Lecture 3: Towards Real-World Planning with the Situation Calculus

Mikhail Soutchanski

(Acknowledgement: follows M.Soutchanski and R.Young's paper
Planning as Theorem Proving with Heuristics
that is available at *https://ceur-ws.org/Vol-3585/*
and an earlier version with a review of history of deductive planning at
https://arxiv.org/abs/2303.13638)

TMU, Department of Computer Science
https://www.cs.torontomu.ca/mes

July 10, 2025

Why Deductive Planning?

We are interested in planning with determinisitc actions. However, there was previous work on reasoning about actions in stochastic domains, when both sensors and effectors are noisy: see References.

Consider theorem proving (deductive) planning. This is a historically first approach to planning: C.Green [IJCAI-1969]. Now, it is based on the Basic Action Theories (BAT) $\mathcal{D} = \Sigma \wedge \mathcal{D}_{ss} \wedge \mathcal{D}_{ap} \wedge \mathcal{D}_{una} \wedge \mathcal{D}_{S_0}$, where the initial theory $\mathcal{D}_{S_0}$ is in special fragment of first order logic (FOL) with standard semantics.

This is a general approach to a broad realistic class of planning problems.

- Can plan without the Domain Closure Assumption (DCA), i.e., there are finitely many $C_1, \ldots, C_n$ s.t. $\forall x(x = C_1 \vee \cdots \vee x = C_n)$. In reality, objects can be unknown, created or destroyed at run-time.
- Can plan without the Closed World Assumption (CWA), i.e., $\mathcal{D}_{S_0}$ is not a conjunction of atomic statements, but it can be an incomplete DB.
- Can plan when actions and fluents have parameters that vary over infinite domains, since search for a plan is done over the situation tree, and situations serve as concise symbolic proxies for infinite models.
- Can plan using action schemas and instantiate actions at run-time (lifted planning), without building explicit state space in advance.

Towards Real-World Planning with SC

Surprisingly, open-world planning without the DCA has not been explored.
Generality vs Tractability dilemma: 5 keys to efficient and realistic planning

1. Consider special (DB-inspired) syntactic form for $\mathcal{D}_{S_0}$ (but no DCA);
2. Goal formulas must have restricted syntax, but can include $\exists$-quantifiers;
3. Preconditions must be efficiently evaluated (over realistic domains);
4. Efficient algorithms to solve the projection problem, e.g., progression;
5. Heuristic search over situations with domain independent heuristics.

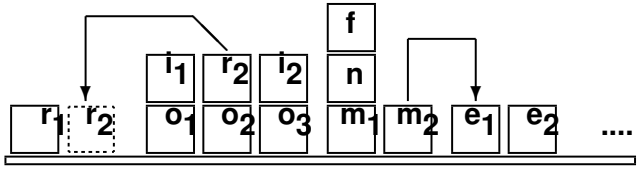
Before we discuss how to do automated planning with situation calculus, we review briefly the landscape of modern automated planning: only the big picture. Our review is simplified for brevity.

In this course, we assume that the attendees know about algorithms for un-informed systematic search (such as Depth-First Search, Breadth-First Search), about informed heuristic search, understand how heuristics can be constructed and compared, know A^* algorithm, its basic variations, and the related concepts. These prerequisites are covered in several undergraduate level text books.

STRIPS vs ADL vs Blocks World in 3D space

STRIPS: actions have *unconditional* effects, and preconditions are conjunctions of fluents, possibly with negation of "$=$".
ADL: SSAs have *context conditions*, i.e., an action has an effect on a fluent only if a context condition holds. See: Bernhard Nebel "On the Compilability and Expressive Power of Propositional Planning Formalisms", JAIR, 2000, v.12, p.271-315

BW. Lecture 1: ADL formulation. Actions: $move(x, y)$, $moveToTable(x)$.



