Planning with State-Dependent Action Costs
ICAPS 2016 Tutorial

Robert Mattmüller    Florian Geißer
June 13, 2016
Part I

Theory
Section

Background
What are State-Dependent Action Costs?

Action costs:
- unit
- constant
- state-dependent

\[
\text{cost} \left( \text{flyTo} (\text{London}) \right) = |x_{\text{London}} - x_{\text{current}}| + |y_{\text{London}} - y_{\text{current}}|.
\]
What are State-Dependent Action Costs?

Action costs: \textcolor{red}{\textbf{unit}} \quad \textcolor{red}{\textbf{constant}} \quad \textcolor{red}{\textbf{state-dependent}}

\[
\begin{align*}
\text{cost}(\text{fly}(\text{Madrid}, \text{London})) &= 1, \\
\text{cost}(\text{fly}(\text{Paris}, \text{London})) &= 1, \\
\text{cost}(\text{fly}(\text{Freiburg}, \text{London})) &= 1, \\
\text{cost}(\text{fly}(\text{Istanbul}, \text{London})) &= 1.
\end{align*}
\]
What are State-Dependent Action Costs?

Action costs: unit constant state-dependent

\[
\begin{align*}
\text{cost}(\text{fly}(\text{Madrid}, \text{London})) &= 14, \\
\text{cost}(\text{fly}(\text{Paris}, \text{London})) &= 5, \\
\text{cost}(\text{fly}(\text{Freiburg}, \text{London})) &= 10, \\
\text{cost}(\text{fly}(\text{Istanbul}, \text{London})) &= 32.
\end{align*}
\]
What are State-Dependent Action Costs?

Action costs:  
\[ \text{cost}(\text{flyTo}(\text{London})) = |x_{\text{London}} - x_{\text{current}}| + |y_{\text{London}} - y_{\text{current}}| \]
\[ = |x_{\text{current}}| + |y_{\text{current}}|. \]
Why Study State-Dependent Action Costs?

- **Human perspective:**
  - “natural” and “elegant”
  - *modeler-friendly* \(\sim\) less error-prone?

- **Machine perspective:**
  - more *structured* \(\sim\) exploit in algorithms?
  - fewer redundancies, exponentially more *compact*

- **Language support:**
  - numeric PDDL, PDDL 3
  - RDDL, MDPs (state-dependent rewards!)

- **Applications:**
  - modeling *preferences and soft goals*
  - PSR domain

*(Abbreviation: SDAC = state-dependent action costs)*
Handling State-Dependent Action Costs

Good news:
- Computing $g$ values in forward search still easy.

Challenge:
- But what about SDAC-aware $h$ values?
- Or can we simply compile SDAC away?

This tutorial:
- Proposed answers to these challenges.
Handling State-Dependent Action Costs

Roadmap:

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

### Running Example

#### Example (Household domain)

**Actions:**

- $\text{vacuumFloor} = \langle \top, \text{floorClean}\rangle$
- $\text{washDishes} = \langle \top, \text{dishesClean}\rangle$
- $\text{doHousework} = \langle \top, \text{floorClean} \land \text{dishesClean}\rangle$

**Cost functions:**

- $\text{cost}_{\text{vacuumFloor}} = [\neg \text{floorClean}] \cdot 2$
- $\text{cost}_{\text{washDishes}} = [\neg \text{dishesClean}] \cdot (1 + 2 \cdot [\neg \text{haveDishwasher}])$
- $\text{cost}_{\text{doHousework}} = \text{cost}_{\text{vacuumFloor}} + \text{cost}_{\text{washDishes}}$
State-Dependent Action Costs

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)
State-Dependent Action Costs

Compilation I: “Parallel Action Decomposition”

Example

dishesClean, haveDishwasher: 0

dishesClean, ¬haveDishwasher: 0

¬dishesClean, haveDishwasher: 1

¬dishesClean, ¬haveDishwasher: 3

washDishes( dC, hD) = ⟨ dC ∧ hD, dC⟩, cost = 0

washDishes( dC, ¬hD) = ⟨ dC ∧ ¬hD, dC⟩, cost = 0

washDishes( ¬dC, hD) = ⟨¬dC ∧ hD, dC⟩, cost = 1

washDishes( ¬dC, ¬hD) = ⟨¬dC ∧ ¬hD, dC⟩, cost = 3
State-Dependent Action Costs

Compilation I: “Parallel Action Decomposition”

