Principles of AI Planning

12. Planning with State-Dependent Action Costs

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Bernhard Nebel and Robert Mattmüller

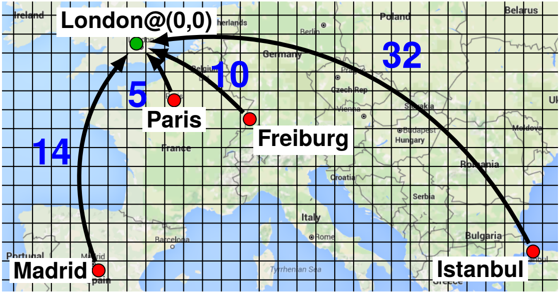
December 18th, 2017

Background

Motivation

- We now know the basics of classical planning.
- Where to go from here? Possible routes:
  - Algorithms: techniques orthogonal to heuristic search (partial-order reduction, symmetry reduction, decompositions, ...)
    $\leadsto$ later
  - Algorithms: techniques other than heuristic search (SAT/SMT planning, BDD-based symbolic planning, ...)
    $\leadsto$ beyond the scope of this course
  - Settings beyond classical planning (nondeterminism, partial observability, numeric planning, ...)
    $\leadsto$ later
  - A slight extension to the expressiveness of classical planning tasks
    $\leadsto$ this chapter

What are State-Dependent Action Costs?



Action costs: unit — constant — state-dependent

$$cost(fly(Madrid, London)) = 1, \quad cost(fly(Paris, London)) = 1,$$
$$cost(fly(Freiburg, London)) = 1, \quad cost(fly(Istanbul, London)) = 1.$$

Why Study State-Dependent Action Costs?

- In classical planning: actions have unit costs.
  - Each action *a* costs 1.
- Simple extension: actions have constant costs.
  - Each action *a* costs some $cost_a \in \mathbb{N}$.
  - Example: Flying between two cities costs amount proportional to distance.
  - Still easy to handle algorithmically, e. g. when computing *g* and *h* values.
- Further extension: actions have state-dependent costs.
  - Each action *a* has cost function $cost_a : S \to \mathbb{N}$.
  - Example: Flying to a destination city costs amount proportional to distance, depending on the current city.

Why Study State-Dependent Action Costs?

- Human perspective:
  - “natural”, “elegant”, and “higher-level”
  - modeler-friendly $\rightsquigarrow$ less error-prone?
- Machine perspective:
  - more structured $\rightsquigarrow$ exploit structure in algorithms?
  - fewer redundancies, exponentially more compact
- Language support:
  - numeric PDDL, PDDL 3
  - RDDL, MDPs (state-dependent rewards!)
- Applications:
  - modeling preferences and soft goals
  - application domains such as PSR

(Abbreviation: SDAC = state-dependent action costs)

Handling State-Dependent Action Costs

Good news:

- Computing g values in forward search still easy.
  (When expanding state s with action a, we know $cost_a(s)$.)

Challenge:

- But what about SDAC-aware h values
  (relaxation heuristics, abstraction heuristics)?
- Or can we simply compile SDAC away?

This chapter:

- Proposed answers to these challenges.

Handling State-Dependent Action Costs

Roadmap:

1. Look at compilations.
2. This leads to edge-valued multi-valued decision diagrams (EVMDDs) as data structure to represent cost functions.
3. Based on EVMDDs, formalize and discuss:
   - compilations
   - relaxation heuristics
   - abstraction heuristics

Definition

A SAS$^+$ planning task with state-dependent action costs or SDAC planning task is a tuple $\Pi = \langle V, I, O, \gamma, (cost_a)_{a \in O} \rangle$ where $\langle V, I, O, \gamma \rangle$ is a (regular) SAS$^+$ planning task with state set S and $cost_a : S \to \mathbb{N}$ is the cost function of a for all $a \in O$.

Assumption: For each $a \in O$, the set of variables occuring in the precondition of a is disjoint from the set of variables on which the cost function $cost_a$ depends.
(Question: Why is this assumption unproblematic?)

Definitions of plans etc. stay as before. A plan is optimal if it minimizes the sum of action costs from start to goal.

For the rest of this chapter, we consider the following running example.

Example (Household domain)

Actions:

$$\mathtt{vacuumFloor} = \langle \top, \mathtt{floorClean} \rangle$$
$$\mathtt{washDishes} = \langle \top, \mathtt{dishesClean} \rangle$$
$$\mathtt{doHousework} = \langle \top, \mathtt{floorClean} \wedge \mathtt{dishesClean} \rangle$$

Cost functions:

$$cost_{\mathtt{vacuumFloor}} = [\neg \mathtt{floorClean}] \cdot 2$$
$$cost_{\mathtt{washDishes}} = [\neg \mathtt{dishesClean}] \cdot (1 + 2 \cdot [\neg \mathtt{haveDishwasher}])$$
$$cost_{\mathtt{doHousework}} = cost_{\mathtt{vacuumFloor}} + cost_{\mathtt{washDishes}}$$

(Question: How much can applying action `washDishes` cost?)

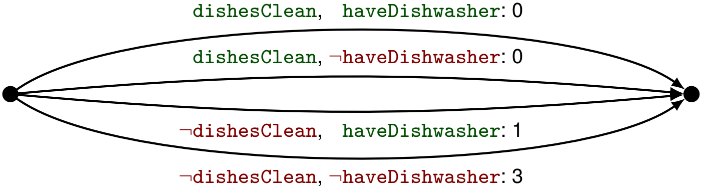
Different ways of compiling SDAC away:

- Compilation I: “Parallel Action Decomposition”
- Compilation II: “Purely Sequential Action Decomposition”
- Compilation III: “EVMDD-Based Action Decomposition” (combination of Compilations I and II)

State-Dependent Action Costs

Compilation I: “Parallel Action Decomposition”

