Finding multiple plans for classical planning problems

Shirin Sohrabi and Michael Katz and Junkyu Lee
IBM Research AI, USA

David Speck
University of Basel, Switzerland

Based on these papers

- A Novel Iterative Approach to Top-k Planning, Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. ICAPS 2018.
- Symbolic Top-k Planning, Speck, D.; Mattmüller, R.; and Nebel, B. AAAI 2020.
- Top-Quality Planning: Finding Practically Useful Sets of Best Plans, Katz, M.; Sohrabi, S.; and Udrea, O. AAAI 2020.
- Reshaping Diverse Planning, Katz, M.; and Sohrabi, S. AAAI 2020.
- Bounding Quality in Diverse Planning, Katz, M.; Sohrabi, S.; and Udrea, O. AAAI 2022.
- Loopless Top-K Planning, von Tschammer, J.; Mattmüller, R.; and Speck, D. AAAI 2022.
- Who Needs These Operators Anyway: Top Quality Planning with Operator Subset Criteria, Katz, M.; and Sohrabi, S. AAAI 2022.
- On K* Search for Top-k Planning, Lee, J.; Katz, M.; and Sohrabi, S. SoCS 2023.
- K* and Partial Order Reduction for Top-quality Planning, Katz, M.; and Lee, J. SoCS 2023.
- K* Search Over Orbit Space for Top-k Planning, Katz, M.; and Lee, J. IJCAI 2023.
- Some Orders Are Important: Partially Preserving Orders in Top-Quality Planning, Katz, M.; Lee, J.; Kang, J.; and Sohrabi, S. SoCS 2024.
- Unifying and Certifying Top-Quality Planning, Katz, M.; Lee, J.; and Sohrabi, S. ICAPS 2024.

Outline

1. Introduction and motivation
2. Planning problems
   1. Diverse planning
   2. Top-k and top-quality planning
3. Planning methods
   1. ForbidIterative
   2. K*
   3. SymK
4. Hands-on session

What is AI Planning

Task of finding a procedural course of action for a declaratively described system to reach its goals while optimizing overall performance measures.

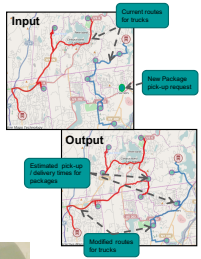
Basic planning problem includes:

- initial states of the world
- desired goals
- a set of possible actions

Synthesize a plan that is guaranteed to generate a state which contains the desired goals.

https://www.odtap.com/2018/10

AI Planning is Everywhere



Source: https://www.livemint.com/Opinion/tijzm8fIw2RY98Jvdm8LzJ/Opinion–Cyber-security-a-complex-behaviour-problem.html
Source: https://www.rigzone.com/news/schlumberger_in_400mm_deal_to_sell_drilling_assets-15-may-2019-158842-article/

AI Planning is Everywhere

Yann LeCun ∞
@ylecun

Please ignore the deluge of complete nonsense about Q*.
One of the main challenges to improve LLM reliability is to replace Auto-Regressive token prediction with planning.

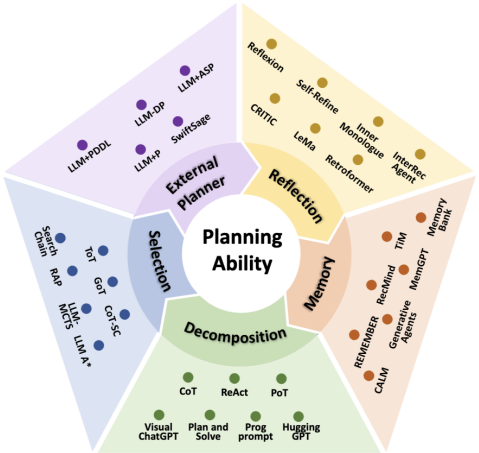
Pretty much every top lab (FAIR, DeepMind, OpenAI etc) is working on that and some have already published ideas and results.

It is likely that Q* is OpenAI attempts at planning. They pretty much hired Noam Brown (of Libratus/poker and Cicero/Diplomacy fame) to work on that.

[Note: I've been advocating for deep learning architecture capable of planning since 2016].

2:02 PM · Nov 24, 2023 · **1.4M** Views

782 Reposts **126** Quotes **6,122** Likes **1,889** Bookmarks

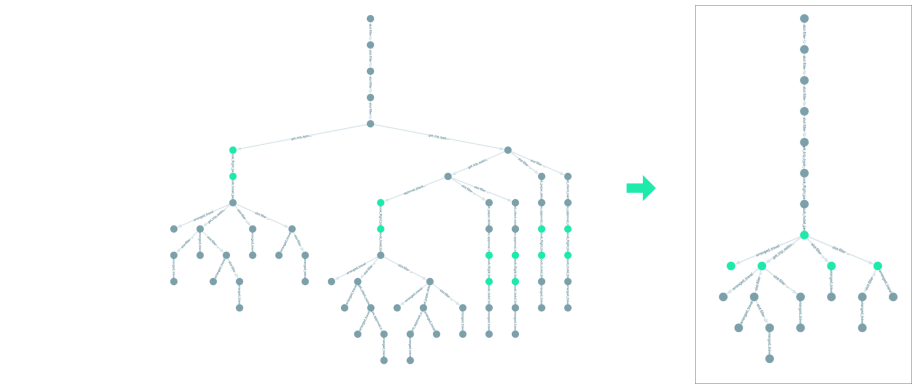


Understanding the planning of LLM agents: A survey, Huang et. al, 2024

AI Planning Applications

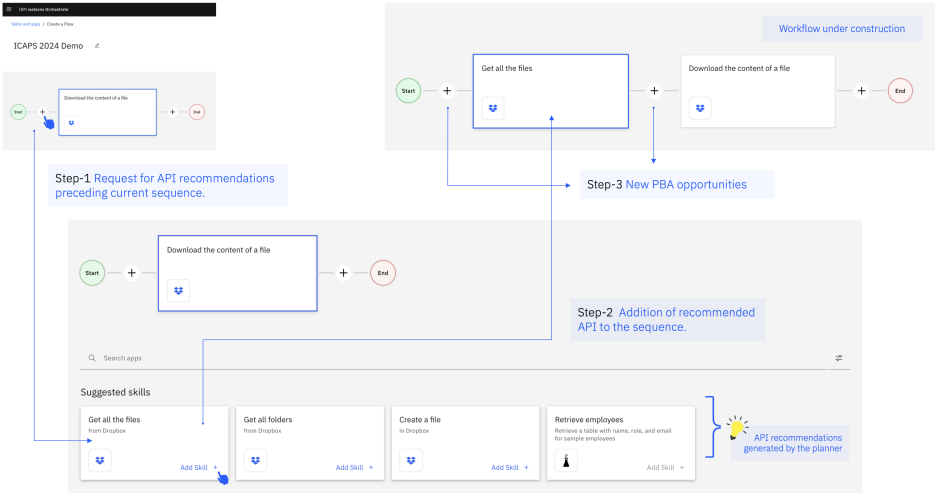
- Hypothesis Exploration for Malware Detection, AAAI 2013
- Automated Large-scale data analysis, ICAPS 2015
- Scenario planning for enterprise risk management, AAAI 2018
- D3WA+: A Case Study of XAIP in a Model Acquisition Task, ICAPS 2020
- Exploring Context-Free Languages via Planning, ICAPS 2020
- Automated Planning for Business Process Management. BPM 2021
- Interactive Plan Selection (Lemming), AAAI 2024
- Sequencing APIs through AI Planning, ICAPS 2024
- ...

Interactive Plan Selection (Lemming)

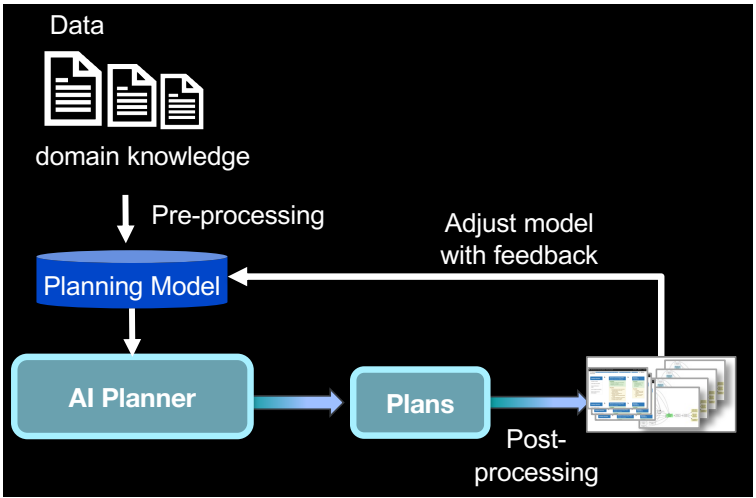


- Selection view: focus greedily on selection of disjunctive action landmarks to make the most disambiguating choices
- Build view: build a plan when selection view becomes too large

Sequencing APIs through AI Planning



General framework for Computing more than one Plan



Why Multiple Plans

1. Under-specified action models
   - The domain knowledge may not be complete. Knowledge is unknown or is hard to represent (e.g., missing precondition or effect)
2. Under-specified user preferences and domain constraints
   - Action ordering preferences not known upfront or cannot be represented
   - Known unimportant orderings
3. Replanning/plan failure

Different Class of Problems

Multiple plans are often summarized/clustered to provide the recommendation.

- want k best quality plans $\rightarrow$ top-k planning
- want top-quality plans (bound on the quality) $\rightarrow$ top-quality planning
  - do not care about the order of actions $\rightarrow$ unordered top-quality planning
  - care about some orders but not all $\rightarrow$ partially-ordered top-quality planning
- want a set of diverse plans $\rightarrow$ diverse planning
  - bounded on quality or diversity or both
  - care about optimally wrt quality or diversity or both
  - ...