Lecture 2: STRIPS version with a single action $move(x, y, z)$. Another STRIPS version includes the following 3 different actions:

move-b-to-b(x,y,z) move block x from the block y to another block z,
move-t-to-b(x, y) move block x from the table to the block y,
move-b-to-t(x, y) move block x from the block y to the table.

Informally, planning requires finding a (minimum cost) sequence of actions going from the given initial state to a specified goal state.

Variation: blocks are 3D cubes, and all blocks have geometric coordinates in 3D space. An arm can grasp a block and move it in real space and time.

LLM Kaya Stechly, Karthik Valmeekam, Subbarao Kambhampati "Chain of Thoughtlessness: An Analysis of CoT in Planning". *https://arxiv.org/abs/2405.04776*

PDDL vs Situation Calculus: a BW example

PDDL = The Planning Domain Definition Language is a standard input language used at the International Planning Competitions (at *ICAPS*).

PDDL is action-centered . Situation Calculus is fluent-centered.

```
(: action          move-b-to-b
                   : parameters (?bM ?bFrom ?bTo)
: precondition     (and (clear ?bM) (clear ?bTo)
                   (on ?bM ?bFrom)  (not (= ?bM ?bTo)))
: effect           (and (not (clear ?bTo)) (not (on ?bM ?bFrom))
                   (on ?bM ?bTo)  (clear ?bFrom))
 )
```

Precondition axiom

$$\forall x, y, z, s.\ poss(move\text{-}b\text{-}to\text{-}b(x,y,z), s) \leftrightarrow clear(x,s) \wedge clear(z,s) \wedge on(x, y, s) \wedge x \neq z$$

SSA for fluent $on(x, y, s)$

$$\forall x, y, a, s.\ on(x, y, do(a, s)) \leftrightarrow \exists z(a = move\text{-}b\text{-}to\text{-}b(x, z, y)) \vee a = move\text{-}t\text{-}to\text{-}b(x, y) \vee on(x, y, s) \wedge \neg\exists z(a = move\text{-}b\text{-}to\text{-}b(x, y, z)) \wedge \neg(a = move\text{-}b\text{-}to\text{-}t(x, y)).$$

Declarative semantics of STRIPS in the situation calculus: Fangzhen Lin and Ray Reiter “How to progress a database". Artif. Intell. 1997, vol 92, N1-2, pages 131–167.

Hector Geffner and Blai Bonet “A Concise Introduction to Models and Methods for Automated Planning", Morgan & Claypool, 2013.

Landscape of Automated Planning: Part 2

Numerical planning: fluents can take numerical values. Goals include $\leq, \geq$ Parameters vary over a finite domain: DCA+CWA: STRIPS with numbers. In general, the planning problem is undecidable: See [Helmert, AIPS2002].

Temporal planning includes durative actions. They are represented using a pair of instantaneous *begin* and *end* actions. Actions can overlap over time. DCA+CWA. Both fluents and actions have parameters that vary over a finite domain of named objects. Not reducible to classical, if need concurrency.
William Cushing, Subbarao Kambhampati, Mausam, Daniel S. Weld: “When is Temporal Planning Really Temporal?" IJCAI 2007: 1852-1859

Motion Planning: how to move a rigid body (can be reduced to a point in a *configuration space*) from its initial to a goal position while avoiding collisions with obstacles. Known as “Piano Movers problem". Actions have parameters that vary over an infinite space. Key ideas: sampling and lazy search.
Steven LaValle “Planning Algorithms", *https://doi.org/10.1017/CBO9780511546877*

Task and Motion Planning (TAMP): how to integrate numerical/temporal planning with motion planning. Both actions and fluents may have parameters that vary over infinite spaces. See details in F.Lagriffoul, N.T.Dantam, C.R.Garrett, A.Akbari, S.Srivastava, L.Kavraki: “Platform-Independent Benchmarks for Task and Motion Planning". IEEE Robotics Autom. Lett. 3(4): 3765-3772 (2018)
N. Dantam “Task and Motion Planning", Encyclopedia of Robotics, 2021, Springer.