Example



$$\mathtt{washDishes}(\ \mathtt{dC},\ \ \mathtt{hD}) = \langle\ \ \mathtt{dC} \wedge\ \ \mathtt{hD}, \mathtt{dC}\rangle, \quad \mathit{cost} = 0$$

$$\mathtt{washDishes}(\ \mathtt{dC}, \neg\mathtt{hD}) = \langle\ \ \mathtt{dC} \wedge \neg\mathtt{hD}, \mathtt{dC}\rangle, \quad \mathit{cost} = 0$$

$$\mathtt{washDishes}(\neg\mathtt{dC},\ \ \mathtt{hD}) = \langle\neg\mathtt{dC} \wedge\ \ \mathtt{hD}, \mathtt{dC}\rangle, \quad \mathit{cost} = 1$$

$$\mathtt{washDishes}(\neg\mathtt{dC}, \neg\mathtt{hD}) = \langle\neg\mathtt{dC} \wedge \neg\mathtt{hD}, \mathtt{dC}\rangle, \quad \mathit{cost} = 3$$

Compilation I

Transform each action into multiple actions:

- one for each partial state relevant to cost function
- add partial state to precondition
- use cost for partial state as constant cost

Properties:

✔ always possible

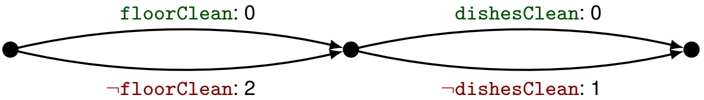
✗ exponential blow-up

Question: Exponential blow-up avoidable? $\rightsquigarrow$ Compilation II

Example

Assume we own a dishwasher:

$$cost_{\texttt{doHousework}} = 2 \cdot [\neg\texttt{floorClean}] + [\neg\texttt{dishesClean}]$$



$$\begin{aligned}
\texttt{doHousework}_1(\ \texttt{fC}) &= \langle\ \texttt{fC}, \texttt{fC}\rangle, & cost &= 0\\
\texttt{doHousework}_1(\neg\texttt{fC}) &= \langle\neg\texttt{fC}, \texttt{fC}\rangle, & cost &= 2\\
\texttt{doHousework}_2(\ \texttt{dC}) &= \langle\ \texttt{dC}, \texttt{dC}\rangle, & cost &= 0\\
\texttt{doHousework}_2(\neg\texttt{dC}) &= \langle\neg\texttt{dC}, \texttt{dC}\rangle, & cost &= 1
\end{aligned}$$

Compilation II

If costs are additively decomposable/separable:

- high-level actions ≈ macro actions
- decompose into sequential micro actions

Properties:

- ✔ only linear blow-up
- ✗ not always possible
- plan lengths not preserved

  E. g., in a state where $\neg\texttt{fC}$ and $\neg\texttt{dC}$ hold, an application of

  $$\texttt{doHousework}$$

  in the SDAC setting is replaced by an application of the action sequence

  $$\texttt{doHousework}_1(\neg\texttt{fC}), \texttt{doHousework}_2(\neg\texttt{dC})$$

  in the compiled setting.

Properties (ctd.):

- plan costs preserved
- blow-up in search space

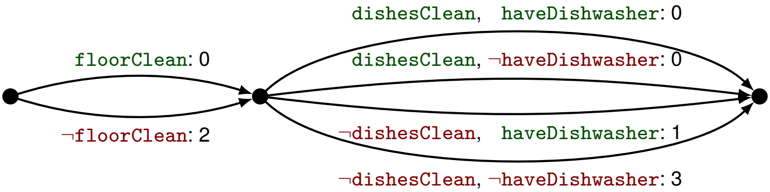
  E. g., in a state where $\neg\mathtt{fC}$ and $\neg\mathtt{dC}$ hold, should we apply $\mathtt{doHousework}_1(\neg\mathtt{fC})$ or $\mathtt{doHousework}_2(\neg\mathtt{dC})$ first?

  $\rightsquigarrow$ impose action ordering!
- attention: we should apply all partial effects at end! Otherwise, an effect of an earlier action in the compilation might affect the cost of a later action in the compilation.

Question: Can this always work (kind of)? $\rightsquigarrow$ Compilation III

Example

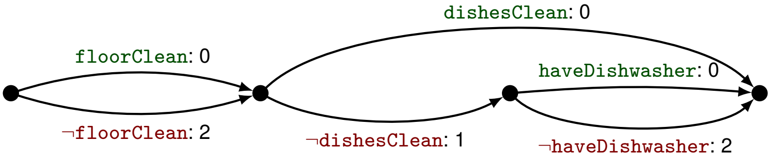
$$cost_{\texttt{doHousework}} = [\neg\texttt{floorClean}] \cdot 2 + [\neg\texttt{dishesClean}] \cdot (1 + 2 \cdot [\neg\texttt{haveDishwasher}])$$



Simplify right-hand part of diagram:

- Branch over single variable at a time.
- Exploit: `haveDishwasher` irrelevant if `dishesClean` is true.

Example (ctd.)



Later:

- Compiled actions
- Auxiliary variables to enforce action ordering

Compilation III

- exploit as much additive separability as possible
- multiply out variable domains where inevitable
- Technicalities:
  - fix variable ordering
  - perform Shannon and isomorphism reduction (cf. theory of BDDs)

Properties:

✓ always possible
- worst-case exponential blow-up, but as good as it gets
- as with Compilation II: plan lengths not preserved, plan costs preserved
- as with Compilation II: action ordering, all effects at end!

Compilation III provides optimal combination of sequential and parallel action decomposition, given fixed variable ordering.

Question: How to find such decompositions automatically?

Answer: Figure for Compilation III basically a reduced ordered edge-valued multi-valued decision diagram (EVMDD)!

[Lai et al., 1996; Ciardo and Siminiceanu, 2002]

EVMDDs

Edge-Valued Multi-Valued Decision Diagrams

EVMDDs:

- Decision diagrams for arithmetic functions
- Decision nodes with associated decision variables
- Edge weights: partial costs contributed by facts
- Size of EVMDD compact in many “typical”, well-behaved cases (Question: For example?)