Outline

1. Introduction and motivation
2. **Planning problems**
   1. Diverse planning
   2. Top-k and top-quality planning
3. Planning methods
   1. ForbidIterative
   2. K*
   3. SymK
4. Hands-on session

Classical Planning Model

Classical planning model is a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where

- Finite and discrete state space S
- A known initial state $s_0 \in S$
- A set $S_G \subseteq S$ of goal states
- Actions $A(s) \subseteq A$ applicable in each $s \in S$
- A deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- Non-negative action costs $c(a, s)$

Classical Planning Model

Classical planning model is a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where

- Finite and discrete state space S
- A known initial state $s_0 \in S$
- A set $S_G \subseteq S$ of goal states
- Actions $A(s) \subseteq A$ applicable in each $s \in S$
- A deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- Non-negative action costs $c(a, s)$

Solution: sequence of applicable actions that maps s_0 into S_G

Language for Classical Planning: Strips

A Strips Planning task is 5-tuple $\Pi = \langle F, O, c, I, G \rangle$:

- F: finite set of atoms (boolean variables)
- O: finite set of operators (actions) of form $\langle Add, Del, Pre \rangle$ (Add/Delete/Preconditions; subsets of atoms)
- $c : O \mapsto \mathbb{R}^{0+}$ captures operator cost
- I: initial state (subset of atoms)
- G: goal description (subset of atoms)

Plan: sequence of applicable actions that maps I into a state consistent with G

From Language to Models

A Strips Planning task $\Pi = \langle F, O, c, I, G \rangle$ determines state model $\mathcal{S}(\Pi)$ where

- the states $s \in S$ are collections of atoms from F
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are ops in O s.t. $Pre(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s) = c(a)$

♠ Solutions of $\mathcal{S}(\Pi)$ are plans of Π

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better
- Top-k planning: find k plans such that no cheaper plans exist

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better
- Top-k planning: find k plans such that no cheaper plans exist
- Top-quality planning: find all plans up to a certain cost

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better
- Top-k planning: find k plans such that no cheaper plans exist
- Top-quality planning: find all plans up to a certain cost
- Diverse planning: variety of problems, aiming at obtaining diverse set of plans, considering plan quality as well

Deep Dive – Diverse Planning History

Diverse planning: Definitions

For Π and natural number k, find a set of plans P such that:

- e.g., Coman & Munoz-Avila AAAI2011:
  $|P| = k$
- Srivastava et al. IJCAI2007, Nguyen et al. AIJ2012:
  $|P| = k, \ \min_{\pi,\pi' \in P} \delta(\pi, \pi') \geq d$
- Vadlamudi & Kambhampati AAAI2016:
  $|P| = k, \ \min_{\pi,\pi' \in P} \delta(\pi, \pi') \geq d, \ \max_{\pi \in P} cost(\pi) \leq c$
- ⟨here comes your favorite definition ⟩

Deep Dive – Diverse Planning History: Diversity Metrics

- Stability [Fox et al. 2006; Comman and Muñoz-Avila 2011]

$$\text{sim}_{\text{stability}}(\pi, \pi') = |A(\pi) \cap A(\pi')|/|A(\pi) \cup A(\pi')|$$

- State [Nguyen et al. 2012]

$$\text{sim}_{\text{state}}(\pi, \pi') = \sum_{i=1}^{k'} \Delta(s_i, s'_i)\ k, \text{ where } \Delta(s, s') = |s \cap s'|/|s \cup s'|$$

- landmarks [Bryce 2014] defined as the size of the symmetric difference of disjunctive landmark disjuncts satisfied by the plans
- Uniqueness [Roberts, Howe, and Ray 2014] It measures whether two plans are permutations of each other, or one plan is a partial plan of the other

Deep Dive – Diverse Planning History: Quality Metrics

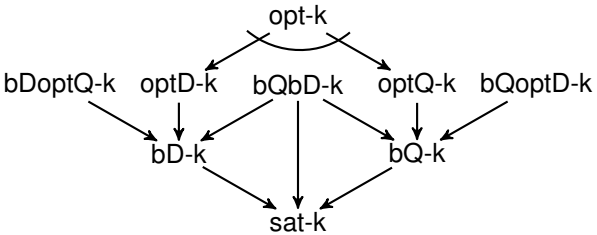
Previous quality metric (relative to known set of plans P)

$$Q_P(P') := \frac{1}{|P'|} \times \sum_{i=1}^{|P'|} \frac{cost(\pi_i)}{cost(\pi'_i)}$$

New quality metric (absolute)

$$Q_a(P') = \frac{c^*}{\max\limits_{\pi \in P'} cost(\pi)}$$

Deep Dive – Diverse Planning Taxonomy of Computational Problems



sat-k Find any solution P ($|P| \leq k$, $|P| < k \Rightarrow P =$ all plans)

bD-k (bQ-k) Find solution P s.t. $D(P) \geq d$ ($Q(P) \geq c$)

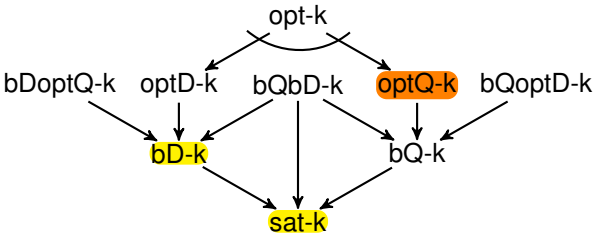
⋮

optD-k Find solution P s.t. $D(P) = \max_{P'} D(P')$

optQ-k Find solution P s.t. $Q(P) = \max_{P'} Q(P')$ (top-k)

opt-k Find solution P on the pareto frontier

Deep Dive – Diverse Planning Taxonomy of Computational Problems



sat-k Find any solution P ($|P| \leq k$, $|P| < k \Rightarrow P =$ all plans)

bD-k (bQ-k) Find solution P s.t. $D(P) \geq d$ ($Q(P) \geq c$)

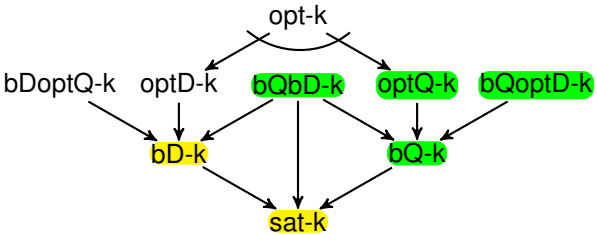
$\vdots$

optD-k Find solution P s.t. $D(P) = \max_{P'} D(P')$

optQ-k Find solution P s.t. $Q(P) = \max_{P'} Q(P')$ (top-k)

opt-k Find solution P on the pareto frontier

Deep Dive – Diverse Planning Taxonomy of Computational Problems



sat-k Find any solution P ($|P| \leq k$, $|P| < k \Rightarrow P =$ all plans)

bD-k (bQ-k) Find solution P s.t. $D(P) \geq d$ ($Q(P) \geq c$)

$\vdots$

optD-k Find solution P s.t. $D(P) = \max_{P'} D(P')$

optQ-k Find solution P s.t. $Q(P) = \max_{P'} Q(P')$ (top-k)

opt-k Find solution P on the pareto frontier

Deep Dive – Top-k Planning

Π : planning task, $\mathcal{P}_\Pi$: the set of all plans for Π

Cost-optimal planning

For Π, find a plan π such that $cost(\pi) \leq cost(\pi')$ for all $\pi' \in \mathcal{P}_\Pi$

Deep Dive – Top-k Planning

Π : planning task, $\mathcal{P}_\Pi$: the set of all plans for Π

Cost-optimal planning

For Π, find a plan π such that $cost(\pi) \leq cost(\pi')$ for all $\pi' \in \mathcal{P}_\Pi$

Top-k planning

For Π and natural number k, find a set of plans P such that:

1. $|P| \leq k$, with $|P| < k$ implying $P = \mathcal{P}_\Pi$
2. for a plan $\pi \in P$, all plans of smaller cost are also in P

Deep Dive – Top-k Planning

Π : planning task, $\mathcal{P}_\Pi$: the set of all plans for Π

Cost-optimal planning

For Π, find a plan π such that $cost(\pi) \leq cost(\pi')$ for all $\pi' \in \mathcal{P}_\Pi$

Top-k planning

For Π and natural number k, find a set of plans P such that:

1. $|P| \leq k$, with $|P| < k$ implying $P = \mathcal{P}_\Pi$
2. for a plan $\pi \in P$, all plans of smaller cost are also in P

⚠ Where does k come from? Often chosen to ensure that a sufficient number of plans is found, increased if not

⚠ Usable? Plans can be indistinguishable by some applications
prominent example: ordering of actions in a plan

Deep Dive – Top-quality planning

Top-quality planning

Given: planning task Π, natural number q
Find: the set of plans $P = \{\pi \in \mathcal{P}_\Pi \mid cost(\pi) \leq q\}$.

Quotient top-quality planning problem

Given: planning task Π, equivalence relation N over its set of plans $\mathcal{P}_\Pi$, natural number q
Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that $\bigcup_{\pi \in P} N[\pi]$ is the solution to the top-quality planning problem.

Deep Dive – Quotient top-quality planning

Unordered top-quality planning problem

Given: planning task Π, natural number q

Find: a solution to the quotient top-quality planning problem under the equivalence relation

$$\mathsf{U}_\Pi = \{(\pi, \pi') \mid \pi, \pi' \in \mathcal{P}_\Pi,\ \mathsf{MS}(\pi) = \mathsf{MS}(\pi')\}.$$

Deep Dive – Quotient top-quality planning

Unordered top-quality planning problem

Given: planning task Π, natural number q

Find: a solution to the quotient top-quality planning problem under the equivalence relation

$$\mathsf{U}_\Pi = \{(\pi, \pi') \mid \pi, \pi' \in \mathcal{P}_\Pi,\ \mathsf{MS}(\pi) = \mathsf{MS}(\pi')\}.$$

Partially ordered top-quality planning problem

Given: planning task Π, set of actions X, and a natural number q

Find: a solution to the quotient top-quality planning problem under the equivalence relation

$$\mathsf{P}_X = \{(\pi, \pi') \mid \pi, \pi' \in \mathcal{P}_\Pi, \mathsf{MS}(\pi) = \mathsf{MS}(\pi'), \pi|_X = \pi'|_X\}.$$

Deep Dive – More top-quality planning problems

Subset top-quality planning problem

Given: planning task Π and a natural number q

Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

1. $\forall \pi \in P$, $cost(\pi) \leq q$, and
2. $\forall \pi' \in \mathcal{P}_\Pi \setminus P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R_\subset$.

$$R_\subset = \{(\pi, \pi') \mid \mathsf{MS}(\pi) \subset \mathsf{MS}(\pi')\}$$

Deep Dive – More top-quality planning problems

Subset top-quality planning problem

Given: planning task Π and a natural number q

Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

i. $\forall \pi \in P$, $cost(\pi) \leq q$, and

ii. $\forall \pi' \in \mathcal{P}_\Pi \setminus P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R_\subset$.

$$R_\subset = \{(\pi, \pi') \mid \mathsf{MS}(\pi) \subset \mathsf{MS}(\pi')\}$$

Loopless top-k planning problem

Given: planning task Π and a natural number k

Find: a set of plans $P \subseteq \mathcal{P}^{\ell\ell}_\Pi$ (all plans without loops) such that

i. $|P| \leq k$, with $|P| < k$ implying $P = \mathcal{P}^{\ell\ell}_\Pi$

ii. for a plan $\pi \in P$, all plans of smaller cost are also in P

Deep Dive – More top-quality planning problems

Subset top-quality planning problem

Given: planning task Π and a natural number q

Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

(i) $\forall \pi \in P$, $cost(\pi) \leq q$, and

(ii) $\forall \pi' \in \mathcal{P}_\Pi \setminus P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R_\subset$.

$$R_\subset = \{(\pi, \pi') \mid \mathsf{MS}(\pi) \subset \mathsf{MS}(\pi')\}$$

Loopless top-quality planning problem

Given: planning task Π and a natural number q

Find: a set of plans $P = \{\pi \in \mathcal{P}^{\ell\ell}_\Pi \mid cost(\pi) \leq q\}$.