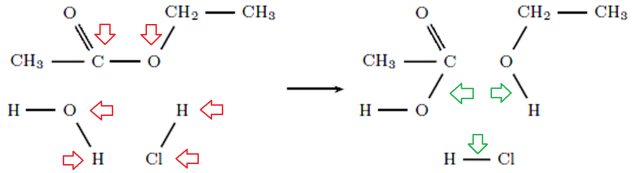
Conformant planning: variant of “classical" planning with DCA but without CWA. No sensing. Search over action sequences in Jörg Hoffmann, R. Brafman (2006): “Conformant planning via heuristic forward search". Artif. Intell. 170(6-7): 507-541

Landscape: from Classical to Renaissance Planning

So-called “classical" planning: DCA + CWA. Goal is a conjunction of fluents. Fluents and actions parameters vary over a finite domain of named objects.

Planners read a domain + an instance descriptions in PDDL, produce a grounded state space, and then search there for a shortest path to a goal.

Potential problem: say an action $A(x_1, x_2, x_3, x_4, x_5)$ has parameters that vary over domain with 100 values. Then, instantiation produces 10^{10} instances. Realistic example: organic chemistry synthesis. Molecule is a graph. Fluents: 4 types of edges, single, double, triple, aromatic. (Re-)actions change edges.



Elias Corey: the 1990 Nobel Prize for "retro-synthetic analysis" (=goal regression).

Tentative solution: lifted planning, i.e., do not ground action schemas before search, but ground them at run-time before you expand a search node.

Rami Matloob and MS: “Exploring Organic Synthesis with State-of-the-Art Planning Techniques". Scheduling and Planning Applications woRKshop (SPARK),ICAPS2016
Arman Masoumi, Megan Antoniazzi, MS: “Modeling Organic Chemistry and Planning Organic Synthesis". GCAI, 2015, p.176-195

The Tree of Situations

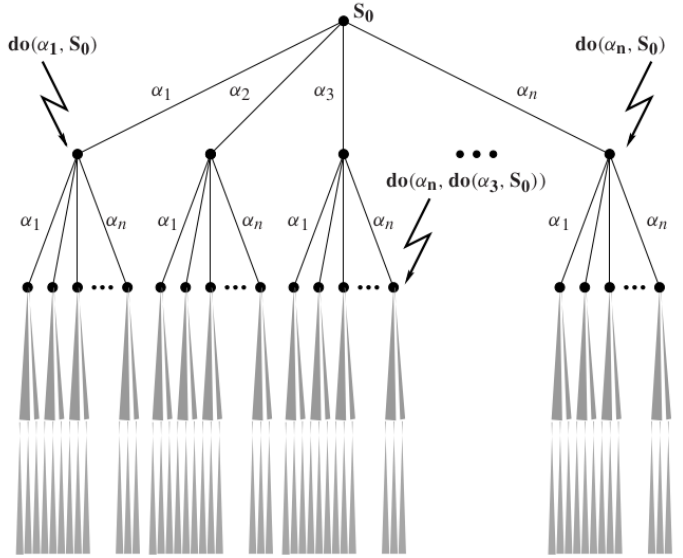


Figure 4.1: The tree of situations for a model with n actions.

Figure: From the Ray Reiter's book Knowledge in Action, Section 4.2.2, Page 51

Bounded Lifted Planning with BATs

Let a BAT has a special initial $\mathcal{D}_{S_0}$, e.g., it can be a consistent set of literals. Let $G(s)$ be a goal formula uniform in s that is an extended conjunctive query, and it has no other free object variables. Let $Length(\sigma)$ be a #actions in σ. For a given $N \geq 0$, the bounded planning problem: find a ground situation σ with $Length(\sigma) \leq N$ such that $\mathcal{D} \models^? executable(\sigma) \wedge G(\sigma)$ (1)

This problem is decidable if $\mathcal{D}_{S_0}$ is a consitent finite set of ground literals. There is an algorithm that is sound and works on some realistic inputs. Note there might be infinitely many infinite size models of a BAT $\mathcal{D}$.