Properties:

✔ satisfy all requirements for Compilation III,
even (almost) uniquely determined by them

✔ already have well-established theory and tool support

✔ detect and exhibit additive structure in arithmetic functions

EVMDDs

Edge-Valued Multi-Valued Decision Diagrams

Consequence:

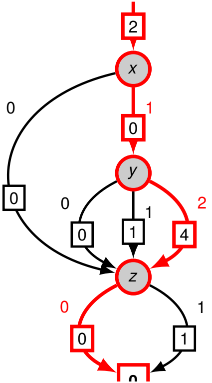
- represent cost functions as EVMDDs
- exploit additive structure exhibited by them
- draw on theory and tool support for EVMDDs

Two perspectives on EVMDDs:

- graphs specifying how to decompose action costs
- data structures encoding action costs
  (used independently from compilations)

Example (EVMDD Evaluation)

$cost_a = xy^2 + z + 2$ $\qquad \mathscr{D}_x = \mathscr{D}_z = \{0,1\},\ \mathscr{D}_y = \{0,1,2\}$



- Directed acyclic graph
- Dangling incoming edge
- Single terminal node **0**
- Decision nodes with:
  - decision variables
  - edge label
  - edge weights
- We see: z independent from rest, y only matters if $x \neq 0$.
- $s = \{x \mapsto 1,\ y \mapsto 2,\ z \mapsto 0\}$

Properties of EVMDDs:

- ✔ Existence for finitely many finite-domain variables
- ✔ Uniqueness/canonicity if reduced and ordered
- ✔ Basic arithmetic operations supported

(Lai et al., 1996; Ciardo and Siminiceanu, 2002)

Given arithmetic operator $\otimes \in \{+, -, \cdot, \dots\}$, EMVDDs $\mathcal{E}_1$, $\mathcal{E}_2$.
Compute EVMDD $\mathcal{E} = \mathcal{E}_1 \otimes \mathcal{E}_2$.

Implementation: procedure $\mathtt{apply}(\otimes, \mathcal{E}_1, \mathcal{E}_2)$:

- Base case: single-node EVMDDs encoding constants
- Inductive case: apply $\otimes$ recursively:
  - push down edge weights
  - recursively apply $\otimes$ to corresponding children
  - pull up excess edge weights from children

Time complexity [Lai et al., 1996]:

- additive operations: product of input EVMDD sizes
- in general: exponential

Compilation

EVMDD-Based Action Compilation

Idea: each edge in the EVMDD becomes a new micro action with constant cost corresponding to the edge constraint, precondition that we are currently at its start EVMDD node, and effect that we are currently at its target EVMDD node.

Example (EVMDD-based action compilation)

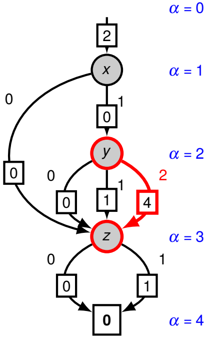
Let $a = \langle \chi, e \rangle$, $cost_a = xy^2 + z + 2$.

Auxiliary variables:

- One semaphore variable σ with $\mathscr{D}_\sigma = \{0, 1\}$ for entire planning task.
- One auxiliary variable $\alpha = \alpha_a$ with $\mathscr{D}_{\alpha_a} = \{0, 1, 2, 3, 4\}$ for action a.

Replace a by new auxiliary actions (similarly for other actions).

Example (EVMDD-based action compilation, ctd.)



$$
\begin{aligned}
a^{\chi} &= \langle \chi \wedge \sigma = 0 \wedge \alpha = 0, & \\
&\qquad \sigma := 1 \wedge \alpha := 1\rangle, & cost = 2 \\
a^{1,x=0} &= \langle \alpha = 1 \wedge x = 0,\ \alpha := 3\rangle, & cost = 0 \\
a^{1,x=1} &= \langle \alpha = 1 \wedge x = 1,\ \alpha := 2\rangle, & cost = 0 \\
a^{2,y=0} &= \langle \alpha = 2 \wedge y = 0,\ \alpha := 3\rangle, & cost = 0 \\
a^{2,y=1} &= \langle \alpha = 2 \wedge y = 1,\ \alpha := 3\rangle, & cost = 1 \\
a^{2,y=2} &= \langle \alpha = 2 \wedge y = 2,\ \alpha := 3\rangle, & cost = 4 \\
a^{3,z=0} &= \langle \alpha = 3 \wedge z = 0,\ \alpha := 4\rangle, & cost = 0 \\
a^{3,z=1} &= \langle \alpha = 3 \wedge z = 1,\ \alpha := 4\rangle, & cost = 1 \\
a^{e} &= \langle \alpha = 4,\ e \wedge \sigma := 0 \wedge \alpha := 0\rangle, & cost = 0
\end{aligned}
$$

EVMDD-Based Action Compilation

Definition (EVMDD-based action compilation)

Let $\Pi = \langle V, I, O, \gamma, (cost_a)_{a \in O} \rangle$ be an SDAC planning task, and for each action $a \in O$, let $\mathcal{E}_a$ be an EVMDD that encodes the cost function $cost_a$.

Let $EAC(a)$ be the set of actions created from a using $\mathcal{E}_a$ similar to the previous example. Then the EVMDD-based action compilation of Π using $\mathcal{E}_a$, $a \in O$, is the task $\Pi' = EAC(\Pi) = \langle V', I', O', \gamma' \rangle$, where

- $V' = V \cup \{\sigma\} \cup \{\alpha_a \mid a \in O\}$,
- $I' = I \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$,
- $O' = \bigcup_{a \in O} EAC(a)$, and
- $\gamma' = \gamma \wedge (\sigma = 0) \wedge \bigwedge_{a \in O} (\alpha_a = 0)$.

EVMDD-Based Action Compilation

Let Π be an SDAC task and $\Pi' = \mathit{EAC}(\Pi)$ its EVMDD-based action compilation (for appropriate EVMDDs $\mathcal{E}_a$).