Deep Dive – Dominance top-quality planning

Dominance top-quality planning problem

Given: planning task Π and a natural number q
Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

(i) $\forall \pi \in P$, $cost(\pi) \leq q$,

(ii) $\forall \pi' \notin P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R$,

(iii) P is minimal under $\subseteq$ among all $P' \subseteq \mathcal{P}_\Pi$ for which conditions (i) and (ii) hold.

Deep Dive – Dominance top-quality planning

Dominance top-quality planning problem

Given: planning task Π and a natural number q
Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

(i) $\forall \pi \in P$, $cost(\pi) \leq q$,

(ii) $\forall \pi' \notin P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R$,

(iii) P is minimal under $\subseteq$ among all $P' \subseteq \mathcal{P}_\Pi$ for which conditions (i) and (ii) hold.

- for an empty relation we get the top-quality planning problem

Deep Dive – Dominance top-quality planning

Dominance top-quality planning problem

Given: planning task Π and a natural number q
Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

(i) $\forall \pi \in P$, $cost(\pi) \leq q$,

(ii) $\forall \pi' \notin P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R$,

(iii) P is minimal under $\subseteq$ among all $P' \subseteq \mathcal{P}_\Pi$ for which conditions (i) and (ii) hold.

- for R_U we get the unordered top-quality planning problem

Deep Dive – Dominance top-quality planning

Dominance top-quality planning problem

Given: planning task Π and a natural number q
Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

(i) $\forall \pi \in P$, $cost(\pi) \leq q$,

(ii) $\forall \pi' \notin P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R$,

(iii) P is minimal under $\subseteq$ among all $P' \subseteq \mathcal{P}_\Pi$ for which conditions (i) and (ii) hold.

for $R_\subset$ we get subset top-quality planning problem

Deep Dive – Dominance top-quality planning

Dominance top-quality planning problem

Given: planning task Π and a natural number q
Find: a set of plans $P \subseteq \mathcal{P}_\Pi$ such that

i. $\forall \pi \in P$, $cost(\pi) \leq q$,

ii. $\forall \pi' \notin P$ with $cost(\pi') \leq q$, $\exists \pi \in P$ such that $(\pi, \pi') \in R$,

iii. P is minimal under $\subseteq$ among all $P' \subseteq \mathcal{P}_\Pi$ for which conditions (i) and (ii) hold.

Loopless relation $R_{\ell\ell}$

$(\pi, \pi') \in R_{\ell\ell}$ if and only if (a) $\pi \in \mathcal{P}^{\ell\ell}_\Pi$ and (b) if S' are the states traversed by π, then π' traverses some $s \in S'$ more than once.

for $R_{\ell\ell}$ we get loopless top-quality planning problem

The Three Questions

How hard are these problems?

Surprisingly very few results known

The Three Questions

How hard are these problems?

Surprisingly very few results known

How to solve these problems in practice?

Next section

The Three Questions

How hard are these problems?

Surprisingly very few results known

How to solve these problems in practice?

Next section

How to check whether a set of plans is a solution?

Tuesday, 13:00 - 14:30

Computational complexity of finding multiple plans

Consider the decision problem analogous to top-k planning.

Bounded Top-k Plan Existence Problem

GIVEN: Planning task Π, length bound $\ell \in \mathbb{N}_0$, #plans $k \in \mathbb{N}_0$
QUESTION: Are there at least k plans for Π of length at most ℓ?

$\rightsquigarrow$ PSPACE-complete (as finding a single plan!)

Polynomially Bounded Top-k Plan Existence Problem

GIVEN: Planning task Π, polynomial p, length bound $\ell \in \mathbb{N}_0$, #plans $k \in \mathbb{N}_0$
QUESTION: Are there at least k plans for Π of length at most $\ell \leq p(|\Pi|)$?

$\rightsquigarrow$ PP-hard (vs. finding a single plan: NP-complete)

$\rightsquigarrow$ In practice: very likely to be much harder than ordinary planning!

Computational complexity of finding multiple plans

Consider the decision problem analogous to top-k planning.

Bounded Top-k Plan Existence Problem

GIVEN: Planning task Π, length bound $\ell \in \mathbb{N}_0$, #plans $k \in \mathbb{N}_0$
QUESTION: Are there at least k plans for Π of length at most ℓ?

$\rightsquigarrow$ PSPACE-complete (as finding a single plan!)

Polynomially Bounded Top-k Plan Existence Problem

GIVEN: Planning task Π, polynomial p, length bound $\ell \in \mathbb{N}_0$, #plans $k \in \mathbb{N}_0$
QUESTION: Are there at least k plans for Π of length at most $\ell \leq p(|\Pi|)$?

$\rightsquigarrow$ PP-hard (vs. finding a single plan: NP-complete)

$\rightsquigarrow$ In practice: very likely to be much harder than ordinary planning!

Computational complexity of finding multiple plans

Consider the decision problem analogous to top-k planning.

Bounded Top-k Plan Existence Problem

GIVEN: Planning task Π, length bound $\ell \in \mathbb{N}_0$, #plans $k \in \mathbb{N}_0$
QUESTION: Are there at least k plans for Π of length at most ℓ?

$\rightsquigarrow$ PSPACE-complete (as finding a single plan!)

Polynomially Bounded Top-k Plan Existence Problem

GIVEN: Planning task Π, polynomial p, length bound $\ell \in \mathbb{N}_0$, #plans $k \in \mathbb{N}_0$
QUESTION: Are there at least k plans for Π of length at most $\ell \leq p(|\Pi|)$?

$\rightsquigarrow$ PP-hard (vs. finding a single plan: NP-complete)

$\rightsquigarrow$ In practice: very likely to be much harder than ordinary planning!

Outline

ForbidIterative

Idea

1. Use classical planners to find a single solution
2. Optional: extend the set of found solutions (by e.g., symmetries)
3. Transform the planning problem to forbid the found solutions (+more)
4. Repeat until enough plans found or proved unsolvable

ForbidIterative

Idea

1. Use classical planners to find a single solution
2. Optional: extend the set of found solutions (by e.g., symmetries)
3. Transform the planning problem to forbid the found solutions (+more)
4. Repeat until enough plans found or proved unsolvable

Transformations

- Forbid a single plan (path) or multiple plans (graph) – top-k and top-quality

ForbidIterative

Idea

1. Use classical planners to find a single solution
2. Optional: extend the set of found solutions (by e.g., symmetries)
3. Transform the planning problem to forbid the found solutions (+more)
4. Repeat until enough plans found or proved unsolvable

Transformations

- Forbid a single plan (path) or multiple plans (graph) – top-k and top-quality
- Forbid a plan and all its reorderings – unordered top-quality

ForbidIterative

Idea

1. Use classical planners to find a single solution
2. Optional: extend the set of found solutions (by e.g., symmetries)
3. Transform the planning problem to forbid the found solutions (+more)
4. Repeat until enough plans found or proved unsolvable

Transformations

- Forbid a single plan (path) or multiple plans (graph) – top-k and top-quality
- Forbid a plan and all its reorderings – unordered top-quality
- Forbid a plan and all plans whose actions multisets are super-sets of its actions multiset – subset top-quality

ForbidIterative

Idea

1. Use classical planners to find a single solution
2. Optional: extend the set of found solutions (by e.g., symmetries)
3. Transform the planning problem to forbid the found solutions (+more)
4. Repeat until enough plans found or proved unsolvable

Transformations

- Forbid a single plan (path) or multiple plans (graph) – top-k and top-quality
- Forbid a plan and all its reorderings – unordered top-quality
- Forbid a plan and all plans whose actions multisets are super-sets of its actions multiset – subset top-quality
- Forbid all plans that start from a particular sequence of actions

ForbidIterative

Idea

1. Use classical planners to find a single solution
2. Optional: extend the set of found solutions (by e.g., symmetries)
3. Transform the planning problem to forbid the found solutions (+more)
4. Repeat until enough plans found or proved unsolvable

Transformations

- Forbid a single plan (path) or multiple plans (graph) – top-k and top-quality
- Forbid a plan and all its reorderings – unordered top-quality
- Forbid a plan and all plans whose actions multisets are super-sets of its actions multiset – subset top-quality
- Forbid all plans that start from a particular sequence of actions
- Forbid a plan and all plans that loop through at least one of its states.

Plan Forbid Transformation

Input: task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ and plan $\pi = o_1 \dots o_n$.
Output: task $\Pi^-_\pi = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost' \rangle$.

- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}, \overline{v}_0, \dots \overline{v}_n\}$, additional binary variables,
- $s'_0[v] = s_0[v]$ for all $v \in \mathcal{V}$,
  $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, and $s'_0[\overline{v}_i] = 0$ for all $1 \leq i \leq n$,
- $s'_\star[v] = s_\star[v]$ for all $v \in \mathcal{V}$ s.t. $s_\star[v]$ defined, and $s'_\star[\overline{v}] = 0$,
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O}, o \notin \pi\} \cup \{o^f_1, \dots, o^f_n\} \cup \{o^d, o^r \mid o \in \pi\}$,
  $cost'(o^e) = cost'(o^f) = cost'(o^d) = cost'(o^r) = cost(o)$,
  where

Plan Forbid Transformation

Input: task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ and plan $\pi = o_1 \dots o_n$.
Output: task $\Pi^-_\pi = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost' \rangle$.

- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}, \overline{v}_0, \dots \overline{v}_n\}$, additional binary variables,
- $s'_0[v] = s_0[v]$ for all $v \in \mathcal{V}$,
  $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, and $s'_0[\overline{v}_i] = 0$ for all $1 \leq i \leq n$,
- $s'_\star[v] = s_\star[v]$ for all $v \in \mathcal{V}$ s.t. $s_\star[v]$ defined, and $s'_\star[\overline{v}] = 0$,
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O}, o \notin \pi\} \cup \{o^f_1, \dots, o^f_n\} \cup \{o^d, o^r \mid o \in \pi\}$,
  $cost'(o^e) = cost'(o^f) = cost'(o^d) = cost'(o^r) = cost(o)$,
  where

$$o^e = \langle pre(o), eff(o) \cup \{\langle \overline{v}, 0 \rangle\} \rangle$$

Action not on plan – discarding plan from consideration

Plan Forbid Transformation

Input: task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ and plan $\pi = o_1 \dots o_n$.
Output: task $\Pi^-_\pi = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost' \rangle$.

- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}, \overline{v}_0, \dots \overline{v}_n\}$, additional binary variables,
- $s'_0[v] = s_0[v]$ for all $v \in \mathcal{V}$,
  $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, and $s'_0[\overline{v}_i] = 0$ for all $1 \leq i \leq n$,
- $s'_\star[v] = s_\star[v]$ for all $v \in \mathcal{V}$ s.t. $s_\star[v]$ defined, and $s'_\star[\overline{v}] = 0$,
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O}, o \notin \pi\} \cup \{o^f_1, \dots, o^f_n\} \cup \{o^d, o^r \mid o \in \pi\}$,
  $cost'(o^e) = cost'(o^f) = cost'(o^d) = cost'(o^r) = cost(o)$,
  where

$$o^f_i = \langle pre(o_i) \cup \{\langle \overline{v}, 1 \rangle, \langle \overline{v}_{i-1}, 1 \rangle\}, \quad \mathit{eff}(o_i) \cup \{\langle \overline{v}_{i-1}, 0 \rangle, \langle \overline{v}_i, 1 \rangle\} \rangle$$

Action on plan, following the plan

Plan Forbid Transformation

Input: task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ and plan $\pi = o_1 \dots o_n$.
Output: task $\Pi^-_\pi = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost' \rangle$.

- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}, \overline{v}_0, \dots \overline{v}_n\}$, additional binary variables,
- $s'_0[v] = s_0[v]$ for all $v \in \mathcal{V}$,
  $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, and $s'_0[\overline{v}_i] = 0$ for all $1 \leq i \leq n$,
- $s'_\star[v] = s_\star[v]$ for all $v \in \mathcal{V}$ s.t. $s_\star[v]$ defined, and $s'_\star[\overline{v}] = 0$,
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O}, o \notin \pi\} \cup \{o^f_1, \dots, o^f_n\} \cup \{o^d, o^r \mid o \in \pi\}$,
  $cost'(o^e) = cost'(o^f) = cost'(o^d) = cost'(o^r) = cost(o)$,
  where

$$o^d = \langle pre(o) \cup \{\langle \overline{v}, 1 \rangle\} \cup \bigcup_{o=o_i} \{\langle \overline{v}_{i-1}, 0 \rangle\}, \mathit{eff}(o) \cup \{\langle \overline{v}, 0 \rangle\} \rangle$$

Action on plan, but breaks following the plan – discarding plan from consideration

Plan Forbid Transformation

Input: task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ and plan $\pi = o_1 \dots o_n$.
Output: task $\Pi^-_\pi = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost' \rangle$.

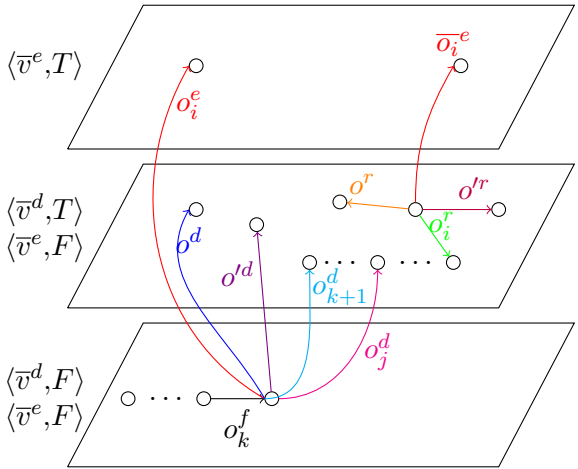
- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}, \overline{v}_0, \dots \overline{v}_n\}$, additional binary variables,
- $s'_0[v] = s_0[v]$ for all $v \in \mathcal{V}$,
  $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, and $s'_0[\overline{v}_i] = 0$ for all $1 \leq i \leq n$,
- $s'_\star[v] = s_\star[v]$ for all $v \in \mathcal{V}$ s.t. $s_\star[v]$ defined, and $s'_\star[\overline{v}] = 0$,
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O}, o \notin \pi\} \cup \{o^f_1, \dots, o^f_n\} \cup \{o^d, o^r \mid o \in \pi\}$,
  $cost'(o^e) = cost'(o^f) = cost'(o^d) = cost'(o^r) = cost(o)$,
  where

$$o^r = \langle pre(o) \cup \{\langle \overline{v}, 0 \rangle\}, \mathit{eff}(o) \rangle$$

Action on plan, but the plan is already discarded from consideration

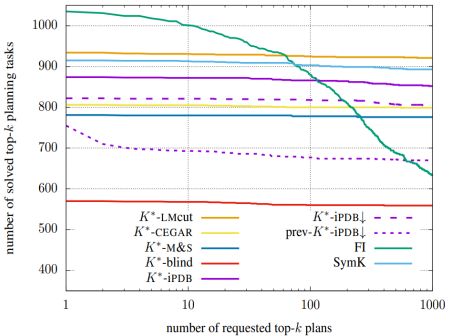
Loopless top-quality planning Transformation

- ☺ Based on *Plan Forbid Transformation*
- 👍 Given $\pi \in \mathcal{P}_{\Pi}^{\ell\ell}$, forbids π and all π' such that $(\pi, \pi') \in R_{\ell\ell}$.



K^* Planner

The state of the art planner for finding multiple plans.



[Lee, Katz, and Sohrabi 2023]

Overview of K^* Planner

- Implements K^* search [Aljazzar and Leue 2011] in Fast Downward [Helmert, M. 2006]
- K^* alternates A^* [Hart, Nilsson, and Raphael 1968] and Eppstein's Algorithm for finding k-shortest paths [Eppstein 1998]
- Several modifications made to K^* and Eppstein's Algorithm to improve performance and to use as a planner

Variations of K^* Planner

- Integrating K^* search with existing techniques is easy
  - K^* stores all edges (side tracked edges) generated during A^* search and decodes all available paths using Eppstein's algorithm
  - Any adaptation made to A^* would work in K^* with a minor modification

- State-of-the art performance on top-k and top-quality planning problems
  - **Top-K**: K^* + Symmetry Pruning [Katz and Lee, IJCAI 2023]
  - **Unordered Top-Q**: K^* + Partial Order Reduction [Katz and Lee, SoCS 2023]
  - **Partially Ordered Top-Q**: K^* + Modified Partial Order Reduction [Katz, Lee and Sohrabi SoCS 2024]

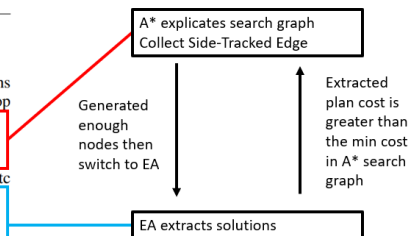
K^* Search

- Outline of K^* Search Algorithm
  - Build data structure for Eppstein's algorithm during A^*
  - Switch A^* and Eppstein's algorithm
  - Decode solution paths

Algorithm 1 Generic K^* Search for Top-k Planning

Input: Single goal-state planning task Π, k
Output: Top-k plans

```
1:  Initialize K* search
2:  P ← ∅                                   ▷ Initialize set of found plans
3:  while True do                           ▷ K* outer-loop
4:      while ¬(OPEN_A* = ∅ ∨ SWITCH-TO-EA()) do
5:          Expand an A* node
6:          Update data structures G_A*, T_A*, H_in, etc
7:      PREPAREEA()                         ▷ Update H_T, H_G, G_EA, etc
8:      while ¬(OPEN_EA = ∅ ∨ SWITCH-TO-A*()) do
9:          Expand a node in path graph G_EA
10:         Reconstruct a plan and update P
11:         if |P| = k then return P
12:     if OPEN_A* and OPEN_EA are empty then return P
```



K-Best Enumeration with Eppstein's Algorithm

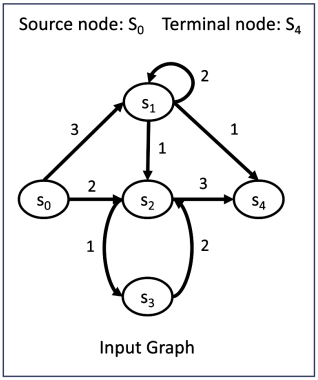
- Finding k-best paths in an explicit graph with n nodes and m edges
  - single source and single terminal node, loops allowed $O(m + n \log n + k)$ by Eppstein's algorithm [Eppstein 1998]
  - single source and single terminal node, no loop allowed $O(kn(m + n \log n)$ by Yen's algorithm [Yen 1971]
  - Note that Dijkstra' algorithm is $O(m + n \log n)$ [Dijkstra 1959]
- A recipe for enumerating K-best solutions in an optimization problem [Eppstein 2014]
  - Problem decomposition by dynamic programming
  - Solution space partitioning
  - Fast selection for finding the k-best items from a set of items

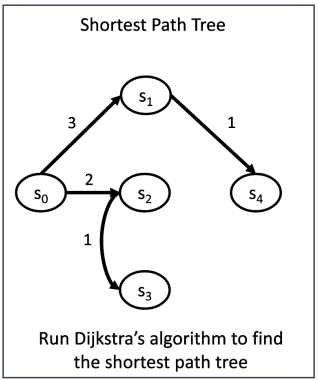
We will focus on Eppstein's algorithm and show the essential idea by examples

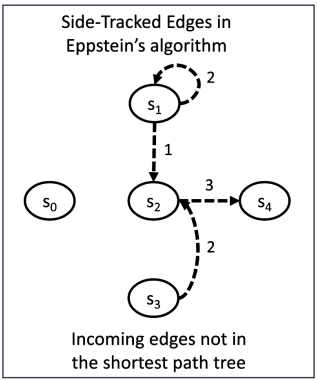
Eppstein's Path Representation 1/3

- Side-Tracked Edges (STE): Incoming edges not in the shortest path tree

  Note that STE is defined relative to incoming edges in K^* , but it is defined relative to outgoing edges in EA.

