# UNI

## Principles of AI Planning

12. Planning with State-Dependent Action Costs

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel and Robert Mattmüller

December 18th, 2017



#### Background

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

Relaxations

Abstractions

Summary

References

# 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, ...)
    - √ later
  - Algorithms: techniques other than heuristic search (SAT/SMT planning, BDD-based symbolic 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

Backgrour

State-Dependent Action Costs

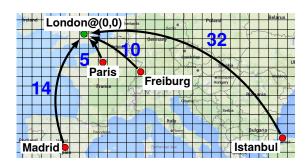
Edge-Valued Multi-Valued Decision Diagran

Abstractions

Summary

Cummary





#### Backgroun

#### State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrams

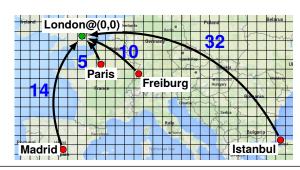
Compilation

Relaxations

Abstractions

Summary





Backgroun

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagra

Compilation

Relaxations

Abstractions

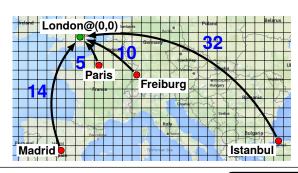
Summary

eferences

Action costs: unit constant state-dependent

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





Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagra

Compilation

Relaxations

Abstractions

Summary

eferences

Action costs:

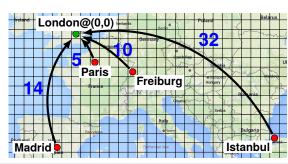
unit

constant

state-dependent

 $cost(fly(Madrid, London)) = 14, \quad cost(fly(Paris, London)) = 5, \\ cost(fly(Freiburg, London)) = 10, \quad cost(fly(Istanbul, London)) = 32.$ 





Action costs: (unit) constant state-dependent

$$cost(flyTo(London)) = |x_{London} - x_{current}| + |y_{London} - y_{current}|$$
  
=  $|x_{current}| + |y_{current}|$ .

Backgroun

#### State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagra

Compilation

Relaxations

Abstractions

Summary

## 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 Diagrar

Oomphation

Abstractions

Summarv

. ,

## Why Study State-Dependent Action Costs?





- Human perspective:
  - "natural", "elegant", and "higher-level"
  - modeler-friendly \( \simes \) less error-prone?
- Machine perspective:
  - more structured \( \to \) 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

Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrar

Jomphalior

Abstractions

Summary

elerences

(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.

Backgroun

State-Dependent Action Costs Edge-Valued

Edge-Valued Multi-Valued Decision Diagran

Abstractions

Summarv

## Handling State-Dependent Action Costs



#### Roadmap:

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

Backgroun

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrar

Compliatio

Abstractions

Summary





#### Definition

A SAS<sup>+</sup> 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<sup>+</sup> 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 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.

Backgroun

State-Dependent Action Costs Edge-Valued

Edge-Valued Multi-Valued Decision Diagram

.

Abstractions

Summary

#### Example (Household domain)

#### Actions:

```
\label{eq:vacuumFloor} \begin{split} & \text{vacuumFloor} = \langle \top, \, \text{floorClean} \rangle \\ & \text{washDishes} = \langle \top, \, \text{dishesClean} \rangle \\ & \text{doHousework} = \langle \top, \, \text{floorClean} \wedge \, \text{dishesClean} \rangle \end{split}
```