$executable(s) \leftrightarrow ((s = S_0) \vee \exists a \exists s' (s = do(a, s') \wedge poss(a, s') \wedge executable(s')))$

This can be equivalently reformulated for $N > 2$ as check if $\mathcal{D} \models^? G(S_0)$ or if there exists a ground α_1 s.t. $\mathcal{D} \models^? poss(\alpha_1, S_0) \wedge G(do(\alpha_1, S_0))$ or if there exist ground α_1, α_2 such that

$$\mathcal{D} \models^? poss(\alpha_1, S_0) \wedge poss(\alpha_2, do(\alpha_1, S_0)) \wedge G(do([\alpha_1, \alpha_2], S_0))$$

or for some ground σ s.t. $2 < Length(\sigma) \leq N$ $\mathcal{D} \models^? executable(\sigma) \wedge G(\sigma)$

(1) The planner has to search over executable sequences of actions on a situation tree. State space is implicit. (State is a logical model, can be ∞)

(2) To find the next action the planner must check among the actions $A_1(\bar{x}_1), \ldots, A_k(\bar{x}_k)$ for which values of their object arguments these actions are possible in $do([\alpha_1, \cdots, \alpha_i], S_0)$. This can be done at run-time.

(3) An efficient planner needs control that selects the most promising next possible action to execute: need a domain independent heuristic.

A^* Search over Situation Tree to Find a Plan I

Find a ground S with $Length(S) \leq N$ s.t. $\mathcal{D} \models^? executable(S) \wedge G(S)$ (1)

Require: $(\mathcal{D}, G)$ - a BAT $\mathcal{D}$ and a goal formula G
Require: H - Heuristic function
Require: N - Upper-bound on plan length /* e.g., N=100 */
Ensure: ground S that satisfies (1) ▷ Plan is the list of actions in S

1: **procedure** PLAN($\mathcal{D}, G, N, H, S$)
2: $PriorityQueue \leftarrow \varnothing$ ▷ Initialize PQ
3: $S_0.Val \leftarrow (N + 1)$
4: $PriorityQueue.insert(S_0, S_0.Val)$
5: $Init \leftarrow$ InitialState($\mathcal{D}_{S_0}$) ▷ Initialize state
6: **while not** $PriorityQueue.empty()$ **do**
7: $S \leftarrow PriorityQueue.remove()$
8: $Now \leftarrow$ Progress($Init, S$) ▷ Current state Now
9: **if** Satisfy(Now, G) **then**
10: **return** S ▷ Found a plan
11: **end if**
12: $Acts \leftarrow$ FindAllPossibleActions(Now)
13: **if** $Acts == \varnothing$ **then**
14: **continue** ▷ No actions are possible in S

A^* Search over Situation Tree to Find a Plan II

15: **end if**
16: **for** $A_i \in Acts$ **do**
17: $S_n \leftarrow do(A_i, S)$ ▷ S_n is next situation
18: $St \leftarrow$ Progress(Now, A_i) ▷ Next state
19: **if** Length(S_n) $\geq N$ **then**
20: **continue** ▷ S_n exceeds upper bound
21: **else** $d \leftarrow N -$ Length(S_n) ▷ d is depth bound
22: **end if**
23: $S_n.Val \leftarrow$ Length(S_n) $+ H(\mathcal{D}, G, d, S_n, St)$ ▷ f-value of S_n
24: $PriorityQueue.insert(S_n, S_n.Val)$ ▷ Store $S_n.Val$ in PQ
25: **end for**
26: **end while**
27: **return** $False$ ▷ No plan for bound N
28: **end procedure**

$H(\mathcal{D}, G, d, S_n, St)$ is a planning graph inspired heuristic with delete relaxation. Similar but better informed than the well-known FF heuristic: Jörg Hoffmann, Bernhard Nebel: "The FF Planning System: Fast Plan Generation Through Heuristic Search". JAIR, 2001, vol 14, pages 253-302.