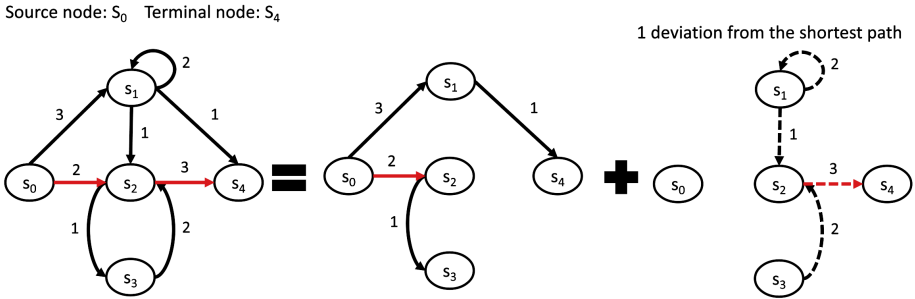






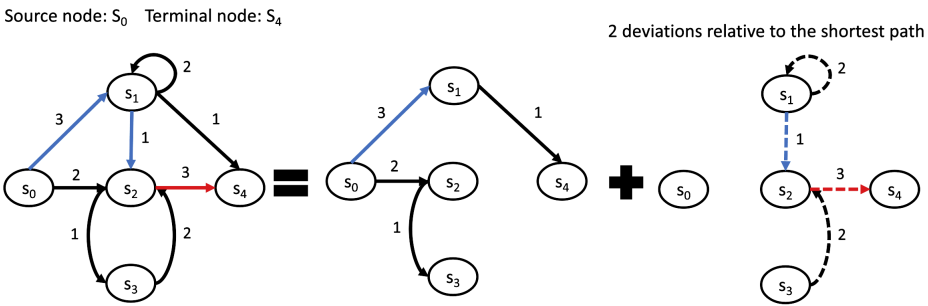
Eppstein's Path Representation 2/3

- Given a shortest path tree on the left, the suboptimal path can be encoded by one red side-tracked edge



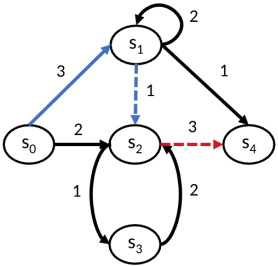
Eppstein's Path Representation 3/3

- Given a shortest path tree on the left, the suboptimal path can be encoded by one red side-tracked edge and one blue side-tracked edge



The Optimal Substructure Used in Eppstein's Algorithm

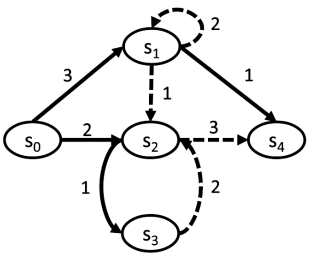
- Dynamic programming executes over the optimal substructure
  - The deviation decisions can be made along the shortest path trees from the terminal node to the source node
  - The deviation decisions can be ordered
  - Any subsequent suboptimal path obtained by adding an additional STE has equal or higher cost than the previous paths



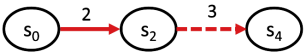
- A Side-Tracked Edge encodes a deviation decision relative to the shortest path tree
- The first decision was to deviate at S4 with (S2, S4)
- The next possible decision can be made at S0 or S2
- The second decision was to deviate at S2 with (S1, S2)

Solution Space Partitioning in Eppstein's Algorithm 1/2

- Given a sequence of side-tracked edges,
  - we can decode the corresponding path using the shortest path tree found by Dijkstra in Eppstein's algorithm (or the search tree maintained by A^* in K^*)
  - we know the total deviation cost relative to the shortest path



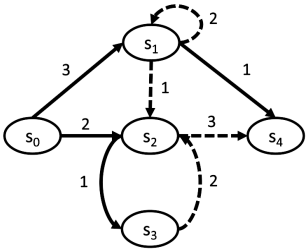
- After making the first deviation decision, the remaining possible choices are to deviate at S2 or S0

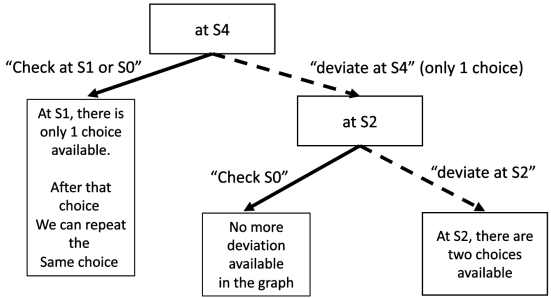


- At S2 there are two Side-Tracked Edges (S1, S2) and (S3, S2)
- At S0 there are no Side-Tracked Edges
- Those two decisions will yield two suboptimal paths with higher costs
- After selecting the second deviation on (S1, S2), we continue this selection procedure either at S1 or S0

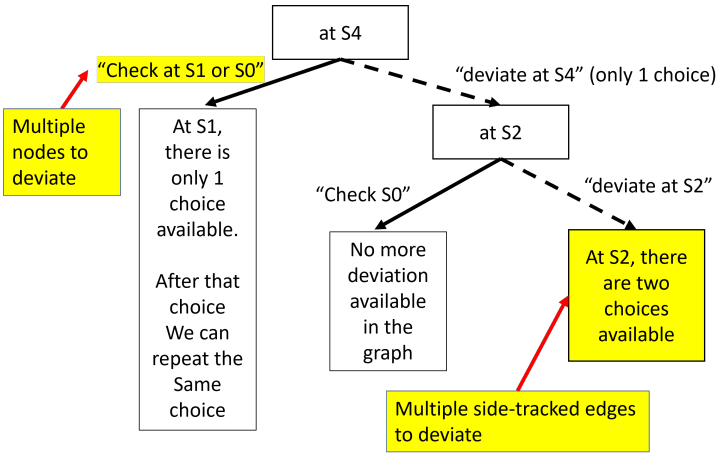
Solution Space Partitioning in Eppstein's Algorithm 2/2

- Deviation decisions can be **partitioned**
  - we can **select at which node to deviate** in the shortest path tree
  - we can **select one of the incoming side-tracked edges** at a selected node
- The paths can be grouped by those decisions in a tree



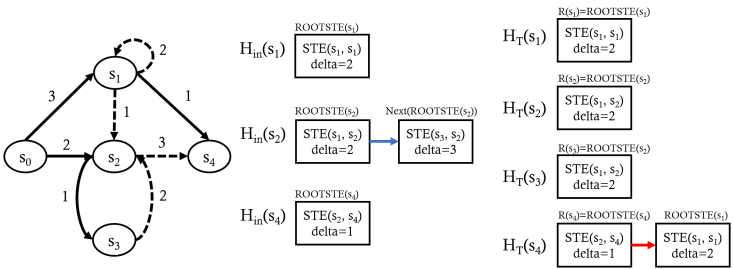


Fast Selection in Eppstein's Algorithm 1/3

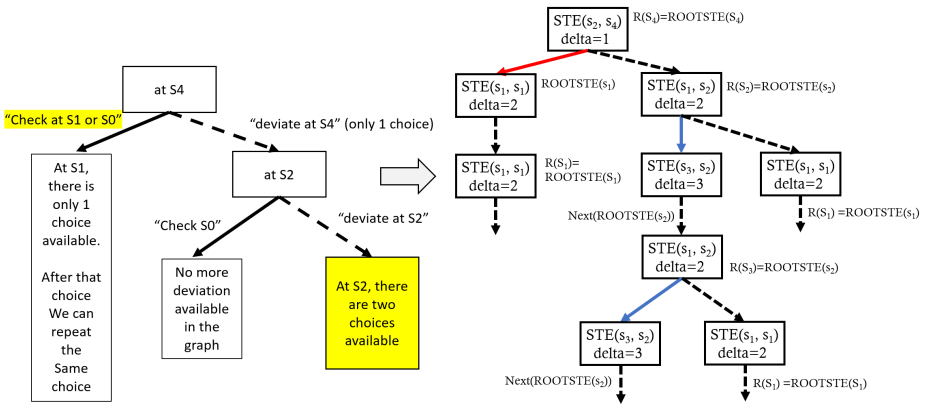


Fast Selection in Eppstein's Algorithm 2/3

- Use Heap to store side-tracked edges
  - $H_{\mathsf{T}}(s_i)$ stores the **root** side-tracked edges over the nodes traced back from s_i in the shortest path tree T, encoding the decision of **which node to deviate**
  - $H_{\mathsf{in}}(s_i)$ stores incoming side-tracked edges toward s_i, encoding selection of **one of the side-tracked edges**