#### Cost functions:

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

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

cost_{doHousework} = cost_{vacuumFloor} + cost_{washDishes}
```

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

Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagram

Compliation

Abstractions

Summary

Compilations



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 Diagran

Compilati

Relaxations

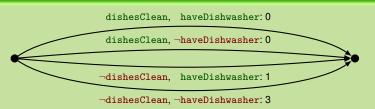
Abstractions

Summary

Compilation I: "Parallel Action Decomposition"



#### Example



washDishes( dC, hD) = 
$$\langle$$
 dC  $\wedge$  hD, dC $\rangle$ ,  $cost = 0$  washDishes( dC, ¬hD) =  $\langle$  dC  $\wedge$  ¬hD, dC $\rangle$ ,  $cost = 0$  washDishes(¬dC, hD) =  $\langle$  ¬dC  $\wedge$  hD, dC $\rangle$ ,  $cost = 1$  washDishes(¬dC, ¬hD) =  $\langle$  ¬dC  $\wedge$  ¬hD, dC $\rangle$ ,  $cost = 3$ 

Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagran

Joinpliation

Relaxations

Abstractions

Summary

THE PART OF THE PA



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? → Compilation II

Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrar

ompliation

Abstractions

Summary

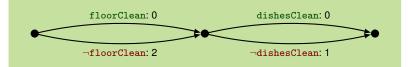
Compilation II: "Purely Sequential Action Decomposition"



#### Example

Assume we own a dishwasher:

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



$$\label{eq:doHousework1} \begin{array}{ll} \mbox{doHousework}_1(\ \ \mbox{fC}) = \langle \ \ \mbox{fC}, \ \ \mbox{fC} \rangle, & \mbox{cost} = 0 \\ \mbox{doHousework}_1(\ \mbox{\neg fC}) = \langle \mbox{\neg fC}, \ \mbox{fC} \rangle, & \mbox{cost} = 2 \\ \mbox{doHousework}_2(\ \mbox{dC}) = \langle \mbox{dC}, \ \mbox{dC} \rangle, & \mbox{cost} = 0 \\ \mbox{doHousework}_2(\mbox{\neg dC}) = \langle \mbox{\neg dC}, \mbox{dC} \rangle, & \mbox{cost} = 1 \\ \end{array}$$

Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Relaxations

Abstractions

Summary

Compilation II: "Purely Sequential Action Decomposition"



#### Backgroun

#### State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrar

Compliation

Relaxations

Abstractions

Summary

References

#### Compilation II

If costs are additively decomposable/separable:

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

Compilation II: "Purely Sequential Action Decomposition"



#### Properties:

- only 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

 $doHousework_1(\neg fC), doHousework_2(\neg dC)$ 

in the compiled setting.

Backgroun

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagram

Joinpliation

Abstractions

Summary

Compilation II: "Purely Sequential Action Decomposition"



# FREIBT -

#### 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 doHousework<sub>1</sub>( $\neg fC$ ) or doHousework<sub>2</sub>( $\neg dC$ ) first?

- → 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)? --> Compilation III

Background

State-Dependent Action Costs Edge-Valued Multi-Valued

Compilation

Relaxations

Abstractions

\_

Summary

Compilation III: "EVMDD-Based Action Decomposition"

THE STATE OF THE S



Example

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

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

dishesClean, haveDishwasher: 0

floorClean: 0

dishesClean, ¬haveDishwasher: 0

¬dishesClean, haveDishwasher: 1

¬dishesClean, ¬haveDishwasher: 3

#### 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

Compilation

Relaxations

Abstractions

Summary

Compilation III: "EVMDD-Based Action Decomposition"









Later:

- Compiled actions
- Auxiliary variables to enforce action ordering

Background

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Relaxations

Abstractions

Summary

Compilation III: "EVMDD-Based Action Decomposition"



#### 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!

Backgroun

State-Dependent Action Costs

Edge-Valued Multi-Valued Decision Diagram

----

Abstractions

Summary

Julililaly

eterences

Compilation III: "EVMDD-Based Action Decomposition"



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]

State-Dependent Action Costs

Relaxations

Abstractions

Summary

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

Backgroun

State-Dependent

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Abstractions

\_

Summary

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)

#### Background

State-Dep Action Co

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Relaxations

Abstractions

Summary

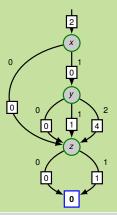
Edge-Valued Multi-Valued Decision Diagrams



## Example (EVMDD Evaluation)

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

$$\mathcal{D}_X = \mathcal{D}_Z = \{0, 1\}, \ \mathcal{D}_Y = \{0, 1, 2\}$$



- 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  $x \neq 0$ .

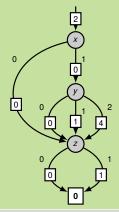
Edge-Valued Multi-Valued Decision Diagrams

# II IBURG

#### Example (EVMDD Evaluation)

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

$$\mathcal{D}_{x} = \mathcal{D}_{z} = \{0, 1\}, \ \mathcal{D}_{y} = \{0, 1, 2\}$$



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

$$cost_a(s) =$$

#### Backgroun

State-Dependent

Edge-Valued Multi-Valued Decision Diagrams

Compliation

Relaxations

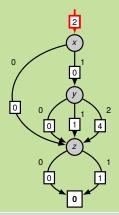