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
Assume we own a dishwasher:

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

\[
\begin{align*}
\text{doHousework}_1(\text{fC}) = \langle \text{fC, fC} \rangle, & \quad cost = 0 \\
\text{doHousework}_1(\neg \text{fC}) = \langle \neg \text{fC, fC} \rangle, & \quad cost = 2 \\
\text{doHousework}_2(\text{dC}) = \langle \text{dC, dC} \rangle, & \quad cost = 0 \\
\text{doHousework}_2(\neg \text{dC}) = \langle \neg \text{dC, dC} \rangle, & \quad cost = 1
\end{align*}
\]
Compilation II: “Purely Sequential Action Decomposition”

If costs **additively decomposable:**
- high-level actions \( \approx \) **macro actions**
- decompose into **sequential micro actions**

**Properties:**
- ✔️ linear blow-up
- ✗ not always possible
  - plan lengths not preserved, costs preserved
  - blow-up in search space \( \leadsto \) action ordering!
  - attention: all partial effects at end!

**Question:** Can this **always work** (kind of)? \( \leadsto \) Compilation III
State-Dependent Action Costs
Compilation III: “EVMDD-Based Action Decomposition”

Example

\[
\text{cost}_{\text{doHousework}} = \left[\neg\text{floorClean}\right] \cdot 2 + \left[\neg\text{dishesClean}\right] \cdot (1 + 2 \cdot \left[\neg\text{haveDishwasher}\right])
\]

Simplify right-hand part of diagram:

- Branch over single variable at a time.
- Exploit: \text{haveDishwasher} irrelevant if \text{dishesClean} is true.
State-Dependent Action Costs

Compilation III: “EVMDD-Based Action Decomposition”

Example (ctd.)

Later:
- Compiled actions
- Auxiliary variables to enforce action ordering
State-Dependent Action Costs
Compilation III: “EVMDD-Based Action Decomposition”

Compilation III

- exploit as much additive decomposability as possible
- multiply out variable domains where inevitable
- Technicalities:
  - fix variable ordering
  - perform Shannon and isomorphism reduction

Properties:

- always possible
- worst-case exponential blow-up, but as good as it gets
- plan lengths not preserved, costs preserved
- as before: action ordering, all partial effects at end!
Compilation III provides **optimal** combination of sequential and parallel action decomposition, given fixed variable ordering.

**Question**: How to find such decompositions **automatically**?

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

[Lai et al., 1996; Ciardo and Siminiceanu, 2002]
EVMDDs:  
- 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” cases  

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
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)
EVMDDs
edge-valued multi-valued decision diagrams

Example (EVMDD Evaluation)

\[ \text{cost}_a = xy^2 + z + 2 \]

\[ D_x = D_z = \{0, 1\}, \quad D_y = \{0, 1, 2\} \]

- Directed acyclic graph
- Dangling incoming edge
- Single terminal node 0
- Decision nodes with:
  - decision variables
  - edge label
  - edge weights
EVMDDs

Edge-Valued Multi-Valued Decision Diagrams

Example (EVMDD Evaluation)

\[\text{cost}_a = xy^2 + z + 2\]

\[D_x = D_z = \{0, 1\}, \quad D_y = \{0, 1, 2\}\]

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

\[\text{cost}_a(s) = \]
EVMDDs

Edge-Valued Multi-Valued Decision Diagrams

Example (EVMDD Evaluation)

\[ \text{cost}_a = xy^2 + z + 2 \]

\[ D_x = D_z = \{0, 1\}, \; D_y = \{0, 1, 2\} \]

\[ s = \{x \mapsto 1, \; y \mapsto 2, \; z \mapsto 0\} \]

\[ \text{cost}_a(s) = 2 + \]
EVMDDs

Edge-Valued Multi-Valued Decision Diagrams

Example (EVMDD Evaluation)

\[\text{cost}_a = xy^2 + z + 2\]

\[D_x = D_z = \{0, 1\}, \quad D_y = \{0, 1, 2\}\]

\[s = \{x \mapsto 1, \quad y \mapsto 2, \quad z \mapsto 0\}\]

\[\text{cost}_a(s) = 2 + 0 +\]
EVMDDs
Edge-Valued Multi-Valued Decision Diagrams

Example (EVMDD Evaluation)

\[ \text{cost}_a = xy^2 + z + 2 \]

\[ D_x = D_z = \{0, 1\}, \quad D_y = \{0, 1, 2\} \]