Fast Selection in Eppstein's Algorithm 3/3

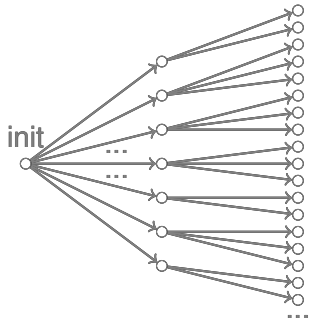


The tree on the right enumerates k-best paths

Some Modifications Made in K^* Planner

- Eppstein's search space for K-best enumeration
  - Eppstein's algorithm generates the solution paths from a graph over the heaps
  - K^* planner traverses the k-best paths with the search tree shown in the earlier slide
  - K^* planner doesn't do fast selection with heaps but it executes the dynamic programming
- Usage of Heaps
  - Eppstein's algorithm uses 3-ary heaps, heap graphs, etc
  - K^* planner stores side-tracked edge objects in a hash-table and it uses linked lists to store the pointers to the side-tracked edge objects
- Additional details for improving performance
  - Unlike the original K^* search that performs Eppstein's algorithm over the expanded nodes, K^* planner uses all generated nodes
  - K^* planner restarts Eppstein's algorithm from scratch every switching
  - K^* planner implements the lazy version of Eppstein's algorithm
- Adaptation as a PDDL planner
  - K^* planner transforms an input PDDL problem to have a single terminal node
  - The PDDL task transformation preserves the quality of heuristics under the transformation

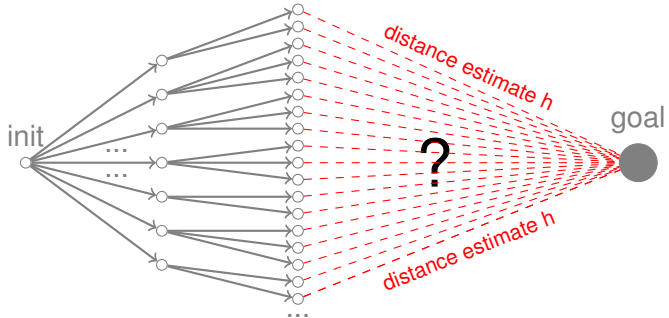
Tackling the State Space Explosion Problem



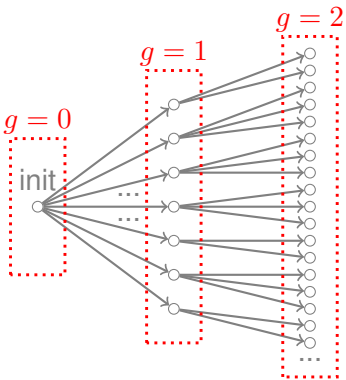
?

goal

Tackling the State Space Explosion Problem



Tackling the State Space Explosion Problem

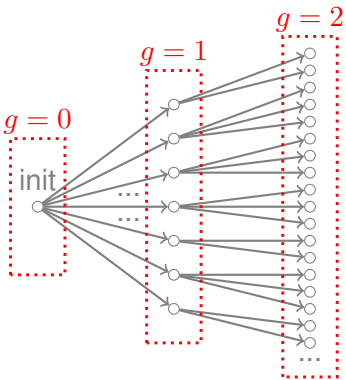


Symbolic search + Decision Diagrams
to the rescue!

?

goal

Tackling the State Space Explosion Problem



Symbolic search + Decision Diagrams
to the rescue!

?

goal

Not only good for finding a **single plan**, but also **multiple plans**. But why?

SymK: Symbolic Search for Finding Multiple Plans

Symbolic Search for finding a single plan

- Exhaustive search with concise data structures
  - Enumeration of all reachable states; including goal states
  - Paths represented implicitly but can be reconstructed

$\rightsquigarrow$ Promising candidate for efficiently finding multiple plans

$\rightsquigarrow$ Minor modifications to generate all plans with increasing costs!

SymK: Symbolic Search for Finding Multiple Plans

Symbolic Search for finding a single plan

- Exhaustive search with concise data structures
  - Enumeration of all reachable states; including goal states
  - Paths represented implicitly but can be reconstructed

$\rightsquigarrow$ Promising candidate for efficiently finding multiple plans

$\rightsquigarrow$ Minor modifications to generate all plans with increasing costs!

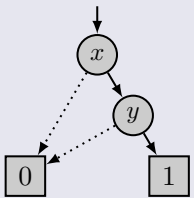
Outline

- Summary on finding a single plan with symbolic search
  - Forward, backward & bidirectional search
- Finding multiple plans with symbolic search

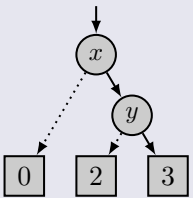
Symbolic Search in a Nutshell

- Operations on sets of states $S' \subseteq \mathcal{S}$
- S' represented by characteristic function $\chi_{S'}$
- Actions represented as transition relations
- Forward, backward & bidirectional search

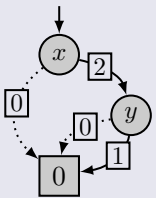
Datastructure: Decision Diagrams (DD)



Binary DD



Algebraic DD



Edge-Valued Binary DD

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

Symbolic Uniform-Cost Search

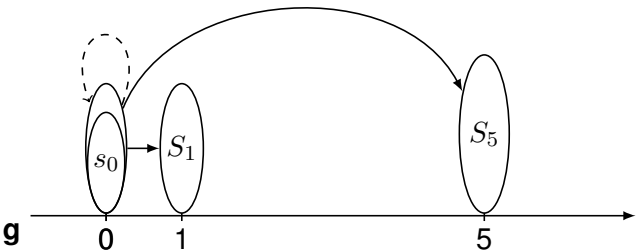
Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

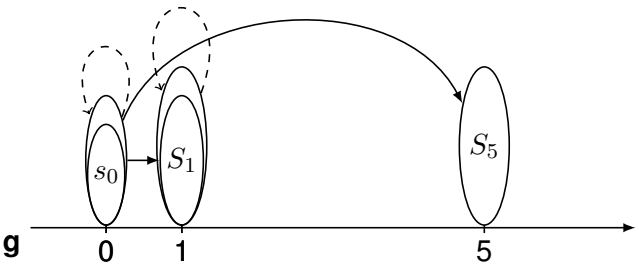
- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state



Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

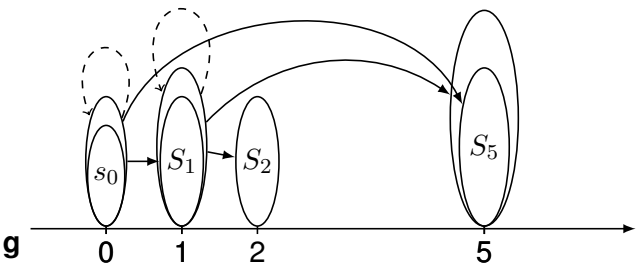
- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state



Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

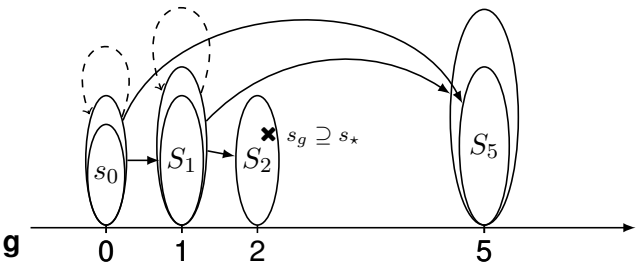
- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state



Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

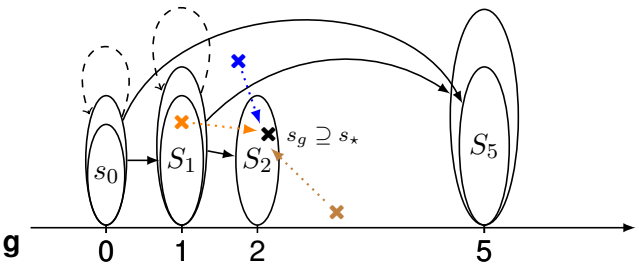


Plan reconstruction

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

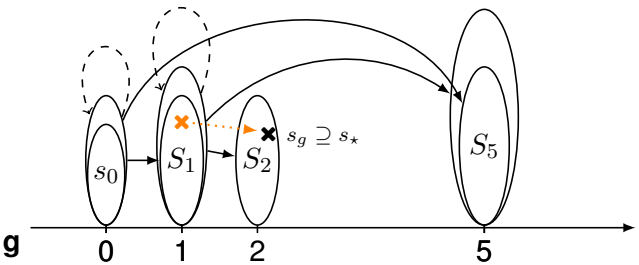


Plan reconstruction

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

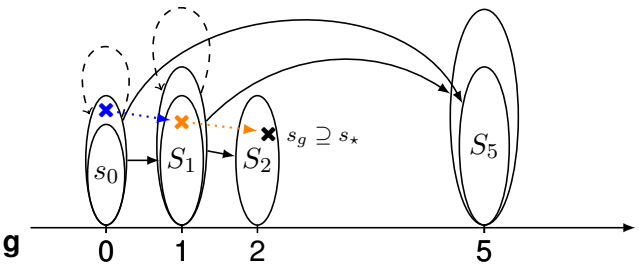


Plan reconstruction

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

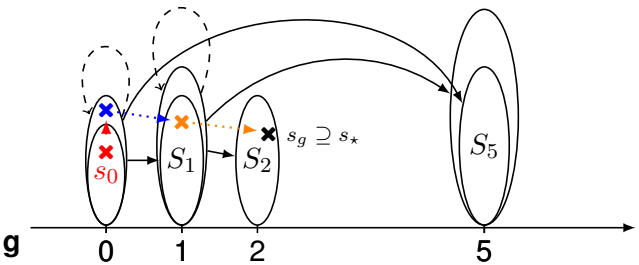


Plan reconstruction

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state

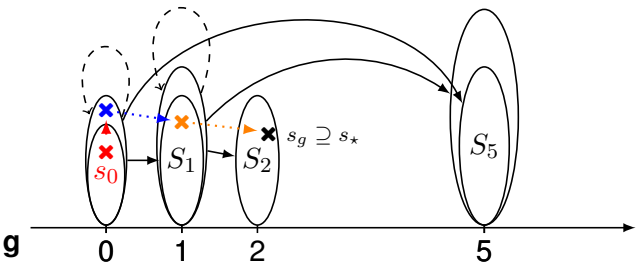


Plan reconstruction

Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each transition relation with action cost c:
  - Compute states reachable with $i + c$
  - Insert the result in the corresponding bucket
- Termination: Expansion of goal state



Plan reconstruction

1. red
2. blue
3. orange

Paths not explicitly stored!