Abstractions

Summary

Edge-Valued Multi-Valued Decision Diagrams

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

$$\mathcal{D}_X = \mathcal{D}_Z = \{0, 1\}, \ \mathcal{D}_Y = \{0, 1, 2\}$$



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

$$oldsymbol{o}$$
  $cost_a(s) = 2 +$ 

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Relaxations

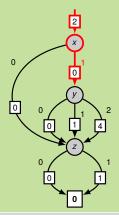
Abstractions

Summary

# BURG

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

$$\mathcal{D}_X = \mathcal{D}_Z = \{0, 1\}, \ \mathcal{D}_Y = \{0, 1, 2\}$$



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

$$cost_a(s) = 2 + 0 +$$

Edge-Valued Multi-Valued Decision Diagrams

Compliation

Relaxations

Abstractions

Summary

Edge-Valued Multi-Valued Decision Diagrams

# II IBURG

Multi-Valued Decision Diagrams

Compliation

Relaxations

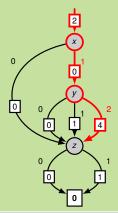
Abstractions

Summary

#### Example (EVMDD Evaluation)

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

$$\mathcal{D}_{x} = \mathcal{D}_{z} = \{0, 1\}, \ \mathcal{D}_{y} = \{0, 1, 2\}$$



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

$$cost_a(s) = 2 + 0 + 4 +$$

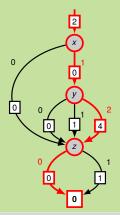
Edge-Valued Multi-Valued Decision Diagrams

## II IBURG

#### Example (EVMDD Evaluation)

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

$$\mathcal{D}_{x} = \mathcal{D}_{z} = \{0, 1\}, \ \mathcal{D}_{y} = \{0, 1, 2\}$$



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

$$cost_a(s) = 2 + 0 + 4 + 0 = 6$$

Multi-Valued Decision Diagrams

Compilation

Relaxations

Abstractions

Summary

Edge-Valued Multi-Valued Decision Diagrams



#### 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)

#### Backgroun

State-Dependent

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Relaxations

Abstractions

Summary

Arithmetic operations on EVMDDs



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

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

- Base case: single-node EVMDDs encoding constants
- Inductive case: apply ⊗ recursively:
  - push down edge weights
  - recursively apply ⊗ 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

Backgroun

State-Dependent

Edge-Valued Multi-Valued Decision Diagrams

Compilation

Relaxations

Summary



Compilation

Background

Compilation Relaxations

Abstractions

Summary

## **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  $\sigma$  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).

Backgroun

Compilation

Relaxations

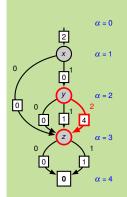
Abstractions

Summary

# **EVMDD-Based Action Compilation**



### Example (EVMDD-based action compilation, ctd.)



$a^{\chi} = \langle \chi \wedge \sigma = 0 \wedge \alpha = 0,$	
$\sigma \coloneqq 1 \wedge \alpha \coloneqq 1 \rangle,$	cost = 2
$a^{1,x=0} = \langle \alpha = 1 \land x = 0, \ \alpha := 3 \rangle,$	cost = 0
$a^{1,x=1} = \langle \alpha = 1 \land x = 1, \alpha := 2 \rangle,$	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
$a^{2,y=2} = \langle \alpha = 2 \land y = 2, \ \alpha := 3 \rangle,$	cost = 4
$a^{3,z=0} = \langle \alpha = 3 \wedge z = 0, \ \alpha := 4 \rangle,$	cost = 0
$a^{3,z=1} = \langle \alpha = 3 \land z = 1, \ \alpha := 4 \rangle,$	cost = 1
$a^e = \langle \alpha = 4, e \wedge \sigma := 0 \wedge \alpha := 0 \rangle$ .	cost = 0

### 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<sub>a</sub>.

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  $\Pi$  using  $\mathcal{E}_a$ ,  $a \in O$ , is the task  $\Pi' = EAC(\Pi) = \langle V', I', O', \gamma' \rangle$ , where

$$V' = V \cup \{\sigma\} \cup \{\alpha_a \mid a \in O\},$$

$$\blacksquare I' = I \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\},\$$

$$O' = \bigcup_{a \in O} EAC(a)$$
, and

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

# **EVMDD-Based Action Compilation**



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

### Proposition

 $\Pi'$  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  $\Pi$  and the largest used EVMDD.

### Proof.

By construction.

B. d. .

Compilation

Relaxations

riolaxatione

Abstractions Summary

# **EVMDD-Based Action Compilation**



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

### **Proposition**

 $\Pi$  and  $\Pi'$  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  $\Pi$ , 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, ..., n.

For each i = 1, ..., 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  $a_i^0, \ldots, a_i^{k_i}$ (for some  $k_i \in \mathbb{N}$ ; including  $a_i^{\chi}$  and  $a_i^e$ ).

Compilation Relaxations

Abstractions Summary



# AR L

### Proof (ctd.)

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

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,

$$cost(\pi') = cost(a_1^0) + \dots + cost(a_n^{k_1}) + \dots + cost(a_n^0) + \dots + cost(a_n^{k_n}) = cost_{a_1}(s_0) + \dots + cost_{a_n}(s_{n-1}) = cost(\pi).$$

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

Backgroun

Compilation Belayations

Abstractions

Summary



### 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

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

Optimal plan for  $\Pi$ :

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

Backgroun

Compilation

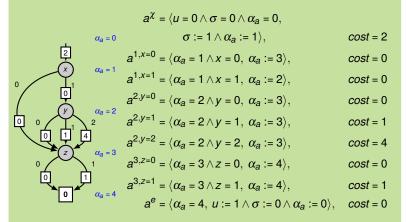