Proposition

Π' has only state-independent costs.

Proof.

By construction. $\square$

Proposition

The size $\|\Pi'\|$ is in the order $O(\|\Pi\| \cdot \max_{a \in O} \|\mathcal{E}_a\|)$, i. e. polynomial in the size of Π and the largest used EVMDD.

Proof.

By construction. $\square$

EVMDD-Based Action Compilation

Let Π be an SDAC task and $\Pi' = EAC(\Pi)$ its EVMDD-based action compilation (for appropriate EVMDDs $\mathcal{E}_a$).

Proposition

Π and Π' admit the same plans (up to replacement of actions by action sequences). Optimal plan costs are preserved.

Proof.

Let $\pi = a_1, \dots, a_n$ be a plan for Π, and let $s_0, \dots, s_n$ be the corresponding state sequence such that a_i is applicable in s_{i-1} and leads to s_i for all $i = 1, \dots, n$.

For each $i = 1, \dots, n$, let $\mathcal{E}_{a_i}$ be the EVMDD used to compile a_i. State s_{i-1} determines a unique path through the EVMDD $\mathcal{E}_{a_i}$, which uniquely corresponds to an action sequence $a_i^0, \dots, a_i^{k_i}$ (for some $k_i \in \mathbb{N}$; including a_i^χ and a_i^e).

EVMDD-Based Action Compilation

Proof (ctd.)

By construction, $cost(a_i^0) + \dots + cost(a_i^{k_i}) = cost_{a_i}(s_{i-1})$. Moreover, the sequence $a_i^0, \dots, a_i^{k_i}$ is applicable in $s_{i-1} \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$ and leads to $s_i \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$.

Therefore, by induction, $\pi' = a_1^0, \dots, a_1^{k_1}, \dots, a_n^0, \dots, a_n^{k_n}$ is applicable in $s_0 \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$ (and leads to a goal state). Moreover,
$cost(\pi') = cost(a_1^0) + \dots + cost(a_1^{k_1}) + \dots + cost(a_n^0) + \dots + cost(a_n^{k_n}) = cost_{a_1}(s_0) + \dots + cost_{a_n}(s_{n-1}) = cost(\pi)$.

Still to show: Π' admits no other plans. It suffices to see that the semaphore σ prohibits interleaving more than one EVMDD evaluation, and that each α_a makes sure that the EVMDD for a is traversed in the unique correct order. □

EVMDD-Based Action Compilation

Example

Let $\Pi = \langle V, I, O, \gamma \rangle$ with $V = \{x, y, z, u\}$, $\mathscr{D}_x = \mathscr{D}_z = \{0, 1\}$, $\mathscr{D}_y = \mathscr{D}_u = \{0, 1, 2\}$, $I = \{x \mapsto 1, y \mapsto 2, z \mapsto 0, u \mapsto 0\}$, $O = \{a, b\}$, and $\gamma = (u = 2)$ with

$$a = \langle u = 0, u := 1 \rangle, \qquad cost_a = xy^2 + z + 2,$$
$$b = \langle u = 1, u := 2 \rangle, \qquad cost_b = z + 1.$$

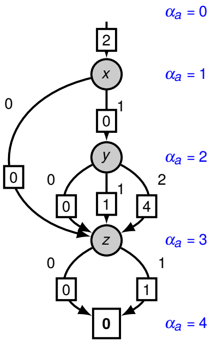
Optimal plan for Π:

$$\pi = a, b \text{ with } cost(\pi) = 6 + 1 = 7.$$

EVMDD-Based Action Compilation

Example (Ctd.)

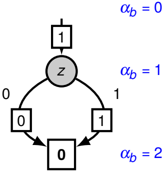
Compilation of a:



$$
\begin{aligned}
a^{\chi} &= \langle u = 0 \land \sigma = 0 \land \alpha_a = 0, \\
&\qquad \sigma := 1 \land \alpha_a := 1\rangle, && cost = 2 \\
a^{1,x=0} &= \langle \alpha_a = 1 \land x = 0,\ \alpha_a := 3\rangle, && cost = 0 \\
a^{1,x=1} &= \langle \alpha_a = 1 \land x = 1,\ \alpha_a := 2\rangle, && cost = 0 \\
a^{2,y=0} &= \langle \alpha_a = 2 \land y = 0,\ \alpha_a := 3\rangle, && cost = 0 \\
a^{2,y=1} &= \langle \alpha_a = 2 \land y = 1,\ \alpha_a := 3\rangle, && cost = 1 \\
a^{2,y=2} &= \langle \alpha_a = 2 \land y = 2,\ \alpha_a := 3\rangle, && cost = 4 \\
a^{3,z=0} &= \langle \alpha_a = 3 \land z = 0,\ \alpha_a := 4\rangle, && cost = 0 \\
a^{3,z=1} &= \langle \alpha_a = 3 \land z = 1,\ \alpha_a := 4\rangle, && cost = 1 \\
a^{e} &= \langle \alpha_a = 4,\ u := 1 \land \sigma := 0 \land \alpha_a := 0\rangle, && cost = 0
\end{aligned}
$$

Example (Ctd.)