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

\[ \text{cost}_a(s) = 2 + 0 + 4 + \]
EVMDDs
Edge-Valued Multi-Valued Decision Diagrams

Example (EVMDD Evaluation)

\[ \text{cost}_a = xy^2 + z + 2 \]

\[ D_x = D_z = \{0, 1\}, \quad D_y = \{0, 1, 2\} \]

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

\[ \text{cost}_a(s) = 2 + 0 + 4 + 0 = 6 \]
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)
EVMDDs

Arithmetic operations on EVMDDs

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

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

- **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
Section

Compilation
Example (EVMDD-based action compilation)

Let $a = \langle \text{pre}, \text{eff} \rangle$, $\text{cost}_a = xy^2 + z + 2$.

Auxiliary variables:

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

Replace $a$ by new auxiliary actions (similarly for other actions).
Example (EVMDD-based action compilation, ctd.)

\[ a^{pre} = \langle pre \land \sigma = 0 \land \alpha = 0, \sigma = 1 \land \alpha = 1 \rangle, \quad \text{cost} = 2 \]

\[ a^{1,x=0} = \langle \alpha = 1 \land x = 0, \alpha = 3 \rangle, \quad \text{cost} = 0 \]

\[ a^{1,x=1} = \langle \alpha = 1 \land x = 1, \alpha = 2 \rangle, \quad \text{cost} = 0 \]

\[ a^{2,y=0} = \langle \alpha = 2 \land y = 0, \alpha = 3 \rangle, \quad \text{cost} = 0 \]

\[ a^{2,y=1} = \langle \alpha = 2 \land y = 1, \alpha = 3 \rangle, \quad \text{cost} = 1 \]

\[ a^{2,y=2} = \langle \alpha = 2 \land y = 2, \alpha = 3 \rangle, \quad \text{cost} = 4 \]

\[ a^{3,z=0} = \langle \alpha = 3 \land z = 0, \alpha = 4 \rangle, \quad \text{cost} = 0 \]

\[ a^{3,z=1} = \langle \alpha = 3 \land z = 1, \alpha = 4 \rangle, \quad \text{cost} = 1 \]

\[ a^{\text{eff}} = \langle \alpha = 4, \text{eff} \land \sigma = 0 \land \alpha = 0 \rangle, \quad \text{cost} = 0 \]
Let $\Pi$ be an SDAC-task and $\Pi'$ the result of EVMDD-based action compilation applied to $\Pi$.

**Proposition**

$\Pi'$ has only state-independent costs.

**Proposition**

Size of $\Pi'$ is polynomial in size of $\Pi$ times size of largest EVMDD used in compilation.

**Proposition**

$\Pi$ and $\Pi'$ admit the same plans (modulo replacement of actions by action sequences). Optimal plan costs are preserved.
Section

Relaxations
Relaxation Heuristics

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

Question: Also informative in SDAC planning?
Relaxation Heuristics

Definition (Classical additive heuristic $h^{add}$)

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

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

**Question:** How to generalize $h^{add}$ to SDAC?
Relaxations with SDAC

Example

\[
a = \langle \top, x = 1 \rangle \quad \text{cost}_a = 2 - 2y \\
b = \langle \top, y = 1 \rangle \quad \text{cost}_b = 1
\]

\[
s = \{ x \mapsto 0, y \mapsto 0 \}
\]

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

\[
h_s^{\text{add}}(x = 1) = ?
\]
Relaxations with SDAC

Example

\[ a = \langle \top, x = 1 \rangle \quad \text{cost}_a = 2 - 2y \]
\[ b = \langle \top, y = 1 \rangle \quad \text{cost}_b = 1 \]

\[ s = \{ x \mapsto 0, y \mapsto 0 \} \]
\[ h^\text{add}_s (y = 1) = 1 \]
\[ h^\text{add}_s (x = 1) = ? \]
Relaxations with SDAC

Example

\[ a = \langle \top, x = 1 \rangle \quad \text{cost}_a = 2 - 2y \]
\[ b = \langle \top, y = 1 \rangle \quad \text{cost}_b = 1 \]

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

\[ a : 2 \]
\[ b : 1 \]
\[ \Rightarrow \text{cheaper!} \]
Relaxations with SDAC

Minimize over all situations where $a$ is applicable.