Relaxations

Abstractions Summary



### Example (Ctd.)

### Compilation of a:



Compilation Relaxations

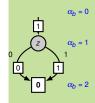
Abstractions

Summary



### Example (Ctd.)

### Compilation of b:



$$b^{\chi} = \langle u = 1 \land \sigma = 0 \land \alpha_b = 0,$$

$$\sigma := 1 \land \alpha_b := 1 \rangle, \qquad cost = 1$$

$$b^{1,z=0} = \langle \alpha_b = 1 \land z = 0, \ \alpha_b := 2 \rangle, \qquad cost = 0$$

$$b^{1,z=1} = \langle \alpha_b = 1 \land z = 1, \ \alpha_b := 2 \rangle, \qquad cost = 1$$

$$b^e = \langle \alpha_b = 2, \ u := 2 \land \sigma := 0 \land \alpha_b := 0 \rangle, \qquad cost = 0$$

Background

Compilation

Relaxations

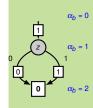
Abstractions

Summary References



### Example (Ctd.)

### Compilation of *b*:



$$b^{\chi} = \langle u = 1 \land \sigma = 0 \land \alpha_b = 0,$$

$$\sigma := 1 \land \alpha_b := 1 \rangle, \qquad cost = 1$$

$$b^{1,z=0} = \langle \alpha_b = 1 \land z = 0, \ \alpha_b := 2 \rangle, \qquad cost = 0$$

$$b^{1,z=1} = \langle \alpha_b = 1 \land z = 1, \ \alpha_b := 2 \rangle, \qquad cost = 1$$

$$b^e = \langle \alpha_b = 2, \ u := 2 \land \sigma := 0 \land \alpha_b := 0 \rangle, \qquad cost = 0$$

Optimal plan for  $\Pi'$  (with  $cost(\pi') = 6 + 1 = 7 = cost(\pi)$ ):

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

Background

Compilation

Relaxations

Abstractions Summary

# Planning with State-Dependent Action Costs



- 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.

Backgroun

Compilation

Relaxations

Abstractions Summary



### Background

Compilation

#### Relaxations

Delete Relaxations in SAS+ Costs in Relaxed

States Additive Heuristic

Relaxed Planning Graph

Abstractions

Summary

References

# Relaxations

### Relaxation Heuristics



Background

Compilation

#### Relaxations

Delete Relaxations in SAS+

States Additive Heurist

Relaxed Planning Graph

Abstractions

Summary

References

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 hadd in
- But what does an action a cost in a relaxed state  $s^+$ ?

a task with state-dependent action costs.

And how to compute that cost?

Backgroun

Compilation

Relaxat

Delete Relaxations in SAS<sup>+</sup>

Costs in Relaxed States

Relaxed Planning

Graph

Abstractions

Summary

### Relaxed SAS<sup>+</sup> 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  $d \in \mathcal{D}_v$  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).

Backgroun

Compilation

.

Delete Relaxations

in SAS+ Costs in Relayed

> Additive Heuristic Relaxed Planning

Grapn

Abstraction

Julilliary

### Relaxed SAS<sup>+</sup> Tasks

- UNI FREIBURG
- 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  $a^+ = \langle x = 2, y := 1 \land z := 0 \rangle$  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.

Backgroun

Compilation

Relaxations

Delete Relayations

in SAS\*

Additive Heuristic Relaxed Planning

Abstractions

Summary



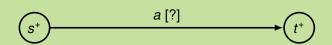
# FREE

### Example

Assume  $s^+$  is the relaxed state with

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

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



### Background

Compilation

### Relaxations

Delete Relaxations in SAS<sup>+</sup>

#### Costs in Relaxed States

Relaxed Planning Graph

### Abstractions

### Summary



REB

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 $x = 0, y = 1, z = 0 \rightsquigarrow a [2]$ $x = 0, y = 2, z = 0 \rightsquigarrow a [2]$ $x = 1, y = 1, z = 0 \rightsquigarrow a [3]$ $x = 1, y = 2, z = 0 \rightsquigarrow a [6]$

Backgroun

Compilation

Relevations

in SAS+ Costs in Relayed

States
Additive Heuristic

Relaxed Planning Graph

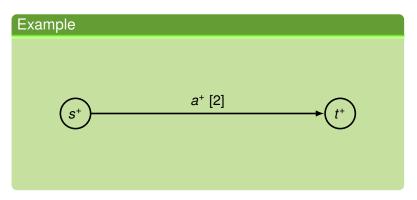
Abstractions

Summary



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.)



Delete Relaxations

Costs in Relayed States

Relaxed Planning

Abstractions



NE NE

### 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  $s^+ \subseteq \{(v,d) | v \in V, d \in \mathscr{D}_v\}$  a relaxed state over V. We call s consistent with  $s^+$  if  $\{(v,s(v)) | v \in V\} \subseteq s^+$ .

### 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  $s^+$ ?)

Backgroun

Compilation

Bullion Bullion

Delete Relaxations in SAS+ Costs in Relayed

States
Additive Heuristic
Relaxed Planning

A le etue eti e e

Abotraction

. . . . . .



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<sub>a</sub>.

Costs in Relayed States

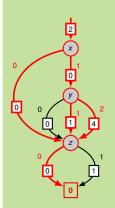
Abstractions



# FREB

### Example

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



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

Backgroun

Compilation

\_ .

Delete Relaxations in SAS+

States
Additive Heuristic
Relaxed Planning

Abotroction

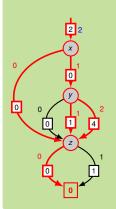
outilitial y



# UNI FREIB

### Example

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



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

Backgroun

Compilation

Compliation

Delete Relaxations in SAS+ Costs in Relaxed

States
Additive Heuristic
Relaxed Planning

Abstractions

Limmani

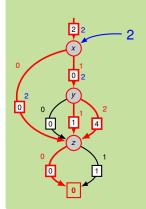
oforonooo



# FREIB

### Example

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



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

Backgroun

Compilation