Compilation of b:



$$
\begin{aligned}
b^{\chi} &= \langle u = 1 \wedge \sigma = 0 \wedge \alpha_b = 0, \\
&\qquad \sigma := 1 \wedge \alpha_b := 1 \rangle, && \mathit{cost} = 1 \\
b^{1,z=0} &= \langle \alpha_b = 1 \wedge z = 0,\ \alpha_b := 2 \rangle, && \mathit{cost} = 0 \\
b^{1,z=1} &= \langle \alpha_b = 1 \wedge z = 1,\ \alpha_b := 2 \rangle, && \mathit{cost} = 1 \\
b^{e} &= \langle \alpha_b = 2,\ u := 2 \wedge \sigma := 0 \wedge \alpha_b := 0 \rangle, && \mathit{cost} = 0
\end{aligned}
$$

Optimal plan for Π' (with $\mathit{cost}(\pi') = 6 + 1 = 7 = \mathit{cost}(\pi)$):

$$
\pi' = \underbrace{a^{\chi}, a^{1,x=1}, a^{2,y=2}, a^{3,z=0}, a^{e}}_{\mathit{cost}=2+0+4+0+0=6}, \underbrace{b^{\chi}, b^{1,z=0}, b^{e}}_{\mathit{cost}=1+0+0=1}.
$$

- Okay. We can compile SDAC away somewhat efficiently. Is this the end of the story?
- No! Why not?
  - Tighter integration of SDAC into planning process might be beneficial.
  - Analysis of heuristics for SDAC might improve our understanding.
- Consequence: Let's study heuristics for SDAC in uncompiled setting.

Relaxations

Relaxation Heuristics

We know: Delete-relaxation heuristics informative in classical planning.

Question: Are they also informative in SDAC planning?

Relaxation Heuristics

- Assume we want to compute the additive heuristic h^{add} in a task with state-dependent action costs.
- But what does an action a cost in a relaxed state s^+?
- And how to compute that cost?

Relaxed SAS$^+$ Tasks

Delete relaxation in SAS$^+$ tasks works as follows:

- Operators are already in effect normal form.
- We do not need to impose a positive normal form, because all conditions are conjunctions of facts, and facts are just variable-value pairs and hence always positive.
- Hence $a^+ = a$ for any operator a, and $\Pi^+ = \Pi$.
- For simplicity, we identify relaxed states s^+ with their on-sets $on(s^+)$.
- Then, a relaxed state s^+ is a set of facts (v, d) with $v \in V$ and $d \in \mathscr{D}_v$ including at least one fact (v, d) for each $v \in V$ (but possibly more than one, which is what makes it a relaxed state).

Relaxed SAS$^+$ Tasks

- A relaxed operator a is applicable in a relaxed state s^+ if all precondition facts of a are contained in s^+.
- Relaxed states accumulate facts reached so far.
- Applying a relaxed operator a to a relaxed state s^+ adds to s^+ those facts made true by a.

Example

Relaxed operator $a^+ = \langle x = 2, y := 1 \wedge z := 0 \rangle$ is applicable in relaxed state $s^+ = \{(x,0),(x,2),(y,0),(z,1)\}$, because precondition $(x,2) \in s^+$, and leads to successor $(s^+)' = s^+ \cup \{(y,1),(z,0)\}$.

Relaxed plans, dominance, monotonicity etc. as before. The above definition generalizes the one for propositional tasks.

Action Costs in Relaxed States

Example

Assume s^+ is the relaxed state with

$$s^+ = \{(x,0),(x,1),(y,1),(y,2),(z,0)\}.$$

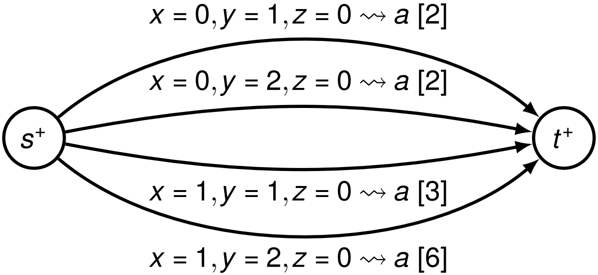
What should action a with $cost_a = xy^2 + z + 2$ cost in s^+?

$$s^+ \xrightarrow{a\ [?]} t^+$$

Action Costs in Relaxed States

Idea: We should assume the cheapest way of applying o^+ in s^+ to guarantee admissibility of h^+.
(Allow at least the behavior of the unrelaxed setting at no higher cost.)

Example



Action Costs in Relaxed States

Idea: We should assume the cheapest way of applying o^+ in s^+ to guarantee admissibility of h^+.
(Allow at least the behavior of the unrelaxed setting at no higher cost.)

Example

$$s^+ \xrightarrow{a^+ \,[2]} t^+$$

Action Costs in Relaxed States

Definition

Let V be a set of FDR variables, $s : V \to \bigcup_{v \in V} \mathscr{D}_v$ an unrelaxed state over V, and $s^+ \subseteq \{(v,d) \mid v \in V, d \in \mathscr{D}_v\}$ a relaxed state over V. We call s consistent with s^+ if $\{(v, s(v)) \mid v \in V\} \subseteq s^+$.

Definition

Let $a \in O$ be an action with cost function $cost_a$, and s^+ a relaxed state. Then the relaxed cost of a in s^+ is defined as

$$cost_a(s^+) = \min_{s \in S \text{ consistent with } s^+} cost_a(s).$$

(Question: How many states s are consistent with s^+?)

Action Costs in Relaxed States

Problem with this definition: There are generally exponentially many states s consistent with s^+ to minimize over.

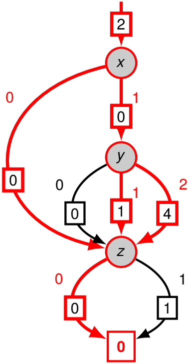
Central question: Can we still do this minimization efficiently?

Answer: Yes, at least efficiently in the size of an EVMDD encoding $cost_a$.

Cost Computation for Relaxed States

Example

Relaxed state $s^+ = \{(x,0),(x,1),(y,1),(y,2),(z,0)\}$.

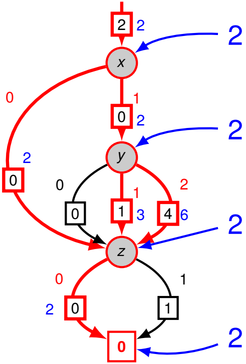


- Computing $cost_a(s^+)$ = minimizing over $cost_a(s)$ for all s consistent with s^+ = minimizing over all start-end-paths in EVMDD following only edges consistent with s^+.
- Observation: Minimization over exponentially many paths can be replaced by top-sort traversal of EVMDD, minimizing over incoming arcs consistent with s^+ at all nodes!

