

Principles of AI Planning

12. Planning with State-Dependent Action Costs

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Bernhard Nebel and Robert Mattmüller

December 16th, 2016



Background

Background

- State-Dependent
- Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

- 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, ...)
 ↔ later
 - Algorithms: techniques other than heuristic search (SAT/SMT planning, ...)
 ↔ beyond the scope of this course
 - Settings beyond classical planning (**nondeterminism**, partial observability, numeric planning, ...)
 ↔ later
 - A slight extension to the expressiveness of classical planning tasks
 ↔ **this chapter**

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

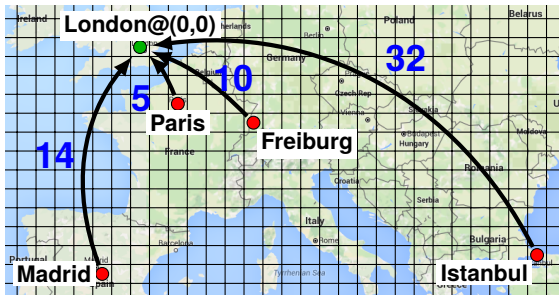
Abstractions

Practice

Summary

References

What are State-Dependent Action Costs?



Action costs:

unit

constant

state-dependent

$$\begin{aligned} \text{cost}(\text{fly}(\text{Madrid}, \text{London})) &= 1, & \text{cost}(\text{fly}(\text{Paris}, \text{London})) &= 1, \\ \text{cost}(\text{fly}(\text{Freiburg}, \text{London})) &= 1, & \text{cost}(\text{fly}(\text{Istanbul}, \text{London})) &= 1. \end{aligned}$$

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** $cost_a : S \rightarrow \mathbb{N}$.
 - Example: Flying to a destination city costs amount proportional to distance, depending on the current city.

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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)

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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.

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References



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 $cost_a : S \rightarrow \mathbb{N}$ is the **cost function** of a for all $a \in O$.

Assumption: For each $a \in O$, the set of variables occurring 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.

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

Example (Household domain)

Actions:

`vacuumFloor = $\langle \top, \text{floorClean} \rangle$`

`washDishes = $\langle \top, \text{dishesClean} \rangle$`

`doHousework = $\langle \top, \text{floorClean} \wedge \text{dishesClean} \rangle$`

Cost functions:

$cost_{\text{vacuumFloor}} = [\neg \text{floorClean}] \cdot 2$

$cost_{\text{washDishes}} = [\neg \text{dishesClean}] \cdot (1 + 2 \cdot [\neg \text{haveDishwasher}])$

$cost_{\text{doHousework}} = cost_{\text{vacuumFloor}} + cost_{\text{washDishes}}$

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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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)

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

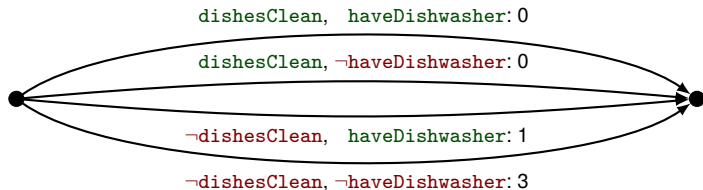
Practice

Summary

References



Example



Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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

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

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

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

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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

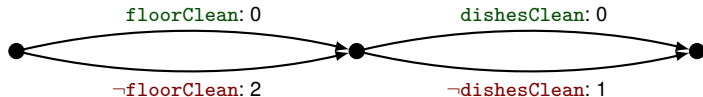
References



Example

Assume we own a dishwasher:

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



$$doHousework_1(\text{fC}) = \langle \text{fC}, \text{fC} \rangle, \quad cost = 0$$

$$doHousework_1(\neg \text{fC}) = \langle \neg \text{fC}, \text{fC} \rangle, \quad cost = 2$$

$$doHousework_2(\text{dC}) = \langle \text{dC}, \text{dC} \rangle, \quad cost = 0$$

$$doHousework_2(\neg \text{dC}) = \langle \neg \text{dC}, \text{dC} \rangle, \quad cost = 1$$

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References



Compilation II

If costs are **additively decomposable**:

- high-level actions \approx **macro actions**
- decompose into **sequential micro actions**

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References



Properties:

- ✓ linear blow-up
- ✗ not always possible
- plan lengths not preserved

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

`doHousework`

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

`doHousework1($\neg fC$), doHousework2($\neg dC$)`

in the compiled setting.