\_ .

Delete Relaxation in SAS<sup>+</sup> Costs in Relaxed

States
Additive Heuristic
Relaxed Planning

Abstractions

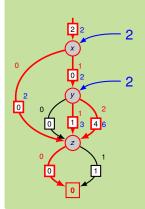
oforonoos



# UNI FREIBL

### Example

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



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

Backgroun

Compilation

Delegan

Delete Relaxations in SAS<sup>+</sup> Costs in Relaxed

States
Additive Heurist
Relaxed Plannir

Abstractions

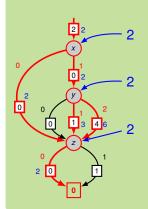
oforonooo



# FREIB

### Example

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



Computing cost<sub>a</sub>(s<sup>+</sup>) = minimizing over cost<sub>a</sub>(s) for all s consistent with s<sup>+</sup> = minimizing over all start-end-paths in EVMDD following only edges consistent with s<sup>+</sup>.

 Observation: Minimization over exponentially many paths can be replaced by top-sort traversal of EVMDD, minimizing over incoming arcs consistent with s<sup>+</sup> at all nodes! Backgroun

Compilation

5

Delete Relaxation in SAS<sup>+</sup> Costs in Relaxed

States
Additive Heurist

Graph

7 IDOII GOIIOTTI

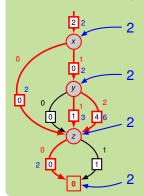
bullillary





### 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,  $s(y) \in \{1, 2\}$ .

Backgroun

Compilation

Relaxations

Delete Relaxations in SAS<sup>+</sup> Costs in Relaxed

States
Additive Heuristic
Relaxed Planning

Abstractions

7 IDOII GOIIOTTI



SE SE

### 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?

Backgroun

Compilation

Delete Relaxation

Costs in Relaxed States

Additive Heurist Relaxed Plannir

Abstractions

ounniur y



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

# Definition (Classical additive heuristic *h*<sup>add</sup>)

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

Question: How to generalize hadd to SDAC?

Backgroun

Compilation

Delete Relaxation in SAS\*

Costs in Relaxed States

Additive Heuristic Relaxed Planning Graph

Abstraction

Abstractions

Summary





### Example

$$a = \langle \top, x = 1 \rangle$$
  $cost_a = 2 - 2y$   
 $b = \langle \top, y = 1 \rangle$   $cost_b = 1$   
 $s = \{x \mapsto 0, y \mapsto 0\}$   
 $h_s^{add}(y = 1) = 1$   
 $h_s^{add}(x = 1) = ?$ 

### Background

### Compilation

### Relaxations

Delete Relaxations in SAS+ Costs in Relaxed

#### Additive Heuristic Relaxed Planning

Graph

### Abstractions

Summary

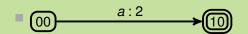


# UNI

### Example

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

$$s = \{x \mapsto 0, y \mapsto 0\}$$
$$h_s^{add}(y=1) = 1$$
$$h_s^{add}(x=1) = ?$$



### Background

### Compilation

### Relaxations

Delete Relaxations in SAS+

States
Additive Heuristic

### Relaxed Planning

Graph

Abstractions

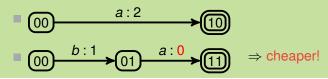


# NE NE

### Example

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

$$s = \{x \mapsto 0, y \mapsto 0\}$$
$$h_s^{add}(y = 1) = 1$$
$$h_s^{add}(x = 1) = ?$$



Background

Compilation

Relaxations

Delete Relaxations in SAS<sup>+</sup>

Costs in Relaxer States

> Additive Heuristic Relaxed Planning

Graph

Abstractions

Janimary



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

### Definition (Additive heuristic hadd for SDAC)

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

Backgroun

Compilation

in SAS+

States

Additive Heuristic

Relaxed Planning Graph

ADSITACTIONS

Summary



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

### Definition (Additive heuristic hadd for SDAC)

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

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

 $S_a$ : set of partial states over variables in cost function

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

Backgroun

Compilation

.

in SAS\*

Costs in Relaxed States

Graph

Abstractions

Summary



### **Theorem**

Let  $\Pi$  be an SDAC planning task, let  $\Pi'$  be an EVMDD-based action compilation of  $\Pi$ , and let s be a state of  $\Pi$ . Then the classical  $h^{add}$  heuristic in  $\Pi'$  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  $\Pi$ .

### Computing hadd for SDAC:

- Option 1: Compute classical hadd on compiled task.
- Option 2: Compute *Cost*<sup>s</sup> directly. How?
  - Plug EVMDDs as subgraphs into RPG
  - ~→ efficient computation of h<sup>add</sup>

Backgroun

Compilation

Dolovetions

in SAS<sup>+</sup>

Costs in Relaxer States

Additive Heuristic Relaxed Planning

an .

Abstractions

Summary

# **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<sup>add</sup>.

Backgroun

Compilation

Delevetiese

Delete Relaxation in SAS+

> Costs in Relaxed States

Relaxed Planning Graph

Abstractions

Summary

# Option 2: RPG Compilation







#### Relaxations

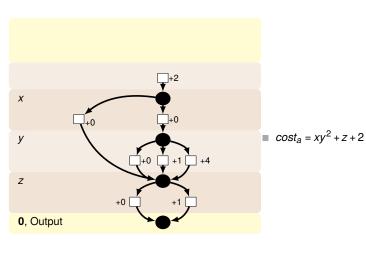
Delete Relaxations in SAS+ Costs in Relaxed

Additive Heuristic

Relaxed Planning Graph

### Abstractions

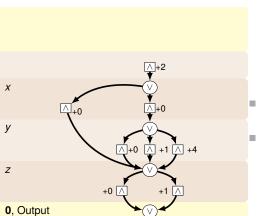
### Summary



# Option 2: RPG Compilation







variable nodes become ∨-nodes

■ weights become ^-nodes Background

Compilation

Relaxations

Delete Relaxations in SAS+

Additive Heuristic
Relaxed Planning

Graph

Abstractions

Summary

# Option 2: RPG Compilation





Background

Relaxations

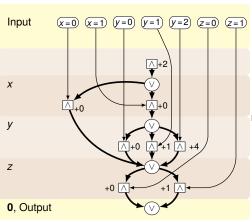
Delete Relaxations

Relaxed Planning Graph

Abstractions