Symbolic Backward Uniform-Cost Search

We can perform the search in backward direction:

- Start with the set of goal states
- Use preimage instead of image operation

Challenges:

1. Multiple goal states
2. Partial states
3. Spurious states

Symbolic Backward Uniform-Cost Search

We can perform the search in backward direction:

- Start with the set of goal states
- Use preimage instead of image operation

Challenges:

1. Multiple goal states $\rightsquigarrow$ Not a problem in symbolic search!
2. Partial states $\rightsquigarrow$ Not a problem in symbolic search!
3. Spurious states

Symbolic Backward Uniform-Cost Search

We can perform the search in backward direction:

- Start with the set of goal states
- Use preimage instead of image operation

Challenges:

1. Multiple goal states $\rightsquigarrow$ Not a problem in symbolic search!
2. Partial states $\rightsquigarrow$ Not a problem in symbolic search!
3. Spurious states $\rightsquigarrow$ Solution: state-invariant pruning

Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$

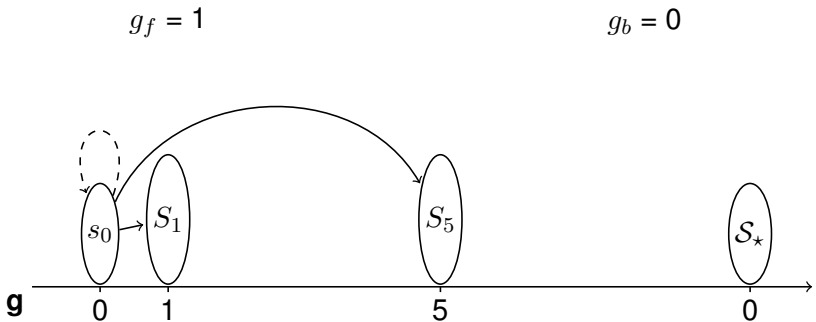
g_f = 0 g_b = 0

Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$

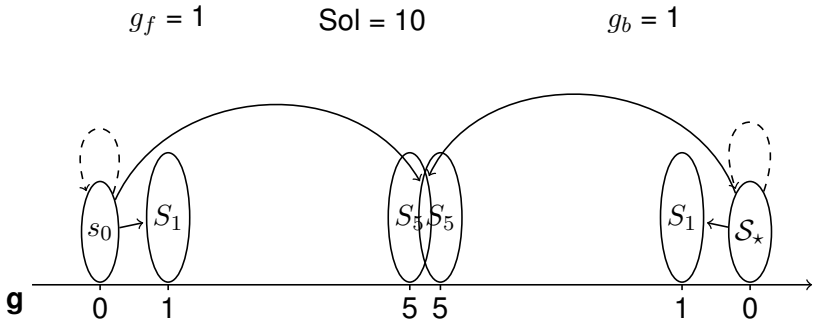
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$



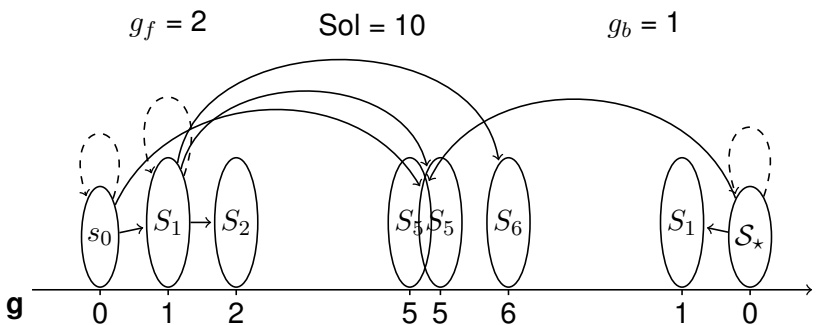
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$



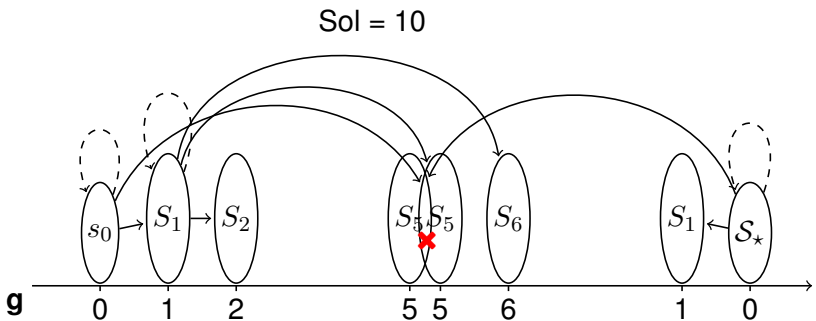
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$



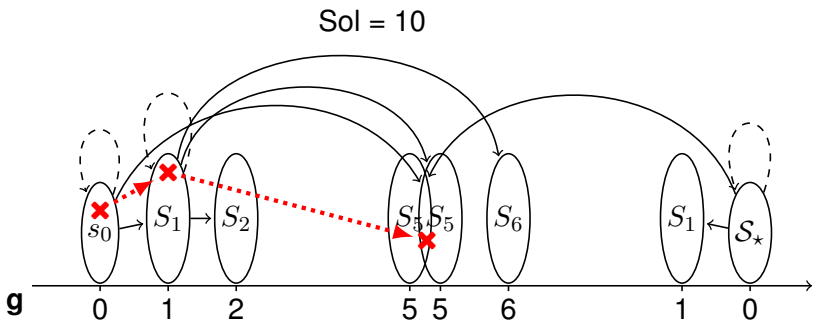
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$



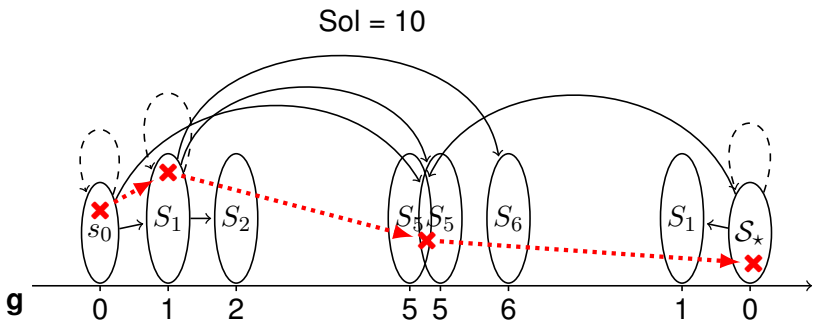
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$



Symbolic Bidirectional Uniform-Cost Search

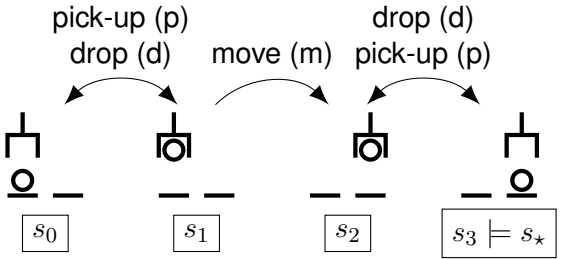
- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{o \in \mathcal{O}} cost(o) \geq Sol$



Symbolic Search for Finding Multiple Plans

- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Symbolic Search for Finding Multiple Plans