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

Properties (ctd.):

- plan costs preserved
- blow-up in search space

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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

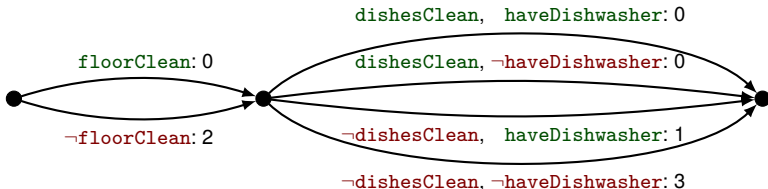
References



Example

$$\text{cost}_{\text{doHousework}} = [\neg \text{floorClean}] \cdot 2 +$$

$$[\neg \text{dishesClean}] \cdot (1 + 2 \cdot [\neg \text{haveDishwasher}])$$



Simplify right-hand part of diagram:

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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

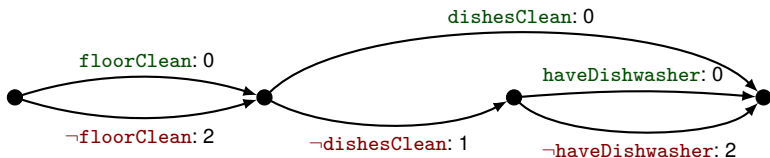
Practice

Summary

References



Example (ctd.)



Later:

- Compiled actions
- Auxiliary variables to enforce action ordering

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

Compilation III

- exploit as much **additive decomposability** 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!

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References



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]

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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**)

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

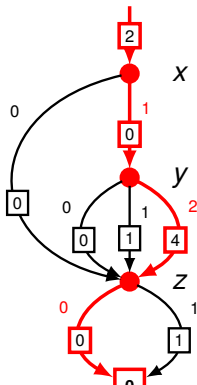
Summary

References

Example (EVMDD Evaluation)

$$cost_a = xy^2 + z + 2$$

$$\mathcal{D}_x = \mathcal{D}_z = \{0, 1\}, \quad \mathcal{D}_y = \{0, 1, 2\}$$



- 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 $x \neq 0$.
- $s = \{x \mapsto 1, y \mapsto 2, z \mapsto 0\}$

Background

State-Dependent
Action CostsEdge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References

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)

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References



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

Implementation: procedure `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

Background

State-Dependent
Action Costs

Edge-Valued
Multi-Valued
Decision Diagrams

Compilation

Relaxations

Abstractions

Practice

Summary

References



Compilation

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References



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

Example (EVMD-based action compilation)

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

Auxiliary variables:

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

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

Background

Compilation

Relaxations

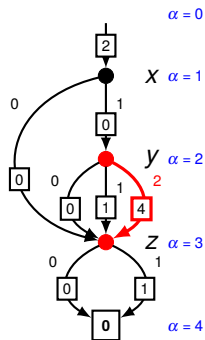
Abstractions

Practice

Summary

References

Example (EVMDD-based action compilation, ctd.)



$$a^\chi = \langle \chi \wedge \sigma = 0 \wedge \alpha = 0, \\ \sigma := 1 \wedge \alpha := 1 \rangle,$$

$cost = 2$

$$a^{1,x=0} = \langle \alpha = 1 \wedge x = 0, \alpha := 3 \rangle,$$

$cost = 0$

$$a^{1,x=1} = \langle \alpha = 1 \wedge x = 1, \alpha := 2 \rangle,$$

$cost = 0$

$$a^{2,y=0} = \langle \alpha = 2 \wedge y = 0, \alpha := 3 \rangle,$$

$cost = 0$

$$a^{2,y=1} = \langle \alpha = 2 \wedge y = 1, \alpha := 3 \rangle,$$

$cost = 1$

$$a^{2,y=2} = \langle \alpha = 2 \wedge y = 2, \alpha := 3 \rangle,$$

$cost = 4$

$$a^{3,z=0} = \langle \alpha = 3 \wedge z = 0, \alpha := 4 \rangle,$$

$cost = 0$

$$a^{3,z=1} = \langle \alpha = 3 \wedge z = 1, \alpha := 4 \rangle,$$

$cost = 1$

$$a^e = \langle \alpha = 4, e \wedge \sigma := 0 \wedge \alpha := 0 \rangle,$$

$cost = 0$

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References



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
- $\gamma' = \gamma \wedge (\sigma = 0) \wedge \bigwedge_{a \in O} (\alpha_a = 0)$.

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References

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

Proposition

Π' has only state-independent costs.

Proof.

By construction. □

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. □

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References

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^x and a_i^e).