Cost Computation for Relaxed States

Example

Relaxed state $s^+ = \{(x,0),(x,1),(y,1),(y,2),(z,0)\}$.



- $cost_a(s^+) = 2$
- Cost-minimizing s consistent with s^+: $s(x) = s(z) = 0$, $s(y) \in \{1,2\}$.

Cost Computation for Relaxed States

Theorem

A top-sort traversal of the EVMDD for $cost_a$, adding edge weights and minimizing over incoming arcs consistent with s^+ at all nodes, computes $cost_a(s^+)$ and takes time in the order of the size of the EVMDD.

Proof.

Homework? □

The following definition is equivalent to the RPG-based one.

Definition (Classical additive heuristic h^{add})

$$h^{add}(s) = h^{add}_s(GoalFacts)$$

$$h^{add}_s(Facts) = \sum_{fact \in Facts} h^{add}_s(fact)$$

$$h^{add}_s(fact) = \begin{cases} 0 & \text{if } fact \in s \\ \min_{\text{achiever } a \text{ of } fact} [h^{add}_s(pre(a)) + cost_a] & \text{otherwise} \end{cases}$$

Question: How to generalize h^{add} to SDAC?

Example

$$a = \langle \top,\ x = 1 \rangle \qquad cost_a = 2 - 2y$$

$$b = \langle \top,\ y = 1 \rangle \qquad cost_b = 1$$

$$s = \{x \mapsto 0, y \mapsto 0\}$$

$$h_s^{add}(y = 1) = 1$$

$$h_s^{add}(x = 1) = ?$$

- $00 \xrightarrow{a\,:\,2} \boxed{10}$
- $00 \xrightarrow{b\,:\,1} 01 \xrightarrow{a\,:\,0} \boxed{11}$ $\Rightarrow$ cheaper!

(Here, we need the assumption that no variable occurs both in the cost function and the precondition of the same action):

Definition (Additive heuristic h^{add} for SDAC)

$$h^{add}_s(fact) = \begin{cases} 0 & \text{if } fact \in s \\ \min\limits_{\text{achiever } a \text{ of } fact} [h^{add}_s(pre(a)) + cost_a] & \text{otherwise} \end{cases}$$

$$h^{add}_s(fact) = \begin{cases} 0 \\ \min\limits_{\text{achiever } a \text{ of } fa} \end{cases}$$

$$Cost^s_a = \min_{\hat{s} \in S_a} [cost_a(\hat{s}) + h^{add}_s(\hat{s})]$$

S_a: set of partial states over variables in cost function

$|S_a|$ exponential in number of variables in cost function

Relaxations with SDAC

Theorem

Let Π be an SDAC planning task, let Π' be an EVMDD-based action compilation of Π, and let s be a state of Π. Then the classical h^{add} heuristic in Π' gives the same value for $s \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$ as the generalization of h^{add} to SDAC tasks defined above gives for s in Π. $\square$

Computing h^{add} for SDAC:

- Option 1: Compute classical h^{add} on compiled task.
- Option 2: Compute $Cost_a^s$ directly. How?
  - Plug EVMDDs as subgraphs into RPG
  - $\rightsquigarrow$ efficient computation of h^{add}

RPG Compilation

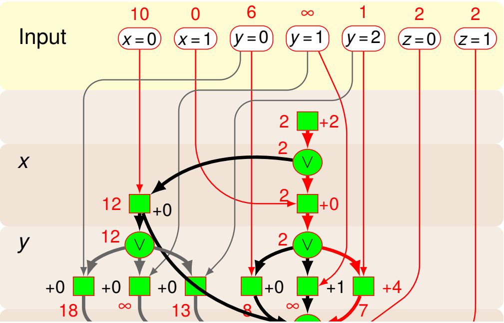
Remark: We can use EVMDDs to compute C_s^a and hence the generalized additive heuristic directly, by embedding them into the relaxed planning task.

We just briefly show the example, without going into too much detail.

Idea: Augment EVMDD with input nodes representing h^{add} values from the previous RPG layer.

- Use augmented diagrams as RPG subgraphs.
- Allows efficient computation of h^{add}.

Option 2: RPG Compilation Option 2: Computing $Cost_a^s$



Evaluate nodes:

- $cost_a = xy^2 + z + 2$
- variable nodes become $\vee$-nodes
- weights become $\wedge$-nodes
- Augment with input nodes
- Ensure complete evaluation
- Insert h^{add} values
- $\wedge$: $\sum$(parents) + weight
- $\vee$: min(parents)
- $Cost_a^s$ =

- Use above construction as subgraph of RPG in each layer, for each action (as operator subgraphs).
- Add AND nodes conjoining these subgraphs with operator precondition graphs.
- Link EVMDD outputs to next proposition layer.

Theorem

Let Π be an SDAC planning task. Then the classical additive RPG evaluation of the RPG constructed using EVMDDs as above computes the generalized additive heuristic h^{add} defined before. □

Abstractions

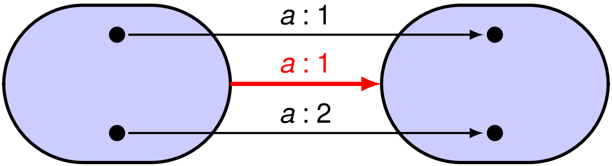
Abstraction Heuristics for SDAC

Question: Why consider abstraction heuristics?

Answer:

- admissibility
- $\rightsquigarrow$ optimality

Abstraction Heuristics for SDAC



Question: What are the abstract action costs?

Answer: For admissibility, abstract cost of a should be

$$cost_a(s^{\text{abs}}) = \min_{\substack{\text{concrete state } s \\ \text{abstracted to } s^{\text{abs}}}} cost_a(s).$$

Problem: exponentially many states in minimization

Aim: Compute $cost_a(s^{\text{abs}})$ efficiently
(given EVMDD for $cost_a(s)$).