Summary

References



Augment with input nodes

# **Option 2: RPG Compilation**







Compilatio

### Relaxations

Delete Relaxations in SAS\*

Costs in Relaxed

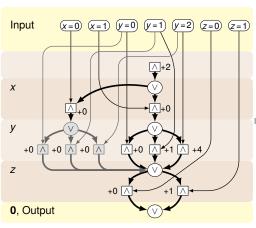
Additive Heuristic Relaxed Planning

Graph
Abstractions

Abstraction

Summary

References



Ensure complete evaluation







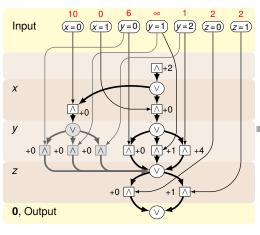
### Relaxations

Delete Relaxations

Relaxed Planning Graph

Abstractions

References



Insert hadd values







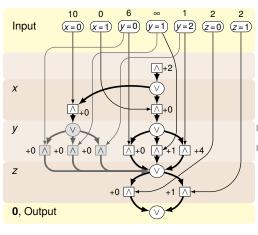
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight







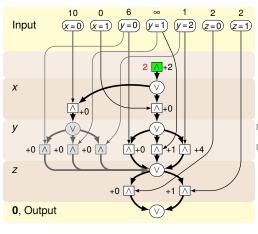
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight

■ ∨: min(parents)







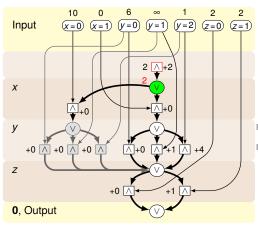
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight







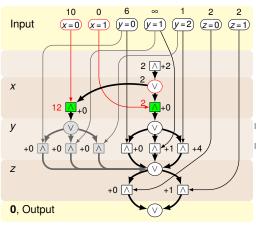
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight







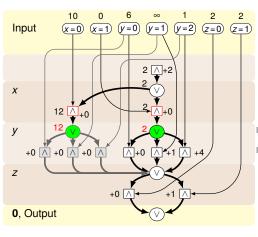
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight







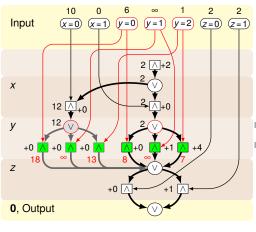
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight







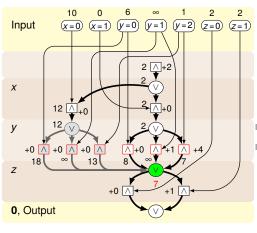
### Relaxations

Delete Relaxations

Relaxed Planning

## Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight





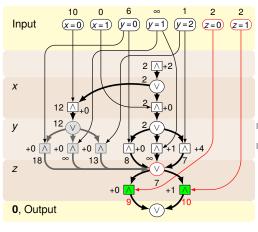


### Relaxations

Delete Relaxations

Relaxed Planning Graph

References



Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight

■ ∨: min(parents)







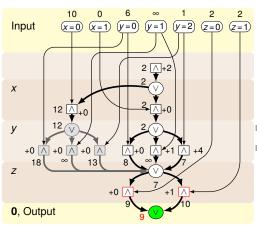
### Relaxations

Delete Relaxations

Relaxed Planning

Graph

References



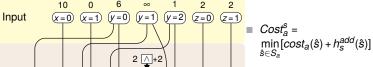
Evaluate nodes:

 $\wedge$ :  $\sum$ (parents) + weight

■ ∨: min(parents)







Delete Relaxations

Relaxed Planning Graph

Abstractions

References

0, Output

12 🔨

+0 1 +0 1

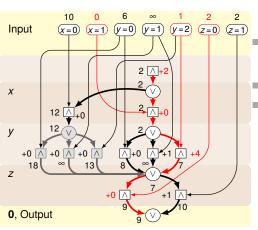
+0 ^

Х

# Option 2: Computing Cost<sub>a</sub><sup>s</sup>







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

$$\bigcirc$$
 cost<sub>a</sub> =  $xy^2 + z + 2$ 

$$\hat{s} = \{x \mapsto 1, y \mapsto 2, z \mapsto 0\}$$

Backgroun

Compilation

Delevations

Delete Relaxations in SAS+

States
Additive Heuristic
Relaxed Planning

Graph

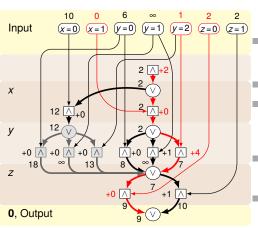
Abstractions

Summary

# Option 2: Computing Cost<sub>a</sub><sup>s</sup>







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

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

$$cost_a(\hat{s}) = 1 \cdot 2^2 + 0 + 2 = 6$$
$$= 2 + 0 + 4 + 0$$

$$h_s^{add}(\hat{s}) = 0 + 1 + 2 = 3$$

### Backgroun

Compilation

Delete Relaxations in SAS+

Additive Heuristic
Relaxed Planning

Graph

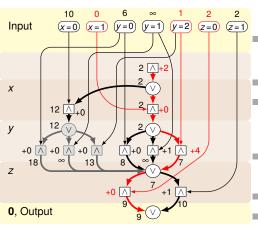
### Abstractions

### Summary

# Option 2: Computing Cost<sub>a</sub><sup>s</sup>







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

$$\bigcirc$$
 cost<sub>a</sub> =  $xy^2 + z + 2$ 

$$cost_a(\hat{s}) = 1 \cdot 2^2 + 0 + 2 = 6$$
  
= 2 + 0 + 4 + 0

$$h_s^{add}(\hat{s}) = 0 + 1 + 2 = 3$$

$$Cost_a^s = 6 + 3 = 9$$

Backgroun

Compilation

Relaxations

Delete Relaxations in SAS+

Additive Heuristic

Graph
Abstractions

Summary

## Additive Heuristic



- 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  $\Pi$  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.

Backgroun

Compilation

Relaxations

Delete Relaxation in SAS\*

States

Additive Heuristi

Relaxed Planning Graph

Abstractions

Summary

Defense



Background

Relaxations

### Abstractions

Cartesian Abstractions CEGAR

Summary

References

# **Abstractions**



Question: Why consider abstraction heuristics?

### Answer:

- admissibility
- ~→ optimality

Background

Compilatio

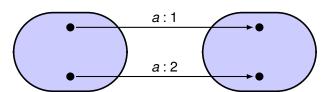
Relaxations

### Abstractions

Abstractions CEGAR

Summary





Background

Compilation

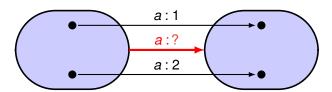
Relaxations

### Abstractions

Cartesian Abstractions CEGAR

Summary





Question: What are the abstract action costs?

Background

Compilation

Relaxations

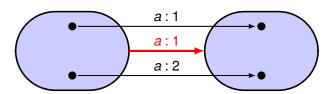
Abstractions

Cartesian Abstractions

CEGAR
Summary

\_ .





Question: What are the abstract action costs?

Answer: For admissibility, abstract cost of a should be

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

Background

Compilation

Relaxations

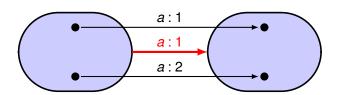
### Abstractions

CEGAR

Summary







Question: What are the abstract action costs?

Answer: For admissibility, abstract cost of a should be

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

Problem: exponentially many states in minimization

Aim: Compute  $cost_a(s^{abs})$  efficiently

(given EVMDD for  $cost_a(s)$ ).

Background

Compilation

Relaxations

### Abstractions

CEGAR

Summary

II EIBURG

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

(Includes projections and domain abstractions.)

Background

Compilation

Relaxations

Abstractions

Cartesian Abstractions

Summary



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

(Includes projections and domain abstractions.)

# Definition (Cartesian abstraction)

A set of states sabs 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, ..., n.

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

[Seipp and Helmert, 2013]

Intuition: Variables are abstracted independently.