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References



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. □

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References

Example

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

$$\begin{aligned} a &= \langle u = 0, u := 1 \rangle, & \text{cost}_a &= xy^2 + z + 2, \\ b &= \langle u = 1, u := 2 \rangle, & \text{cost}_b &= z + 1. \end{aligned}$$

Optimal plan for Π :

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

Background

Compilation

Relaxations

Abstractions

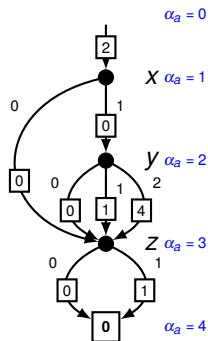
Practice

Summary

References

Example (Ctd.)

Compilation of a :



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

Background

Compilation

Relaxations

Abstractions

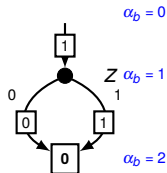
Practice

Summary

References

Example (Ctd.)

Compilation of b :



$$b^{\chi} = \langle u = 1 \wedge \sigma = 0 \wedge \alpha_b = 0, \\ \sigma := 1 \wedge \alpha_b := 1 \rangle, \quad \text{cost} = 1$$

$$b^{1,z=0} = \langle \alpha_b = 1 \wedge z = 0, \alpha_b := 2 \rangle, \quad \text{cost} = 0$$

$$b^{1,z=1} = \langle \alpha_b = 1 \wedge z = 1, \alpha_b := 2 \rangle, \quad \text{cost} = 1$$

$$b^e = \langle \alpha_b = 2, u := 2 \wedge \sigma := 0 \wedge \alpha_b := 0 \rangle, \quad \text{cost} = 0$$

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

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

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References



- 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.

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References



Relaxations

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

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

Question: Are they also informative in SDAC planning?

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

- 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?

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References



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 $d \in \mathcal{D}_v$ 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).

Background

Compilation

Relaxations

Delete Relaxations
in SAS⁺

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References



- 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.

Background

Compilation

Relaxations

Delete Relaxations
in SAS⁺

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

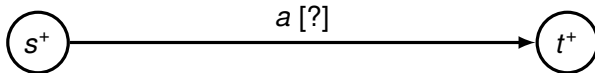
References

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^+ ?



Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic
Relaxed Planning
Graph

Abstractions

Practice

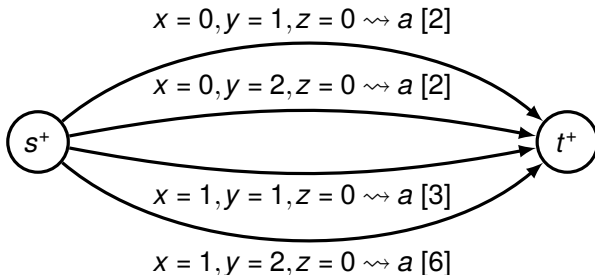
Summary

References

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



Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

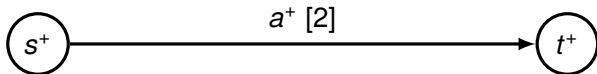
Summary

References

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



Background

Compilation

Relaxations

Delete Relaxations
in SAS⁺

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

Definition

Let V be a set of FDR variables, $s : V \rightarrow \bigcup_{v \in V} \mathcal{D}_v$ an unrelaxed state over V , and $s^+ \subseteq \{(v, d) \mid v \in V, d \in \mathcal{D}_v\}$ 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^+ ?)

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

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 $cost_a$.

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

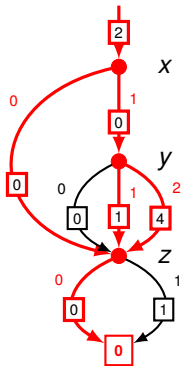
Practice

Summary

References

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!

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

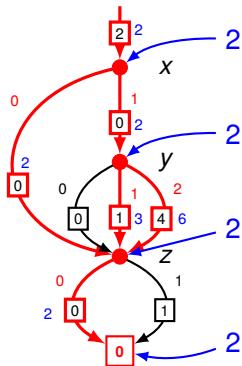
Practice

Summary

References

Example

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



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

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

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? □

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

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

Definition (Classical additive heuristic h^{add})

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

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

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

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

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

Example

$$a = \langle T, x=1 \rangle$$

$$\text{cost}_a = 2 - 2y$$

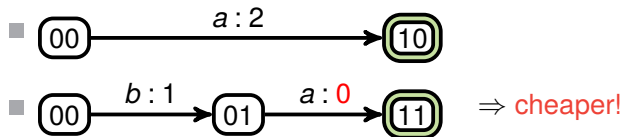
$$b = \langle T, y=1 \rangle$$

$$\text{cost}_b = 1$$

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

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

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



Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

(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)

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

$$h_s^{add}(fact)$$

$$= \begin{cases} 0 \\ \min_{\text{achiever } a \text{ of } fact} \end{cases}$$

$$Cost_a^s = \min_{\hat{s} \in S_a} [cost_a(\hat{s}) + h_s^{add}(\hat{s})]$$

S_a : set of partial states over variables in cost function

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

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

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 . □

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}

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References

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} .