GraphPlan Heuristic with Delete Relaxation

Require: $(\mathcal{D}, G)$ - BAT $\mathcal{D}$, a goal formula G
Require: $d \geq 1$ - Look-ahead bound for the heuristic algorithm
Require: S_n, L - The current situation, its length, and St - current state
Ensure: $Score$ - A heuristic estimate for the given situation

1: **procedure** $H(\mathcal{D}, G, d, S_n, St)$
2: $Depth \leftarrow 0$
3: $PG \leftarrow \langle S_n, St \rangle$ ▷ Initialize Planning Graph
4: **while not** Satisfy(St, G) **and** $Depth \leq d$ **do**
5: $\{ActSet\} \leftarrow$ FindAllPossibleActions(St) ▷ Need only relevant acts
6: $NewActs \leftarrow$ Select new acts from $ActSet$ ▷ that add new fluent(s)
7: **if** $NewActs == \emptyset$ **then** ▷ Goal is unreachable
8: **return** $(L + d + 1)$ ▷ Penalty
9: **end if**
10: $St \leftarrow$ ProgressRelaxed($St, NewActs$)
11: ▷ Add all new positive effects $NewEffs$ to the state
12: $NextLayer \leftarrow \langle NewEffs, NewActs, St \rangle$
13: ▷ Record actions added, their effects, the current state
14: $PG.extend(NextLayer)$
15: $Depth \leftarrow Depth + 1$
16: **end while**
17: $Goal \leftarrow$ Convert G into a set of literals
18: **if** $Depth > d$ **then return** $(L + d)$ ▷ Penalty
19: **else return** $Reachability(\mathcal{D}, Goal, PG)$
20: **end if**
21: **end procedure**

Reachability Score for a Set of Goal Literals I

Require: $(\mathcal{D}, G)$ - A BAT $\mathcal{D}$ and a set G of goal literals
Require: PG - A planning graph, initialized to $\langle S_n, St\rangle$
Ensure: V - A heuristic estimate for achieving G

1: **procedure** *Reachability*($\mathcal{D}$, G, PG)
2: **if** $PG == \langle S_n, St\rangle$ **then return** 0
3: **else** $\langle Effs, Acts, St\rangle \leftarrow PG.removeOuterLayer$
4: **end if**
5: $CurrGoals \leftarrow G \cap Effs$ ▷ The set of achieved goals
6: $NewGoals \leftarrow \emptyset$ ▷ To collect preconditions
7: $BestSupport \leftarrow \emptyset$ ▷ Easiest causes for CurrGoals
8: **for** $g \in CurrGoals$ **do**
9: $Relev \leftarrow$ {actions from $Acts$ with g as add effect}
10: **for** $a \in Relev$ **do**
11: $a.Pre \leftarrow$ {the set of preconditions of a}
12: $a.Estimate \leftarrow Reachability(\mathcal{D}, Pre, PG)$ ▷ recursive call
13: **end for**
14: $BestAct \leftarrow \mathrm{ArgMin}\{a.Estimate$ over $Relev\}$ ▷ different from FF
15: ▷ Find the easiest action from $Relev$ with minimum estimate:
16: ▷ See a comment below how this is different from FF

Reachability Score for a Set of Goal Literals II

17: $NewGoals \leftarrow NewGoals \cup BestAct.Pre$
18: $BestSupport \leftarrow BestSupport \cup BestAct$
19: **end for**
20: $RemainGoals \leftarrow G - CurrGoals$
21: $NextGoals \leftarrow RemainGoals \cup NewGoals$
22: $C_1 \leftarrow \mathrm{Count}(BestSupport)$ ▷ i.e. # of best actions
23: $C_2 \leftarrow Reachability(\mathcal{D}, NextGoals, PG)$
24: **return** $C_1 + C_2$
25: **end procedure**