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

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

Minimize over all situations where \( a \) is applicable.

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

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

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

\( S_a \): set of partial states over variables in cost function

\(|S_a|\) exponential in number of variables in cost function
Relaxations with SDAC

Properties of $h^{add}$ for SDAC:

- **Good**: classical $h^{add}$ on compiled task $=$
  generalized $h^{add}$ on SDAC-task
- **Bad**: exponential blow-up

Computing $h^{add}$ for SDAC:

- **Option 1**: Compute classical $h^{add}$ on compiled task.
- **Option 2**: Compute $Cost^s_a$ directly.
  - Plug EVMDDs as subgraphs into RPG
  - $\Rightarrow$ efficient computation of $h^{add}$
Option 2: RPG Compilation

\[ cost_a = x y^2 + z + 2 \]
Option 2: RPG Compilation

- variable nodes become $\lor$-nodes
- weights become $\land$-nodes
Option 2: RPG Compilation

Input

\[
\begin{align*}
&x = 0 \quad x = 1 \\
&y = 0 \quad y = 1 \quad y = 2 \\
&z = 0 \quad z = 1
\end{align*}
\]

Augment with input nodes

Output

\[
\begin{align*}
&\land +0 \\
&\land +0 \\
&\land +0 \land +1 \land +4 \\
&\lor \land +1 \\
&\lor \land + 0
\end{align*}
\]
Option 2: RPG Compilation

Ensure complete evaluation

Input

x=0 x=1 y=0 y=1 y=2 z=0 z=1

0, Output

x

y

z

Ensure complete evaluation
Option 2: Computing $Cost^S_a$

Input

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>0</th>
<th>6</th>
<th>$\infty$</th>
<th>1</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$x=0$</td>
<td>$x=1$</td>
<td>$y=0$</td>
<td>$y=1$</td>
<td>$y=2$</td>
<td>$z=0$</td>
<td>$z=1$</td>
</tr>
</tbody>
</table>

Graph

- Insert $h^{add}$ values

Output
Option 2: Computing $Cost^S_a$

Evaluate nodes:
- $\land$: $\sum (\text{parents}) + \text{weight}$
- $\lor$: $\min (\text{parents})$

Input:
- $x=0$: 10
- $x=1$: 0
- $y=0$: 6
- $y=1$: $\infty$
- $y=2$: 1
- $z=0$: 2
- $z=1$: 2

Output:
- $x=0$, $y=0$, $z=0$: 0
- $x=1$, $y=0$, $z=0$: 0
- $x=1$, $y=0$, $z=1$: 1
- $x=1$, $y=1$, $z=0$: 2
- $x=1$, $y=1$, $z=1$: 2

Evaluate nodes:
- $\land$: $\sum (\text{parents}) + \text{weight}$
- $\lor$: $\min (\text{parents})$
Option 2: Computing $Cost^S_a$

Evaluate nodes:
- $\land$: $\sum$ (parents) + weight
- $\lor$: min (parents)
Option 2: Computing $Cost^S_a$

Evaluate nodes:
- $\wedge$: $\sum$(parents) + weight
- $\vee$: $\min$(parents)
Option 2: Computing $\text{Cost}^S_a$

Evaluate nodes:

- $\land$: $\sum(\text{parents}) + \text{weight}$
- $\lor$: $\min(\text{parents})$
Option 2: Computing $Cost_{a}^{S}$

Evaluate nodes:
- $\land$: $\sum(\text{parents}) + \text{weight}$
- $\lor$: $\min(\text{parents})$

Input:
- $x=0$, Cost 10
- $x=1$, Cost 0
- $y=0$, Cost 6
- $y=1$, Cost $\infty$
- $y=2$, Cost 1
- $z=0$, Cost 2
- $z=1$, Cost 2