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

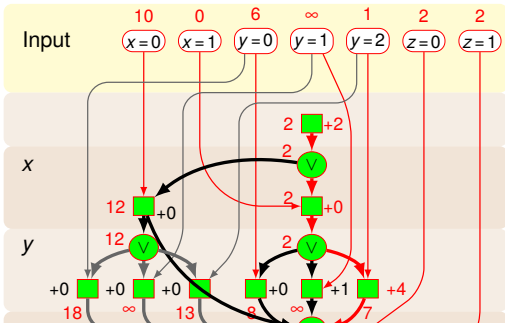
Abstractions

Practice

Summary

References

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 : $\sum(\text{parents}) + \text{weight}$
- \vee : $\min(\text{parents})$
- $Cost_a^S =$

- 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. □

Background

Compilation

Relaxations

Delete Relaxations
in SAS*

Costs in Relaxed
States

Additive Heuristic

Relaxed Planning
Graph

Abstractions

Practice

Summary

References



Abstractions

Background

Compilation

Relaxations

Abstractions

Cartesian
Abstractions

Practice

Summary

References

Background

Compilation

Relaxations

Abstractions

Cartesian
Abstractions

Practice

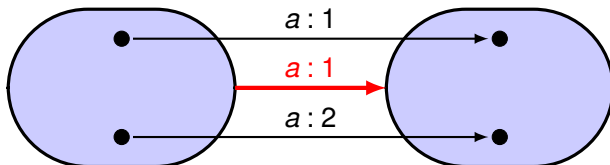
Summary

References

Question: Why consider **abstraction heuristics**?

Answer:

- admissibility
- \rightsquigarrow **optimality**



Question: What are the **abstract action costs**?

Answer: For **admissibility**, abstract cost of a should be

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

Problem: exponentially many states in minimization

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

Background

Compilation

Relaxations

Abstractions

Cartesian
Abstractions

Practice

Summary

References

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 $D_i \subseteq \mathcal{D}_i$ for all $i = 1, \dots, n$.

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!

Background

Compilation

Relaxations

Abstractions

Cartesian
Abstractions

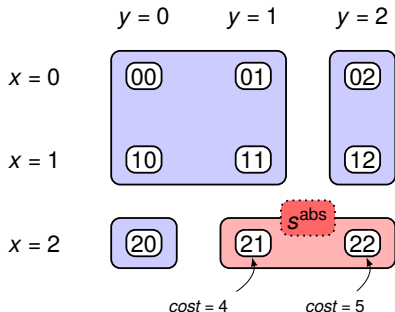
Practice

Summary

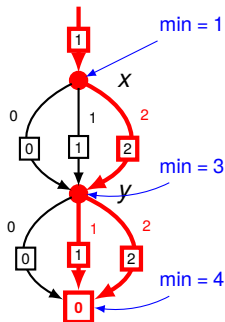
References

Example (Cartesian abstraction)

Cartesian abstraction over x, y



Cost $x + y + 1$
(edges consistent with s^{abs})



- Background
- Compilation
- Relaxations
- Abstractions
 - Cartesian Abstractions
- Practice
- Summary
- References



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 $d \in \mathcal{D}_v$ 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!

Background

Compilation

Relaxations

Abstractions

Cartesian
Abstractions

Practice

Summary

References

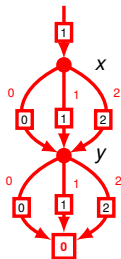
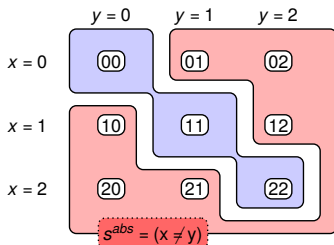
If abstraction **not Cartesian**: two variables can be

- **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!

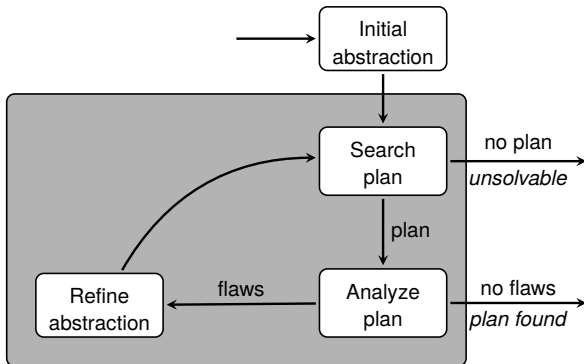


Counterexample-Guided Abstraction Refinement



Wanted: principled way of computing Cartesian abstractions.

