory errors & timeouts
  - Average memory usage
  - Average nodes expanded
  - Time score: value between 0 and 1
- Time limit: 15 min., Memory limit: 3.5GB

Note: We use “VAR (encoding)” and “SET (encoding)” to denote the encodings

Experiment 1 – Baseline comparison

Hypothesis: SET uses less memory, solves more problems, better time score as it can be represented with a bitset internally.

	Finished	Mem. Error	Timeout	Memory	Time score	Nodes exp.
VAR	362	1407	78	43.05 MB	0.752	469.75
SET	346	1143	358	42.78 MB	0.735	469.75

Results: Neither perform great (solved 1/5 of all problems)

- VAR encoding finishes more problems
    - Anomaly in *spider-opt18-strips* domain: SET encoding cannot correctly identify these unsolvable tasks
- Time score better for VAR
- Memory usage similar, expanded nodes exactly the same

Experiment 2 – Heuristics Comparison

Comparing baseline performance with their performance using the zero heuristic and modified goal count heuristic

Hypothesis:

- Both heuristics result in an improvement to all metrics
- Modified goal count heuristics improves performance more
- Zero heuristic should not improve solving much if at all

Experiment 2 – Heuristics Comparison

	Heuristic	Finished
VAR	**None**	362
	Zero	509 *(+147)*
	Goal	**562** *(+200)*
SET	**None**	346
	Zero	513 *(+167)*
	Goal	**566** *(+220)*

Results:

- both significantly improved the number of solved problems

Experiment 2 – Heuristics Comparison

	Heuristic	Time score
VAR	None	0.752
	Zero	0.796 *(+0.044)*
	Goal	0.796 *(+0.044)*
SET	None	0.735
	Zero	0.769 *(+0.034)*
	Goal	**0.770** *(+0.035)*

Results:

- both significantly improved the number of solved problems
- VAR still has better time score, greater improvement

Experiment 2 – Heuristics Comparison

	Heuristic	Nodes exp.
VAR&SET	**None**	469.75
	Zero	127.53 *(-342.22)*
	Goal	**105.71** *(-364.04)*

Results:

- both significantly improved the number of solved problems
- VAR still has better time score, greater improvement
- Nodes expanded still exactly the same, significantly better

Experiment 3 – Excluding Transitions

Comparing the ignore modification to the force modification (using modified goal count heuristic)

Hypothesis:

- Ignore modification should not lead to a great improvement or deterioration
- Force modification could be faster, use less memory due to decreased branching factor
- Force modification could lead to more nodes expanded due to having to use more transitions to apply an action

Experiment 3 – Excluding Transitions

Mod.	Time score
VAR None	0.796
VAR Ignore	0.807
SET None	0.770
SET Ignore	0.782

Results:

- Nothing changed significantly → modification does not really do anything
- slight improvement in time score though

Experiment 3 – Excluding Transitions

Mod.	Finished
VAR None	562
VAR Force	396 *(-166)*
SET None	566
SET Force	383 *(-183)*

Results:

- Significant decrease in solved problems

Experiment 3 – Excluding Transitions

Mod.	Finished	Mem. Error	Timeout
VAR None	562	1266	19
VAR Force	396 *(-166)*	1074	377
SET None	566	995	286
SET Force	383 *(-183)*	738	726

Results:

- Significant decrease in solved problems, timeouts are an issue

Experiment 3 – Excluding Transitions

Mod.	Memory (MB)	Nodes exp.
VAR None	33.64	105.71
VAR Force	55.29	1537.11
SET None	33.43	105.71
SET Force	53.62	1549.47

Results:

- Significant decrease in solved problems, timeouts are an issue
- Nodes expanded exploded, memory used rose
- Interestingly not the same amount of expanded nodes anymore

Experiment 3 – Excluding Transitions

Mod.	Time score
VAR None	0.796
VAR Force	0.764
SET None	0.770
SET Force	0.741

Results:

- Significant decrease in solved problems, timeouts are an issue
- Nodes expanded exploded, memory used rose
- Interestingly not the same amount of expanded nodes anymore
- Time score also got worse

Experiments – General Results

- Baseline encodings are not good
- Dual bounds work great!
- Neither structural modification is worth it

Future Research

- Heuristics have proven to be useful → implementing more powerful heuristics is a promising direction
- Experiment with other kinds of structural modifications, try different directions
- Look into state constraints
- Could expand encodings for STRIPS instead of delete-free STRIPS

Summary

Encodings:

- Variable-to-Variable encoding
- Variable-to-Set encoding

(Potential) Optimizations:

- Zero heuristic & modified goal count heuristic
- Force modification & ignore modification

Experiment Results:

- Baseline models are not great by themselves
- Both heuristics result in a great improvement, mod. goal count heuristic works best
- Ignore modification does not help nor hinder much
- Force modification results in a clear deterioration