Our implementation of the heuristic is more informative than the FF heuristic, because the difficulty of an action A is recursively estimated by the number of previous easiest actions that achieve its preconditions, rather than the sum of the earliest fact layer indices where each individual precondition appears first $\Sigma_{p \in pre(A)} min\{i | p$ is a member of the fact layer at step $i\}$, as in Section 4.2.2 and Figure 2 (p.264) of Hoffman and Nebel's paper about FF (JAIR,2001). The latter sum over-counts the difficulty of an action.

There are several other well-known generic heuristics that are based on different ideas. More empirical assessment is required to compare them.

TPLH: Theorem Proving Lifted Heuristic planner

We have developed a preliminary version of TPLH in PROLOG. It does a simple progression, but one can use regression as well.

- This implementation uses DCA and CWA (built in Prolog), but these are not restrictions to our TPLH approach.
- A* search is done over *a situation tree*, but not the state space
- Uses a domain-independent heuristic based on the Planning Graph (PG) data structure:
  - based on delete relaxation (fluents never become false)
  - inspired by the Fast Forward (FF) heuristic
  - not admissible but very informative: better than FF
  - PG is fully grounded at run time (lifted version is future work)
- we have developed extensions to the base-line TPLH: filter visited states, Greedy Best-First Search (GBFS) instead of A*, 2-queue extension (another queue with "useful" actions).

It is important to filter out repeatedly visited states: two different situations can lead to the same state. For example, think about different order of actions moving the same blocks to the table. We keep in memory a hash table of visited situations, but not states.

Experimental Comparison of FD and BFWS with TPLH

We present an experimental comparison of the baseline version of TPLH with the recent version of FastDownward (FD) planner [Helmert, Univ. of Basel] and Best First Width Search (BFWS) planner [Lipovetzky and Geffner].

Tests were run separately using the TPLH, FD, and BFWS planners. TPLH and FD used the A* algorithm to prioritize shorter plan lengths, whereas BFWS used a default greedy search algorithm based on a width heuristic. Both TPLH and FD did eager search with FF heuristic.

All testing was done on a desktop with an Intel(R) Core(TM) i7-3770 CPU running at 3.40GHz. Tests measured total time spent, plan length, and number of states (situations) visited. Use IPC scores for comparison.

Each participating planner p gets a score S_i^p per planning task i "*expressed as* $S_i^p = C_i^*/C_i^p$*, where* C_i^p *is the total cost of the best solution found by planner* p *for instance* i*, and* C_i^* *is the lowest total cost found so far by any planner, that is,* $C_i^* = min_p\{C_i^p\}$" [LopezCelorrioOlayaAI2015]. Since unsolved problems are scored as 0, *coverage* is taken into account by the score.

Since TPLH assigns cost 1 to each action, plan length measures plan cost. Situation is deemed visited when TPLH evaluates whether it is a goal state.

The TPLH planner has been loaded, compiled and run within **ECLiPSe Constraint Logic Programming System, Version 7.0 #63** (x86_64_linux), released on April 24, 2022. In comparison, the FD and BFWS were compiled into executable files.

Experimental Assessment Across 9 Benchmarks

Testing was done over randomly generated problems for 8 different popular domains: **Barman** (BR), **BlocksWorld** (BW), **ChildSnack** (CS), **Depot** (D), **FreeCell** (FC), **Grippers** (GR), **Logistics** (L), and **Miconic** (M). In addition, testing was also done on 10 pre-existing instances of the **PipesWorld** (PW) domain. Domains are in STRIPS with negated equalities and object typing.