↪ Counterexample-Guided Abstraction Refinement (CEGAR)
(details omitted)



- Background
- Compilation
- Relaxations
- Abstractions
 - Cartesian Abstractions
- Practice
- Summary
- References



Practice

Background

Compilation

Relaxations

Abstractions

Practice

Libraries

PDDL

Summary

References



- **MEDDLY:** Multi-terminal and Edge-valued Decision Diagram LibrarY
- **Authors:** Junaid Babar and Andrew Miner
- **Language:** C++
- **License:** open source (LGPLv3)
- **Advantages:**
 - many different types of decision diagrams
 - mature and efficient
- **Disadvantages:**
 - documentation
- **Code:** <http://meddly.sourceforge.net>

Background

Compilation

Relaxations

Abstractions

Practice

Libraries

PDDL

Summary

References



- **pyevmdd**: EVMDD library for Python
- **Authors**: RM and Florian Geißer
- **Language**: Python
- **License**: open source (GPLv3)
- **Disadvantages**:
 - restricted to EVMDDs
 - neither mature nor optimized
- **Purpose**: our EVMDD playground
- **Code**:
`https://github.com/robertmattmueller/pyevmdd`
- **Documentation**:
`http://pyevmdd.readthedocs.io/en/latest/`

Background

Compilation

Relaxations

Abstractions

Practice

Libraries

PDDL

Summary

References

Usual way of representing costs in PDDL:

- `effects` (increase (total-cost) (<expression>))
- `metric` (minimize (total-cost))

Custom syntax (non-standard PDDL):

- Besides `:parameters`, `:precondition`, and `:effect`, actions may have field
- `:cost` (<expression>)

Background

Compilation

Relaxations

Abstractions

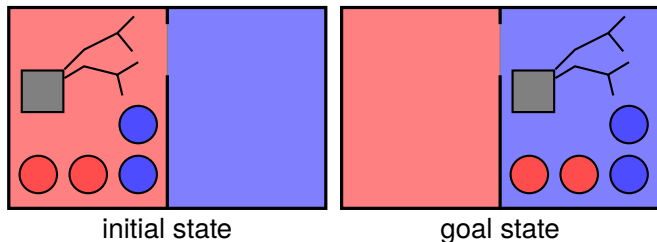
Practice

Libraries

PDDL

Summary

References



- Colored rooms and balls
- Cost of move increases if ball color differs from room color
- Goal did not change!

$$\begin{aligned} \text{cost}(\text{move}) = & \sum_{\text{ROOM}} \sum_{\text{BALL}} (\text{at}(\text{BALL}, \text{ROOM}) \wedge (\text{red}(\text{BALL})) \wedge (\text{blue}(\text{ROOM})) \\ & + \sum_{\text{ROOM}} \sum_{\text{BALL}} (\text{at}(\text{BALL}, \text{ROOM}) \wedge (\text{blue}(\text{BALL})) \wedge (\text{red}(\text{ROOM})) \end{aligned}$$



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

Example (EVMD-based action compilation)

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

Auxiliary variables:

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

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

Background

Compilation

Relaxations

Abstractions

Practice

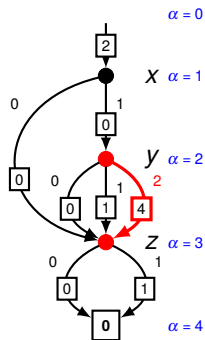
Libraries

PDDL

Summary

References

Example (EVMDD-based action compilation, ctd.)



$$a^x = \langle \chi \wedge \sigma = 0 \wedge \alpha = 0, \\ \sigma := 1 \wedge \alpha := 1 \rangle,$$

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

Background

Compilation

Relaxations

Abstractions

Practice

Libraries

PDDL

Summary

References

- **Disclaimer:**

- Not completely functional
- Still some bugs

- **Uses `pyevmdd`**

- **Language:** Python

- **License:** open source

- **Code:** `https:`

`//github.com/robertmattmueller/sdac-compiler`

Background

Compilation

Relaxations

Abstractions

Practice

Libraries

PDDL

Summary

References



Summary

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References

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.

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References

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.

Background

Compilation

Relaxations

Abstractions

Practice

Summary

References



References

Background

Compilation

Relaxations




Abstractions

Practice

Summary

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.