Output:
- $x=0$, Cost 0
- $x=1$, Cost 10
- $y=0$, Cost 6
- $y=1$, Cost 1
- $y=2$, Cost $\infty$
- $z=0$, Cost 2
- $z=1$, Cost 2

June 13, 2016 Robert Mattmüller, Florian Geißer – Planning with State-Dependent Action Costs
Option 2: Computing $Cost^S_a$

Evaluate nodes:
- $\land$: $\sum$(parents) + weight
- $\lor$: $\min$(parents)
Option 2: Computing $Cost^S_a$

Evaluate nodes:
- $\wedge$: $\sum(\text{parents}) + \text{weight}$
- $\lor$: $\min(\text{parents})$
Option 2: Computing $Cost^S_a$

Evaluate nodes:
- $\land$: $\sum$ (parents) + weight
- $\lor$: min (parents)
Option 2: Computing $Cost_a^S$

Evaluate nodes:
- $\wedge$: $\sum(\text{parents}) + \text{weight}$
- $\lor$: $\min(\text{parents})$

Input:
- $x = 0$, $y = 0$, $z = 0$
- $x = 1$, $y = 1$, $z = 0$
- $\infty$, $y = 2$, $z = 1$

Output:
- $0$, $10$, $y = 0$
- $1$, $2$, $z = 1$
- $2$, $2$, $z = 1$

Graph:
- Nodes with $+$ for sum operation
- Nodes with $\lor$ for min operation
- Nodes with $\wedge$ for sum operation

Diagram:
- Input nodes with values $x = 0, y = 0, z = 0$
- Output nodes with values $0, 10, y = 0$
- Nodes with $\wedge$, $\lor$, and $+$ operations
Option 2: Computing $\text{Cost}_a^S$

\[
\text{Cost}_a^S = \min_{\hat{s} \in S_a} [\text{cost}_a(\hat{s}) + h_{S}^{\text{add}}(\hat{s})]
\]
Option 2: Computing $\text{Cost}^S_a$

$\text{Cost}^S_a = \min_{\hat{s} \in S_a} [\text{cost}_a(\hat{s}) + h^{add}(\hat{s})]$  

$\text{cost}_a = xy^2 + z + 2$  

$\hat{s} = \{x \mapsto 1, y \mapsto 2, z \mapsto 0\}$
Option 2: Computing $Cost^s_a$

\[ Cost^s_a = \min_{\hat{s} \in S_a} \left[ cost_a(\hat{s}) + h_{s}^{add}(\hat{s}) \right] \]

\[ cost_a = xy^2 + z + 2 \]

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

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

\[ = 2 + 0 + 4 + 0 \]

\[ h_{s}^{add}(\hat{s}) = 0 + 1 + 2 = 3 \]
Option 2: Computing $\text{Cost}^s_a$

$\text{Cost}^s_a = \min_{\hat{s} \in S_a} [\text{cost}_a(\hat{s}) + h_{s}^{\text{add}}(\hat{s})]$ 

- $\text{cost}_a = xy^2 + z + 2$
- $\hat{s} = \{x \mapsto 1, y \mapsto 2, z \mapsto 0\}$

- $\text{cost}_a(\hat{s}) = 1 \cdot 2^2 + 0 + 2 = 6$
  
  $= 2 + 0 + 4 + 0$

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

- $\text{Cost}^s_a = 6 + 3 = 9$
Additive Heuristic

RPG compilation:

- RPG subgraph in each layer for each action
- Connect subgraphs with precondition graphs
- Link outputs to next proposition layer

- **Good:** classical $h^{add}$ on compiled task $=$
generalized $h^{add}$ on SDAC-task $=$
cost value computed using RPG compilation
Section

Abstractions
Abstraction Heuristics

**Question:** Why consider abstraction heuristics?

**Answer:**
- admissibility
- $\leadsto$ optimality
Abstraction Heuristics

For admissibility, abstract cost of $a$ should be $\text{cost}_a(s_{\text{abs}}) = \min_{\text{concrete state } s} \text{cost}_a(s)$.

Problem: exponentially many states in minimization

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

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

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

$$\text{cost}_a(s^{\text{abs}}) = \min_{\text{concrete state } s} \text{cost}_a(s).$$
Question: What are the abstract action costs?