Roughly 100 problems with varying numbers (with $\sim$10 BW or up to 20-40) of objects were generated for each of the specified domains, using publicly available PDDL generators. All PDDL domains and generated instances files were automatically translated from PDDL to PROLOG using our program.

The TPLH planner was run over every problem using a 15min time-out limit, and a 512M MB stack size limit, i.e., much less than typical 6 GB memory.

The TPLH planner was given the upper bound $N=100$ for all planning instances that usually had short solutions, e.g., $\leq$25-30 steps.

Before TPLH could be tested on a domain, the domain file was converted online from PDDL to a BAT implemented in PROLOG, and initial state hash tables were built for each individual problem.

In terms of CPU time, as expected, TPLH was much slower than FD and BFWS. TPLH timed out on several instances, but both FD and BFWS solved all the instances within allocated time and memory. Note that the number of objects in the generated instances was relatively small.

IPC Scores (Situations/States Visited)

- A*-U: uses the A* algorithm and does not filter duplicate states
- A*-1: identical to A*-U, but filters duplicate states
- A*-2: alternates between two priority queues when selecting from the frontier
- G-1 and G-2: equivalent to A*-1 and A*-2, but use greedy search strategy

Domain	A*-U	A*-1	A*-2	G-1	G-2	FD	BFWS
BR (100)	49.97	56.40	60.02	68.76	99.37	3.96	8.87
BW (95)	67.42	67.63	62.29	82.13	78.61	7.42	13.13
CS (100)	51.10	51.72	51.53	60.86	100	5.00	32.19
D (76)	56.01	56.08	52.82	64.29	70.44	5.04	24.49
FC (95)	94.92	94.92	83.06	94.92	90.47	9.51	21.70
GR (98)	42.73	43.89	36.53	95.10	89.50	2.81	19.92
L (119)	63.22	66.72	59.64	95.79	107.20	8.56	23.73
M (93)	75.21	76.42	32.72	87.93	71.36	7.70	11.03
PW (6)	1.68	1.76	1.71	5.95	5.27	0.58	1.61
Total (782)	502.27	515.54	440.33	655.73	712.22	50.58	156.68

State-of-the-art: FD = FastDownward, BFWS = Best-First Width Search.

BR=Barman, BW=BlocksWorld, CS=ChildSnack, D=Depot, FC=FreeCell, GR=Gripper, L=Logistics, M=Miconic, PW=PowerWorld.

The best score is 1 (per instance): higher IPC score = better performance. Red: two best scores. Our TPLH planner is better in terms of this metric

IPC Scores (Plan Length)

- A*-U{nfiltered}: uses the A* algorithm and **no** filtering of duplicate states
- A*-1: identical to A*-U, but filters duplicate states, 1 queue
- A*-2: alternates between two priority queues when selecting from the frontier
- G-1 and G-2: equivalent to A*-1 and A*-2, but use greedy search strategy

Domain	A*-U	A*-1	A*-2	G-1	G-2	FD	BFWS
BR (100)	100	100	100	75.31	88.96	100	90.28
BW (95)	89.63	90.14	89.61	73.75	76.41	90.55	57.24
CS (100)	100	100	80	86.75	87.88	100	95.78
D (76)	75.83	75.71	75.94	65.11	67.57	75.77	74.21
FC (95)	93.89	93.89	94.00	93.89	93.53	93.98	92.24
GR (98)	97.31	97.31	97.77	87.88	88.90	97.44	77.48
L (119)	119	119	119	100.75	98.35	119	103.87
M (93)	93	93	93	67.52	69.63	93	80.31
PW (6)	5.81	5.89	5.92	5.01	5.33	6	5.55
Total (782)	774.46	774.86	755.24	655.97	676.58	775.76	676.97

Score for a participating planner p on instance i: $S_i^p = min_p\{C_i^p\}/C_i^p$, where C_i^p is the total cost of a plan for instance i. The best (maximal) score is 1 (per instance): higher IPC score = better performance. Red: two best scores.
Both FD and BFWS are the best state-of-the-art planners. BFWS does greedy search, and for this reason its plans are longer than for FD or TPLH. FD uses an original version of FF heuristic.