→ exploit independence when computing abstract costs!

Background

Compliation

Abstractions

Cartesian Abstractions

Odminary



## Example (Cartesian abstraction)

Cartesian abstraction over x, y

$$y = 0$$
  $y = 1$   $y = 2$ 

$$x = 0$$

$$x = 1$$





Background

Relaxations

Abstractions

Cartesian

Abstractions CEGAR



NE NE

# Example (Cartesian abstraction)

Cartesian abstraction over x, y

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

$$x = 0$$

$$y = 0$$
  $y = 1$ 

00 01



*x* = 1

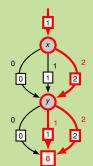
10 11

(12)

*x* = 2

20

21 s<sup>abs</sup> 22



Relaxations

Abstractions

Background

Cartesian Abstractions CEGAR

Summary



NE NE

Background

Relaxations

Abstractions
Cartesian
Abstractions

# Example (Cartesian abstraction)

Cartesian abstraction over x, y

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

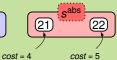
$$x = 0$$

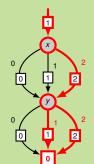
$$y = 0$$
  $y = 1$ 





(10)







Background

Relaxations

Abstractions
Cartesian
Abstractions

# Example (Cartesian abstraction)

Cartesian abstraction over x, y

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

$$x = 0$$

$$y = 0$$
  $y = 1$ 

(00)

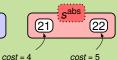
(02)

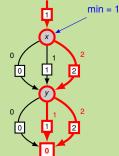


(12)

$$x = 2$$







B. Nebel, R. Mattmüller - Al Planning



Background

Relaxations

Abstractions Cartesian Abstractions

# Example (Cartesian abstraction)

Cartesian abstraction over x, y

Cost x + y + 1(edges consistent with sabs)

$$x = 0$$

$$y = 0$$
  $y = 1$ 

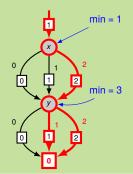
$$x = 1$$











B. Nebel, R. Mattmüller - Al Planning



# NE NE

# Example (Cartesian abstraction)

Cartesian abstraction over x, y

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

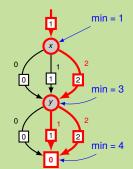
$$y = 0$$
  $y = 1$ 



*x* = 1







Background

Relaxations

Abstractions

Cartesian Abstractions CEGAR

Summary



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:

- 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.
- 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.
- This allows local minimizations over linearly many edges instead of global minimization over exponentially many paths in the EVMDD when computing minimum costs.

→ polynomial in EVMDD size!

Backgroun

Compliation

Cartesian Abstractions

Summary

Not Cartesian!

REIBUR

If abstraction not Cartesian: two variables can be

- $\blacksquare$  independent in cost function ( $\leadsto$  compact EVMDD), but
- dependent in abstraction.
- → cannot consider independent parts of EVMDD separately.

Backgroun

Compilation

Relaxations

Abstractions Cartesian

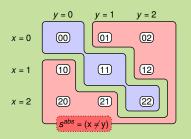
Abstractions CEGAR

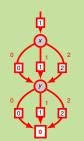
Summary

- independent in cost function (~> compact EVMDD), but
- dependent in abstraction.
- → cannot consider independent parts of EVMDD separately.

# Example (Non-Cartesian abstraction)

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





5E

Compilation

Relaxations

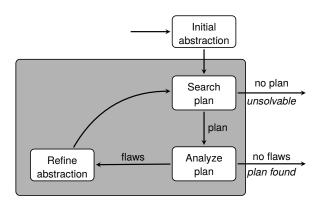
Abstractions Cartesian

Cartesian Abstractions CEGAR

Outilitiary



Wanted: principled way of computing Cartesian abstractions.



Backgroun

Relaxations

Abstractions

Cartesian Abstractions

Summary

#### CEGAR and Cartesian Abstractions



#### Assume the following:

- Initial abstraction is one-state abstraction with single abstract state  $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ .
- Each refinement step takes one abstract state  $s^{abs} = D_1 \times \cdots \times D_n$ , one variable  $v_i$ , and splits  $s^{abs}$  into

$$\square$$
  $D_1 \times \cdots \times D_{i-1} \times D'_i \times D_{i+1} \times \cdots \times D_n$ 

$$\square$$
  $D_1 \times \cdots \times D_{i-1} \times D_i'' \times D_{i+1} \times \cdots \times D_n$ 

such that  $D_i' \cap D_i'' = \emptyset$  and  $D_i' \cup D_i'' = D_i$ .

→ still a Cartesian abstraction

#### So, inductively:

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

Backgroun

Compliation

Abstractions Cartesian

CEGAR

Summary

#### **CEGAR** and Cartesian Abstractions



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.)

Backgroun

Compilatio

Relaxations

Abstractions

Abstractions CEGAR

Summary



#### Some questions:

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

Backgroun

Compliation

neia

Abstractions

CEGAR

Summary



# UNI

#### Example (one package, one truck)

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

- $V = \{t, p\}$  with
  - $\mathcal{D}_t = \{L, R\}$
  - $\square \mathscr{D}_p = \{L, T, R\}$
- $I = \{t \mapsto L, p \mapsto L\}$
- $O = \{ pick-in_i \mid i \in \{L,R\} \}$ 
  - $\cup \{drop\text{-}in_i \mid i \in \{L,R\}\}$
  - $\cup \{move_{i,j} \mid i,j \in \{L,R\}, i \neq j\}, \text{ where }$
  - $\blacksquare$  pick-in<sub>i</sub> =  $\langle t = i \land p = i, p := T \rangle$
  - $\blacksquare$  drop-in<sub>i</sub> =  $\langle t = i \land p = T, p := i \rangle$
  - $\blacksquare$  move<sub>i,j</sub> =  $\langle t = i, t := j \rangle$
- $\gamma = (p = R).$

Backgroun

110100

Cartesian Abstractions

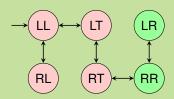
CEGAR



NE NE

#### Example (Ctd.)

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