Answer: For admissibility, abstract cost of \( a \) should be

\[
\text{cost}_a(s^{\text{abs}}) = \min_{\text{concrete state } s} \text{cost}_a(s).
\]

Problem: exponentially many states in minimization

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

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

(Includes projections and domain abstractions.)
Cartesian Abstractions

We will see: possible if the abstraction is Cartesian or coarser.
(Includes projections and domain abstractions.)

**Definition (Cartesian abstraction)**

A set of states \( s^{\text{abs}} \) is **Cartesian** if it is of the form

\[
D_1 \times \cdots \times D_n,
\]

where \( D_i \subseteq D_i \) for all \( i = 1, \ldots, n \).

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

[Seipp and Helmert, 2013]

**Intuition:** Variables are abstracted independently.

\( \rightsquigarrow \) exploit independence when computing abstract costs!
Example (Cartesian abstraction)

**Cartesian abstraction over** \( x, y \)

\[
\begin{array}{c|ccc}
 & y = 0 & y = 1 & y = 2 \\
\hline
x = 0 & 00 & 01 & 02 \\
x = 1 & 10 & 11 & 12 \\
x = 2 & 20 & 21 & 22 \\
\end{array}
\]
Cartesian Abstractions

Example (Cartesian abstraction)

Cartesian abstraction over $x, y$

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

June 13, 2016 Robert Mattmüller, Florian Geißer – Planning with State-Dependent Action Costs
Cartesian Abstractions

Example (Cartesian abstraction)

Cartesian abstraction over \( x, y \)

\[
\begin{array}{ccc}
  x = 0 & y = 0 & 00 \\
  & & 01 \\
  & & 02 \\
  x = 1 & y = 1 & 10 \\
  & & 11 \\
  & & 12 \\
  x = 2 & y = 2 & 20 \\
  & & 21 \\
  & & 22 \\
\end{array}
\]

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

\[
\begin{align*}
  x & = 0 \\
  & 1 \\
  & 2 \\
  y & = 0 \\
  & 1 \\
  & 2 \\
\end{align*}
\]

\[
\begin{align*}
  cost & = 4 \\
  & 5
\end{align*}
\]
Cartesian Abstractions

Example (Cartesian abstraction)

Cartesian abstraction over $x$, $y$

Cost $x + y + 1$
(edges consistent with $s_{\text{abs}}$)

<table>
<thead>
<tr>
<th>$x = 0$</th>
<th>$y = 0$</th>
<th>$y = 1$</th>
<th>$y = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = 0$</td>
<td>00</td>
<td>01</td>
<td>02</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>$x = 2$</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>

$\text{cost} = 4$
$\text{cost} = 5$

$\min = 1$
Cartesian Abstractions

Example (Cartesian abstraction)

Cartesian abstraction over $x$, $y$

Cost $x + y + 1$
(edges consistent with $s_{\text{abs}}$)

$x = 0$
- $y = 0$: $00$
- $y = 1$: $01$
- $y = 2$: $02$

$x = 1$
- $y = 0$: $10$
- $y = 1$: $11$
- $y = 2$: $12$

$x = 2$
- $y = 0$: $20$
- $y = 1$: $21$
- $y = 2$: $22$

$cost = 4$

$min = 1$

$min = 3$
Cartesian Abstractions

Example (Cartesian abstraction)

Cartesian abstraction over $x, y$

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

$x = 0$

$y = 0$ 00
$y = 1$ 01
$y = 2$ 02

$x = 1$

$y = 0$ 10
$y = 1$ 11
$y = 2$ 12

$x = 2$

$y = 0$ 20
$y = 1$ 21
$y = 2$ 22

$cost = 4$

$cost = 5$

$min = 1$

$min = 3$

$min = 4$
Cartesian Abstractions

What happens here? or:

Why does the topsort EVMDD traversal correctly compute $cost_a(s_{\text{abs}})$?

1. For each Cartesian state $s_{\text{abs}}$ and each variable $x$, each value $d \in D_x$ is either consistent with $s_{\text{abs}}$ or not.

2. This implies: at all decision nodes associated with variable $x$, some outgoing edges are enabled, others are disabled.
   This is independent from all other decision nodes/variables.

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

$\Rightarrow$ polynomial in EVMDD size!