Cartesian Abstractions

We will see: possible if the abstraction is Cartesian or coarser.

(Includes projections and domain abstractions.)

Definition (Cartesian abstraction)

A set of states s^{abs} is Cartesian if it is of the form

$$D_1 \times \cdots \times D_n,$$

where $D_i \subseteq \mathscr{D}_i$ for all $i = 1, \ldots, n$.

An abstraction is Cartesian if all abstract states are Cartesian sets.

[Seipp and Helmert, 2013]

Intuition: Variables are abstracted independently.

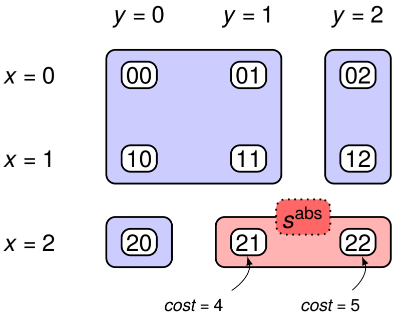
$\rightsquigarrow$ exploit independence when computing abstract costs!

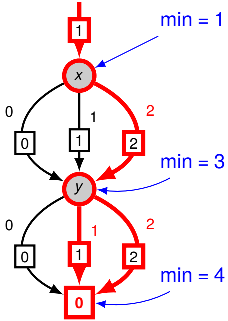
Example (Cartesian abstraction)

Cartesian abstraction over x, y

Cost $x + y + 1$

(edges consistent with s^{abs})





Cartesian Abstractions

Why does the topsort EVMDD traversal (cheapest path computation) correctly compute $cost_a(s^{abs})$?

Short answer: The exact same thing as with relaxed states, because relaxed states are Cartesian sets!

Longer answer:

1. For each Cartesian state s^{abs} and each variable v, each value $d \in \mathscr{D}_v$ is either consistent with s^{abs} or not.
2. This implies: at all decision nodes associated with variable v, some outgoing edges are enabled, others are disabled.

   This is independent from all other decision nodes.
3. This allows local minimizations over linearly many edges instead of global minimization over exponentially many paths in the EVMDD when computing minimum costs.

$\rightsquigarrow$ polynomial in EVMDD size!

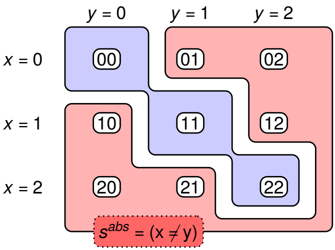
If abstraction not Cartesian: two variables can be

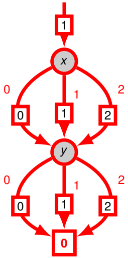
- independent in cost function ($\rightsquigarrow$ compact EVMDD), but
- dependent in abstraction.

$\rightsquigarrow$ cannot consider independent parts of EVMDD separately.

Example (Non-Cartesian abstraction)

$cost : x + y + 1$, $cost(s^{abs}) = 2$, local minim.: 1 $\rightsquigarrow$ underestimate!





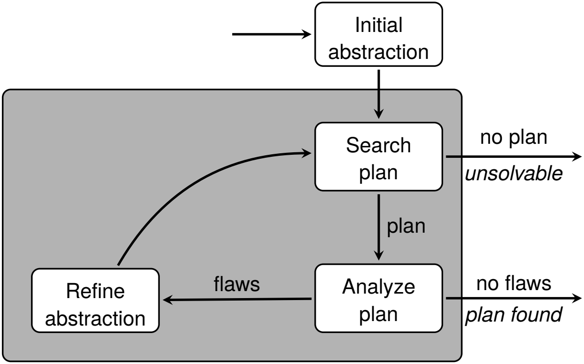
Background
Compilation
Relaxations
Abstractions
Cartesian Abstractions
CEGAR
Summary
References

Counterexample-Guided Abstraction Refinement

Wanted: principled way of computing Cartesian abstractions.

⇝ Counterexample-Guided Abstraction Refinement (CEGAR)

[Clarke et al., 2000] [Seipp and Helmert, 2013]



CEGAR and Cartesian Abstractions

Assume the following:

- Initial abstraction is one-state abstraction with single abstract state $\mathscr{D}_1 \times \cdots \times \mathscr{D}_n$.
  $\rightsquigarrow$ Cartesian abstraction
- Each refinement step takes one abstract state $s^{\text{abs}} = D_1 \times \cdots \times D_n$, one variable v_i, and splits s^{abs} into
  - $D_1 \times \cdots \times D_{i-1} \times D_i' \times D_{i+1} \times \cdots \times D_n$
  - $D_1 \times \cdots \times D_{i-1} \times D_i'' \times D_{i+1} \times \cdots \times D_n$

  such that $D_i' \cap D_i'' = \emptyset$ and $D_i' \cup D_i'' = D_i$.
  $\rightsquigarrow$ still a Cartesian abstraction

So, inductively:

- Initial abstraction is Cartesian.
- Each refinement step preserves being Cartesian.
- $\rightsquigarrow$ All generated abstractions are Cartesian.

Some questions:

- Q: When to split abstract states?
  A: When first flaw is identified. (Details below.)
- Q: How to split abstract states?
  A: So as to resolve that flaw. (Details below.)

Some questions:

- Q: How long to stay in refinement loop?
  A: Until one of the following termination criteria is met:
  - No abstract plan exists.
    $\rightsquigarrow$ Terminate with result “unsolvable”.
  - Abstract plan π is concretizable (= has no flaw).
    $\rightsquigarrow$ Return π as concrete plan.
  - Available resources (time, memory, abstraction size bound, . . .) exhausted.
    $\rightsquigarrow$ Use current abstraction as basis for abstraction heuristic for concrete planning task (i. e., compute abstract goal distances, store in lookup table, . . .).