Summary and Future Work

Novelty and Significance

- To the best of our knowledge, TPLH is the 1st deductive lifted planner that does search over situations using a domain independent heuristic
- TPLH is competitive with FD and BFWS in terms of plan quality (length), while it explores far fewer states.

Future Work

- efficient evaluation of preconditions: find possible actions faster
- a better implementation without DCA and without CWA, using progression for proper initial theories
- experimental evaluation on the new domains where the objects can be created/destroyed at run time
- implementing other domain independent heuristics
- experimental comparisons with other lifted planners.

References: Part 1

Vaishak Belle, Hector J. Levesque: “Reasoning about discrete and continuous noisy sensors and effectors in dynamical systems". Artif. Intell. 2018, vol. 262, pages 189-221.

Elias James Corey “The logic of chemical synthesis: multistep synthesis of complex carbogenic molecules", Nobel Lecture on December 8, 1990.

William Cushing, Subbarao Kambhampati, Mausam, Daniel S. Weld: “When is Temporal Planning Really Temporal?" IJCAI 2007: 1852-1859

N. Dantam “Task and Motion Planning", Encyclopedia of Robotics, 2021.

Hector Geffner, Blai Bonet “A Concise Introduction to Models and Methods for Automated Planning", 2013. Series Title: Synthesis Lectures on Artificial Intelligence and Machine Learning. https://doi.org/10.1007/978-3-031-01564-9

Alfonso Emilio Gerevini, Francesco Percassi, Enrico Scala: “An Effective Polynomial Technique for Compiling Conditional Effects Away". AAAI 2024: 20104-20112 (from ADL to STRIPS)

C. Cordell Green: “Application of Theorem Proving to Problem Solving". IJCAI 1969: pages 219-240.

Malte Helmert: “Decidability and Undecidability Results for Planning with Numerical State Variables". AIPS 2002: pages 44-53

Jörg Hoffmann, Ronen Brafman: “Conformant planning via heuristic forward search". Artif. Intell. 2006, 170(6-7): pages 507-541.

References: Part 2

L Kavraki, P Svestka, JC Latombe, MH Overmars: “Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation, 1996, vol 12 (4), 566-580.

F.Lagriffoul, N.T.Dantam, C.R.Garrett, A.Akbari, S.Srivastava, L.Kavraki: “Platform-Independent Benchmarks for Task and Motion Planning". IEEE Robotics Autom. Lett. 3(4): 3765-3772 (2018)

Jean-Claude Latombe: “Robot Motion Planning", Kluwer Academic, 1996.

Steven LaValle “Planning Algorithms", Cambridge University Press, 2006, available online at *https://doi.org/10.1017/CBO9780511546877*

Fangzhen Lin and Ray Reiter “How to progress a database". Artif. Intell. 1997, vol 92, N1-2, pages 131–167 (Logical semantics for STRIPS).

Arman Masoumi, Megan Antoniazzi, Mikhail Soutchanski: “Modeling Organic Chemistry and Planning Organic Synthesis". The 1st GCAI 2015, EPiC Series in Computing, vol 36, pages 176-195.

Rami Matloob and Mikhail Soutchanski: “Exploring Organic Synthesis with State-of-the-Art Planning Techniques". Scheduling and Planning Applications woRKshop (SPARK) at ICAPS-2016.

Bernhard Nebel “On the Compilability and Expressive Power of Propositional Planning Formalisms", JAIR, 2000, vol.12, p.271-315

Andreas Orthey, Constantinos Chamzas and Lydia E. Kavraki: “Sampling-Based Motion Planning: A Comparative Review", Annu. Rev. Control. Robotics Auton. Syst., 2024, vol 7(1), pages 285-310