June 13, 2016 Robert Mattmüller, Florian Geißer – Planning with State-Dependent Action Costs
Cartesian Abstractions

Not Cartesian!

If abstraction not Cartesian: two variables can be
- **independent** in cost function (⇝ compact EVMDD), but
- **dependent** in abstraction.

⇝ cannot consider independent parts of the EVMDD separately.
Cartesian Abstractions

Not Cartesian!

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 the EVMDD separately.

Example (Non-Cartesian abstraction)

\(\text{cost} : x + y + 1, \text{cost}(s^{\text{abs}}) = 2, \text{local minim.}: 1 \rightsquigarrow \text{underestimate!}\)
Wanted: principled way of computing Cartesian abstractions.

Counterexample-Guided Abstraction Refinement (CEGAR)
Counterexample-Guided Abstraction Refinement

Possible flaws in abstract plan:

1. Concrete state does not fit abstract state (concrete and abstract traces diverge)
2. Action not applicable in concrete state
3. Trace completed, but goal not reached

Here, we need to consider a further type of flaw:

4. Cost-mismatch flaw: Action more costly in concrete state than in abstract state

⇝ resolve cost-mismatch flaws with additional refinement.
Possible flaws in abstract plan:

1. Concrete state does not fit abstract state (concrete and abstract traces diverge)
2. Action not applicable in concrete state
3. Trace completed, but goal not reached

Here, we need to consider a further type of flaw:

4. Cost-mismatch flaw: Action more costly in concrete state than in abstract state
### Counterexample-Guided Abstraction Refinement

#### Possible flaws in abstract plan:

1. Concrete state does not fit abstract state (concrete and abstract traces diverge)
2. Action not applicable in concrete state
3. Trace completed, but goal not reached

Here, we need to consider a further type of flaw:

4. **Cost-mismatch flaw:** Action more costly in concrete state than in abstract state

\[\Rightarrow\] resolve cost-mismatch flaws with additional refinement.
Counterexample-Guided Abstraction Refinement

Example (Cost-mismatch flaw)

\[ a = \langle \top, x \land y \rangle, \quad \text{cost}_a = 2x + 1 \]
\[ b = \langle \top, \neg x \land y \rangle, \quad \text{cost}_b = 1 \]
\[ s_0 = 10 \]
\[ s_* = x \land y \]
Counterexample-Guided Abstraction Refinement

Example (Cost-mismatch flaw)

\[ a = \langle \top, x \land y \rangle, \quad \text{cost}_a = 2x + 1 \quad \text{and} \quad s_0 = 10 \]
\[ b = \langle \top, \neg x \land y \rangle, \quad \text{cost}_b = 1 \quad \text{and} \quad s_\ast = x \land y \]

- **Optimal abstract plan:** \( \langle a \rangle \) (abstract cost 1)
Counterexample-Guided Abstraction Refinement

Example (Cost-mismatch flaw)

\[ a = \langle \top, x \land y \rangle, \quad \text{cost}_{a} = 2x + 1 \quad \quad s_{0} = 10 \]
\[ b = \langle \top, \neg x \land y \rangle, \quad \text{cost}_{b} = 1 \quad \quad s_{*} = x \land y \]

- **Optimal abstract plan:** \( \langle a \rangle \) (abstract cost 1)
- This is also a **concrete plan** (concrete cost 3)
Counterexample-Guided Abstraction Refinement

Example (Cost-mismatch flaw)

- \( b : 1 \)
- \( a : 1 \)

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

\[
b = \langle \top, \neg x \wedge y \rangle, \quad cost_b = 1 \quad s_* = x \wedge y
\]

- **Optimal abstract plan:** \( \langle a \rangle \) (abstract cost 1)
- This is also a **concrete plan** (concrete cost 3)
- **But optimal concrete plan:** \( \langle b, a \rangle \) (concr. and abstract cost 2)
Section

Summary
Summary: EVMDDs

- compact representation of cost functions
- exhibit additive structure

Recall: motivating challenges

- compiling SDAC away $\leadsto$ solved!
  - EVMDD-based action compilation
  - preserves $h^{add}$ and $h^{abs}$

- SDAC-aware $h$ values $\leadsto$ possible!
  - $h^{add}$
  - RPG embedding
  - Cartesian abstraction heuristics
Future Work:

- Other delete-relaxation heuristics such as $h^F$
- Static and dynamic EVMDD variable orders
Part II

Practice
Section

Libraries
EVMDD Libraries

MEDDLY

- **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](http://meddly.sourceforge.net)
EVMDD Libraries

pyevmdd

- **pyevmdd**: EVMDD library for Python
- **Authors**: RM and FG
- **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/
Section

PDDL
PDDL Representation

Usual way of representing costs in PDDL:

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

Custom syntax:

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

Cost of move increases if ball color differs from its room color

Goal did not change!

\[
\text{cost}(\text{move}) = \sum_{\text{room}} \sum_{\text{ball}} (\text{at}(\text{ball}, \text{room}) \land \text{red}(\text{ball}) \land \text{blue}(\text{room})) + \\
\sum_{\text{room}} \sum_{\text{ball}} (\text{at}(\text{ball}, \text{room}) \land \text{blue}(\text{ball}) \land \text{red}(\text{room}))
\]
Colored Gripper

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

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

\[
\text{cost}(\text{move}) = \sum_{\text{room}} \sum_{\text{ball}} (\text{at}(\text{ball}, \text{room}) \land (\text{red}(\text{ball})) \land (\text{blue}(\text{room}))) \\
+ \sum_{\text{room}} \sum_{\text{ball}} (\text{at}(\text{ball}, \text{room}) \land (\text{blue}(\text{ball})) \land (\text{red}(\text{room})))
\]
EVMDD-Based Action Compilation

Example (EVMDD-based action compilation)

Let $a = \langle \text{pre}, \text{eff} \rangle$, $\text{cost}_a = xy^2 + z + 2$.

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).
EVMDD-Based Action Compilation

Example (EVMDD-based action compilation, ctd.)

\[ a^{\text{pre}} = \langle \text{pre} \land \sigma = 0 \land \alpha = 0, \]
\[ \sigma = 1 \land \alpha = 1 \rangle, \quad \text{cost} = 2 \]
\[ a^{1,x=0} = \langle \alpha = 1 \land x = 0, \alpha = 3 \rangle, \quad \text{cost} = 0 \]
\[ a^{1,x=1} = \langle \alpha = 1 \land x = 1, \alpha = 2 \rangle, \quad \text{cost} = 0 \]
\[ a^{2,y=0} = \langle \alpha = 2 \land y = 0, \alpha = 3 \rangle, \quad \text{cost} = 0 \]
\[ a^{2,y=1} = \langle \alpha = 2 \land y = 1, \alpha = 3 \rangle, \quad \text{cost} = 1 \]
\[ a^{2,y=2} = \langle \alpha = 2 \land y = 2, \alpha = 3 \rangle, \quad \text{cost} = 4 \]
\[ a^{3,z=0} = \langle \alpha = 3 \land z = 0, \alpha = 4 \rangle, \quad \text{cost} = 0 \]
\[ a^{3,z=1} = \langle \alpha = 3 \land z = 1, \alpha = 4 \rangle, \quad \text{cost} = 1 \]
\[ a^{\text{eff}} = \langle \alpha = 4, \text{eff} \land \sigma = 0 \land \alpha = 0 \rangle, \quad \text{cost} = 0 \]
EVMDD-Based Action Compilation Tool

- **Disclaimer:**
  - Not completely functional
  - Still some bugs
- **Uses** `pyevmdd`
- **Language:** Python
- **License:** open source
- **Code:** [https://github.com/robertmattmueller/sdac-compiler](https://github.com/robertmattmueller/sdac-compiler)
Part III

Acknowledgements
Acknowledgements

Acknowledgements:

- **Christian Muise**, for taking the time to get our compiler running in the cloud.
- **Erik Wacker**, for working on the compiler.
- **Thomas Keller**, for doing all the research behind this tutorial with us.
Part IV

References
References I

Blai Bonet and Hector Geffner.
Planning as heuristic search: New results.

Blai Bonet, Gábor Loerincs, and Hector Geffner.
A robust and fast action selection mechanism for planning.

Junaid Badar and Andrew Miner.
MEDDLY: Multi-terminal and Edge-valued Decision Diagram Library.