Example (one package, one truck)

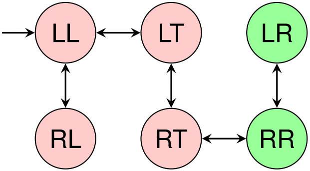
Consider the following FDR planning task $\langle V, I, O, \gamma \rangle$:

- $V = \{t, p\}$ with
  - $\mathscr{D}_t = \{L, R\}$
  - $\mathscr{D}_p = \{L, T, R\}$
- $I = \{t \mapsto L, p \mapsto L\}$
- $O = \{\mathit{pick\text{-}in}_i \mid i \in \{L, R\}\}$
  $\cup \{\mathit{drop\text{-}in}_i \mid i \in \{L, R\}\}$
  $\cup \{\mathit{move}_{i,j} \mid i, j \in \{L, R\}, i \neq j\}$, where
  - $\mathit{pick\text{-}in}_i = \langle t = i \wedge p = i, p := T \rangle$
  - $\mathit{drop\text{-}in}_i = \langle t = i \wedge p = T, p := i \rangle$
  - $\mathit{move}_{i,j} = \langle t = i, t := j \rangle$
- $\gamma = (p = R)$.

CEGAR by Example

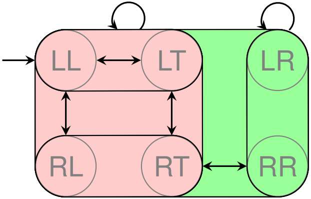
Example (Ctd.)

Before we look at CEGAR applied to this task, here is the concrete transition system (just for reference):



Example (Ctd.)

Refinement step 0 (initial abstraction): Refinement step 1:
Refinement step 2: Refinement step 3: Refinement step 4:



Abstract plan:

$\pi_0 = \langle\rangle \pi_1 = \langle \textit{drop-in}_R \rangle \pi_2 = \langle \textit{move}_{L,R}, \textit{drop-in}_R \rangle \pi_3 = \langle \textit{move}_{L,R}, \textit{drop-in}_R \rangle \pi_4 = \langle \textit{pick-in}_L, \textit{move}_{L,R}, \textit{drop-in}_R \rangle$

Flaw: $s_0 = LL$ is not a goal state.

CEGAR: Flaws

CEGAR for unit-cost tasks. Three kinds of flaws:

- Abstract plan works in concrete transition system, but ends in non-goal state.
  (Step 0 in example.)
- Some step of abstract plan fails in concrete transition system, because operator precondition is violated.
  (Steps 1 and 2 in example.)
- Concrete and abstract paths diverge at some point, because abstract transition system is nondeterministic.
  (Step 3 in example.)

CEGAR: Flaw Resolution

Flaw 1: Abstract plan terminates in concrete non-goal state.

Resolution: Split abstraction of last state s_n of concrete trace into (a) part containing s_n, but containing no concrete goal state, and (b) rest.

CEGAR: Flaw Resolution

Flaw 2: Abstract plan fails because some operator precondition is violated.

Resolution: Split abstraction of state s_{i-1} of concrete trace, where operator precondition χ is violated, into (a) part containing s_{i-1}, but no concrete state in which precondition χ is satisfied, and (b) rest.

CEGAR: Flaw Resolution

Flaw 3: Concrete and abstract paths diverge.

Resolution: Split abstraction of state s_{i-1} of concrete trace, after which paths diverge when applying operator o, into (a) part containing s_{i-1} where applying o always leads to the "wrong" abstract successor state, and (b), rest.

CEGAR: Cost-Mismatch Flaws

Remark: In tasks with state-dependent action costs, there is a fourth type of flaws, so-called cost-mismatch flaws.

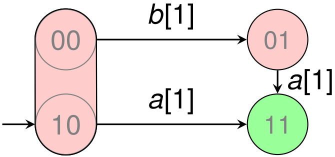
Flaw 4: Action is more costly in concrete state than in abstract state.

Resolution: Split abstraction of violating concrete state into two parts that differ on the value of a variable that is relevant to the cost function of the operator in question, such that we have different cost values in the two parts.

Example (Cost-mismatch flaw)

$$a = \langle \top, \quad x \wedge y \rangle, \quad cost_a = 2x + 1 \qquad s_0 = 10$$

$$b = \langle \top, \neg x \wedge y \rangle, \quad cost_b = 1 \qquad s_\star = x \wedge y$$



- Optimal abstract plan: $\langle a \rangle$ (abstract cost 1)
- This is also a concrete plan (concrete cost $3 \neq 1$)
  $\rightsquigarrow$ split $\{0, 1\} \times \{0\}$
- Cf. optimal concrete plan: $\langle b, a \rangle$ (concr. and abstr. cost 2)

Summary

Summary:

- State-dependent actions costs practically relevant.
- EVMDDs exhibit and exploit structure in cost functions.
- Graph-based representations of arithmetic functions.
- Edge values express partial cost contributed by facts.
- Size of EVMDD is compact in many “typical” cases.
- Can be used to compile tasks with state-dependent costs to tasks with state-independent costs.
- Alternatively, can be embedded into the RPG to compute forward-cost heuristics directly.
- For h^{add}, both approaches give the same heuristic values.
- Abstraction heuristics can also be generalized to state-dependent action costs.

UNI FREIBURG

Future Work and Work in Progress:

- Investigation of other delete-relaxation heuristics for tasks with state-dependent action costs.
- Investigation of static and dynamic EVMDD variable orders.
- Application to cost partitioning, to planning with preferences, ...
- Better integration of SDAC in PDDL.
- Tool support.
- Benchmarks.

References

Ciardo and Siminiceanu, **Using edge-valued decision diagrams for symbolic generation of shortest paths**, in Proc. 4th Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD 2002), pp. 256–273, 2002.

Geißer, Keller, and Mattmüller, **Delete relaxations for planning with state-dependent action costs**, in Proc. 24th Intl. Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 1573–1579, 2015.

Geißer, Keller, and Mattmüller, **Abstractions for planning with state-dependent action costs**, in Proc. 26th Intl. Conference on Automated Planning and Scheduling (ICAPS 2016), pp. 140–148, 2016.