Background

Relaxations

Abstractions

Cartesian Abstractions

Summary



## Example (Ctd.)

Refinement step 0 (initial abstraction):



Abstract plan:  $\pi_0 = \langle \rangle$ 

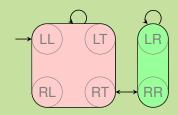
Flaw:  $s_0 = LL$  is not a goal state.

Consequence: Split abstract state  $\{L,R\} \times \{L,T,R\}$  wrt. goal condition  $\gamma = (p = R)$  into goal states and non-goal states.



#### Example (Ctd.)

#### Refinement step 1:



Abstract plan:  $\pi_1 = \langle drop - in_R \rangle$ 

Flaw: Preconditions (t = R) and (p = T) of  $drop-in_R$  not satisfied in  $s_0 = LL$ .

Consequence: Pick one of the unsatisfied preconditions, say (t = R), and split abstract state  $\{L, R\} \times \{L, T\}$  wrt. (t = R).

Background

Compliation

Relaxations

Abstractions
Cartesian
Abstractions

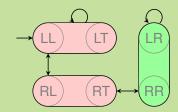
CEGAR

Defense



## Example (Ctd.)

#### Refinement step 2:



Abstract plan:  $\pi_2 = \langle move_{L,R}, drop-in_R \rangle$ 

Flaw: Precondition (p = T) of *drop-in*<sub>R</sub> not satisfied in  $s_1 = RL$ .

Consequence: Split abstract state  $\{R\} \times \{L, T\}$  wrt. (p = T).

Background

Compliation

Relaxations

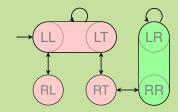
Abstractions
Cartesian
Abstractions

CEGAR Summary



# Example (Ctd.)

#### Refinement step 3:



Abstract plan:  $\pi_3 = \langle move_{L,R}, drop-in_R \rangle$ 

Flaw: Concrete and abstract paths diverge ( $\{R\} \times \{L\}$  vs.  $\{R\} \times \{T\}$ ).

Consequence: Regress from  $\{R\} \times \{T\}$  through  $move_{L,R}$ , obtain  $\{L\} \times \{T\}$ , split abstract state  $\{L\} \times \{L,T\}$  accordingly.

Backgroun

Compilation

Relaxations

Abstractions
Cartesian
Abstractions
CEGAR

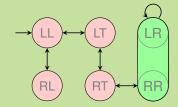
Summary



# FREIB

## Example (Ctd.)

#### Refinement step 4:



Abstract plan:  $\pi_4 = \langle pick-in_L, move_{L,R}, drop-in_R \rangle$ 

Flaw: None.  $\pi_4$  is concretizable!

Consequence: Concrete plan found! Return and terminate.

Background

Compliation

Relaxations

Abstractions Cartesian

Abstractions CEGAR

Deference



#### 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.)

Backgroun

Compliation

Abetractions

Cartesian

Summary

#### **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.

Backgroun

Compilation

Relaxations

Abstractions

Cartesian Abstractions CEGAR

Summary

#### **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  $\chi$  is violated, into (a) part containing  $s_{i-1}$ , but no concrete state in which precondition  $\chi$  is satisfied, and (b) rest.

Backgroun

Compilation

Relaxations

Abstractions

Cartesian

CEGAR

- -

#### **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.

Backgroun

Compilation

Relaxations

Abstractions

Abstractions CEGAR

Outilitiary



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.

Backgroun

.

Abstractions

Cartesian Abstractions CEGAR

. .



# E .

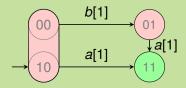
### Example (Cost-mismatch flaw)

$$a = \langle \top, x \wedge y \rangle, cost_a = 2x + 1$$

$$b = \langle \top, \neg x \wedge y \rangle, cost_b = 1$$

$$s_0 = 10$$

$$s_{\star} = x \wedge y$$



Background

Compliation

Relaxations

Abstractions
Cartesian
Abstractions

Abstractions CEGAR



# Background

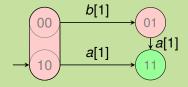
#### Example (Cost-mismatch flaw)

$$a = \langle \top, x \wedge y \rangle, cost_a = 2x + 1$$

$$s_0 = 10$$

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

$$s_{\star} = x \wedge y$$



Optimal abstract plan: \( \alpha \) (abstract cost 1)

Backgrour

Relaxations

Abstractions

Cartesian Abstractions

Summary

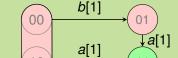


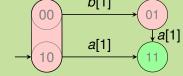
# Example (Cost-mismatch flaw)

$$a = \langle \top, x \wedge y \rangle, \cos t_a = 2x + 1$$

$$s_0 = 10$$
  
 $s_{\star} = x \wedge y$ 

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





- Optimal abstract plan:  $\langle a \rangle$  (abstract cost 1)
- This is also a concrete plan (concrete cost  $3 \neq 1$ )  $\rightsquigarrow$  split  $\{0,1\} \times \{0\}$

Background

Relaxations

Abstractions

CEGAR



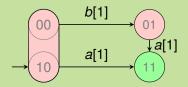
## Example (Cost-mismatch flaw)

$$a = \langle \top, x \wedge y \rangle, cost_a = 2x + 1$$

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

$$s_0 = 10$$

$$s_{\star} = x \wedge y$$



- Optimal abstract plan: (a) (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)

Background

Compilation

Relaxations

Abstractions
Cartesian
Abstractions

CEGAR

Deference



Summary

Background

Compilation Relaxations

Abstractions

Summary

# SDAC Planning and EVMDDs

Conclusion

# UNI

#### 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.
- $\blacksquare$  For  $h^{add}$ , both approaches give the same heuristic values.
- Abstraction heuristics can also be generalized to state-dependent action costs.

Backgroun

Joniphanon

Abstractions

Summarv

References

December 18th, 2017

# SDAC Planning and EVMDDs

Conclusion



#### 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.

Backgroun

Relaxations

Abstractions

Summary



FREIBU

References

Background

Compilation Relaxations

Abstractions

Summary

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.

Backgroun

Relaxations

Abstractions

Summary