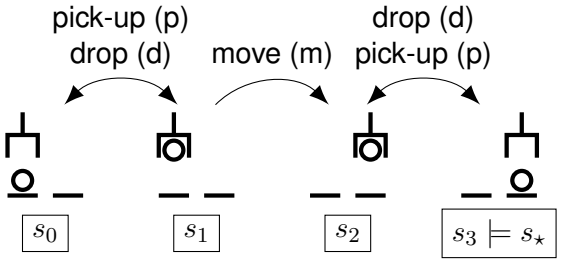


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search: s_0

S_0

Symbolic Search for Finding Multiple Plans

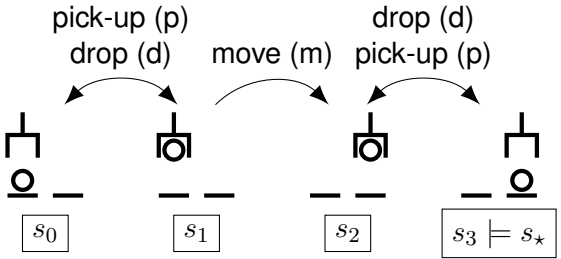


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search: s_0 s_1

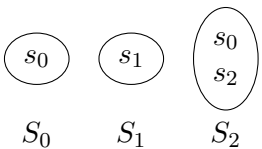
S_0 S_1

Symbolic Search for Finding Multiple Plans

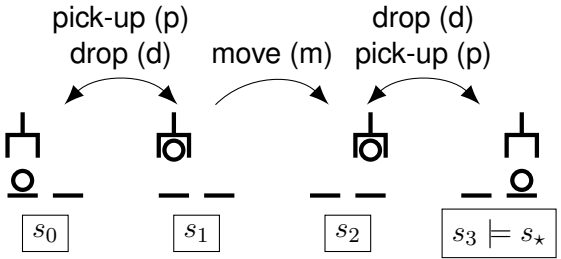


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

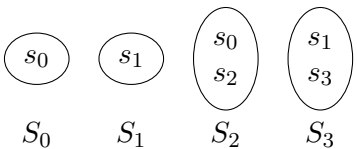


Symbolic Search for Finding Multiple Plans

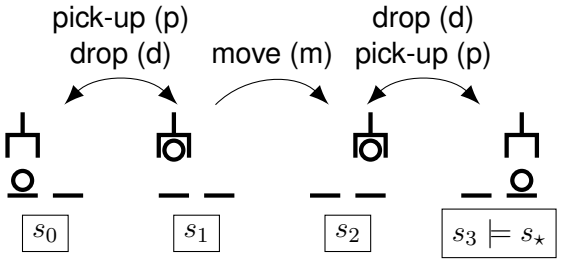


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

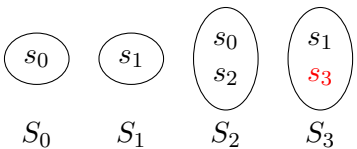


Symbolic Search for Finding Multiple Plans

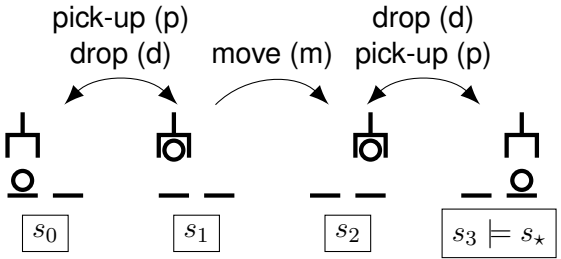


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

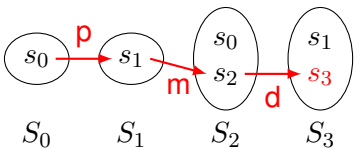


Symbolic Search for Finding Multiple Plans

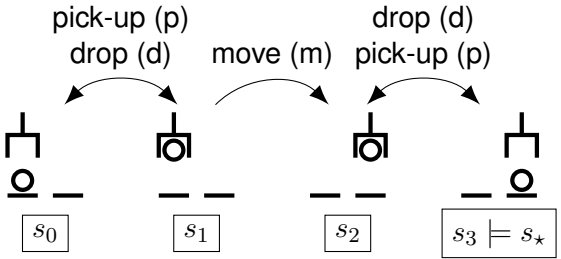


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

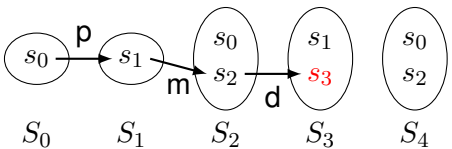


Symbolic Search for Finding Multiple Plans

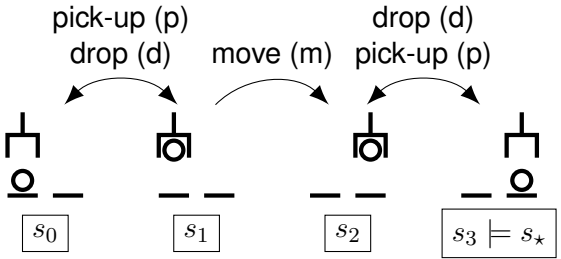


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

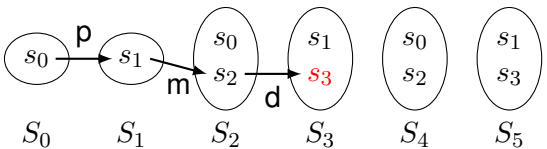


Symbolic Search for Finding Multiple Plans

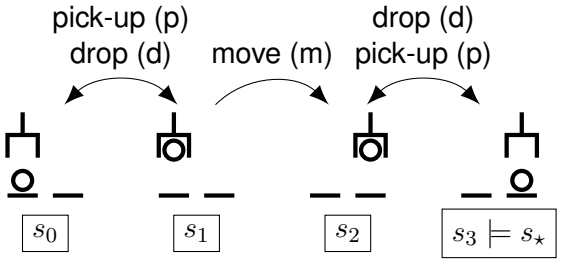


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

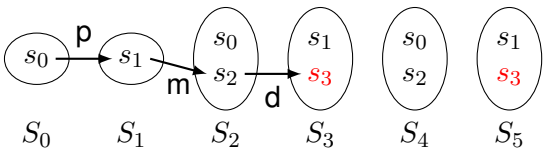


Symbolic Search for Finding Multiple Plans

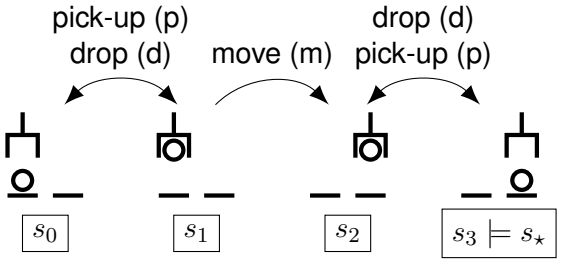


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

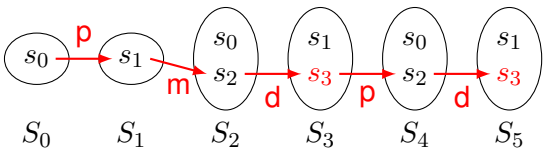


Symbolic Search for Finding Multiple Plans

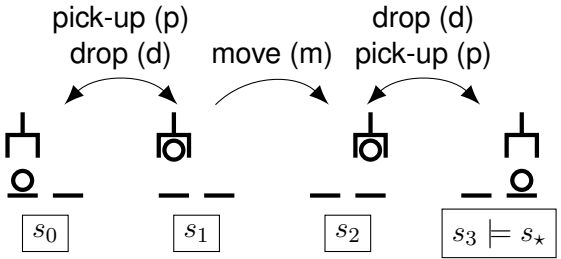


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

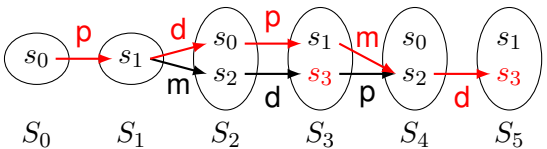


Symbolic Search for Finding Multiple Plans

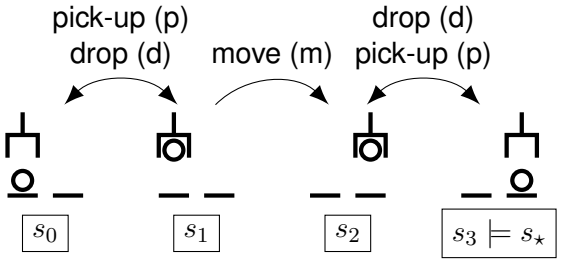


- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:

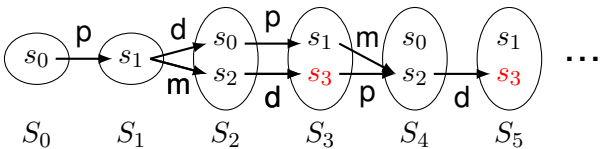


Symbolic Search for Finding Multiple Plans



- Perform symbolic search without closing states
- If goal states are expanded $\rightsquigarrow$ reconstruct all plans

Forward Search:



Symbolic Search for Top-k and Top-q Planning

Symbolic Search for Finding Multiple Plans

- Iterates plans with increasing costs $\leadsto$ anytime behavior

$\leadsto$ Top-k: Stop when k plans are found ...

$\leadsto$ Top-q: Stop when g-values of states exceed the cost bound ...

- ... or no more plans can exist
  - $\equiv$ A bit tricky: reachable states from open list form a fixed point without a goal state

Loopless Plans: Generate-and-Test Approach

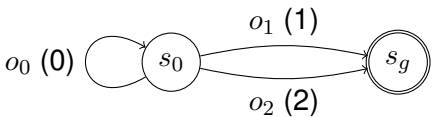
- Request a sufficiently large number of plans ($k = \infty$)
- Enumerate all plans with increasing costs
  - Add loopless plans to the solution set
  - Discard plans with loops

Loopless Plans: Generate-and-Test Approach

- Request a sufficiently large number of plans ($k = \infty$)
- Enumerate all plans with increasing costs
  - Add loopless plans to the solution set
  - Discard plans with loops

Pros & Cons

- ✔ Straightforward approach
- ✔ Sound approach
- ✖ Performance strongly depends on the number of discarded plans
- ✖ Not complete when there are zero cost loops

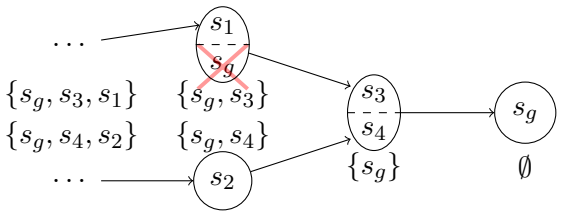


Loopless Plans: Symbolic Search Approach

- Use symbolic search for multiple plan
- Modify the plan reconstruction phase
  - Keep track of visited states
  - Ignore already visited states

Loopless Plans: Symbolic Search Approach

- Use symbolic search for multiple plan
- Modify the plan reconstruction phase
  - Keep track of visited states
  - Ignore already visited states



Pros & Cons

- ✔ Sound and complete approach
- ✔ Generates only relevant plans
- ✖ More complex
- ✖ Unnecessary overhead when few or no loopy plans exist

Finding Plans with Specific Properties

General Approach

- Request a sufficiently large number of plans ($k = \infty$)
- Enumerate all plans with increasing costs
- Optional: Ignore certain states during plan reconstruction

$\rightsquigarrow$ Stop when one or more plans with desired properties are found

Finding Plans with Specific Properties

General Approach

- Request a sufficiently large number of plans ($k = \infty$)
- Enumerate all plans with increasing costs
- Optional: Ignore certain states during plan reconstruction

$\rightsquigarrow$ Stop when one or more plans with desired properties are found

Application Examples

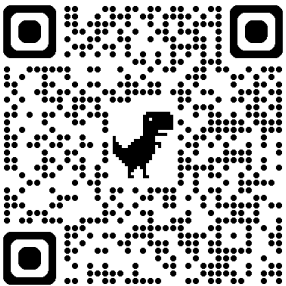
- Satisfying complex user preferences
- Finding HTN plans in over-approximated classical planning models
- Finding longest loopless plans/paths

🏆 Winner of the Combinatorial Reconfiguration Challenge in 2022 and 2023!

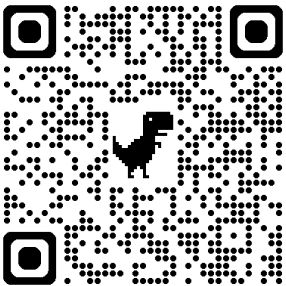
Outline

1. Introduction and motivation
2. Planning problems
   1. Diverse planning
   2. Top-k and top-quality planning
3. Planning methods
   1. ForbidIterative
   2. K*
   3. SymK
4. **Hands-on session**

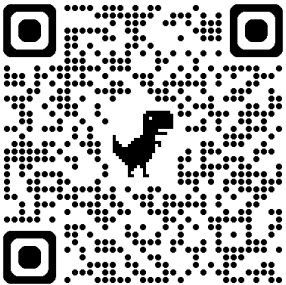
Planners: Online Available on Github



Github:
ForbidIterative



Github:
K*



Github:
SymK

Online Notebooks with Python Bindings

Notebook: ForbidIterative

Notebook: K*

Notebook: SymK

Note: If you want to use the latest versions of the planners with the best raw performance and the newest features, it might be worth to check the github repository and install the planners as standalone tools.