Principles of AI Planning
12. Planning with State-Dependent Action Costs

Bernhard Nebel and Robert Mattmüller
December 18th, 2017

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
  ~> this chapter

December 18th, 2017
B. Nebel, R. Mattmüller – AI Planning 3 / 80

Background

What are State-Dependent Action Costs?

- Action costs: unit, constant, state-dependent
- cost(fly(Madrid, London)) = 1,  cost(fly(Paris, London)) = 1,
  cost(fly(Freiburg, London)) = 1,  cost(fly(Istanbul, London)) = 1.

December 18th, 2017
B. Nebel, R. Mattmüller – AI Planning 4 / 80
Why Study State-Dependent Action Costs?

Background

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 \( \text{cost}_a \in \mathbb{N} \).
  - 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 \( \text{cost}_a : S \rightarrow \mathbb{N} \).
  - Example: Flying to a destination city costs amount proportional to distance, depending on the current city.

References


Handling State-Dependent Action Costs

Good news:
- Computing \( g \) values in forward search still easy.
  (When expanding state \( s \) with action \( a \), we know \( \text{cost}_a(s) \).)

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.

References

December 18th, 2017 B. Nebel, R. Mattmüller – AI Planning 7 / 80
State-Dependent Action Costs

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

Assumption: For each \( a \in O \), the set of variables occurring in the precondition of \( a \) is disjoint from the set of variables on which the cost function \( \text{cost}_a \) depends.

(Question: Why is this assumption unproblematic?)

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

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

Example (Household domain)
Actions:
- \( \text{vacuumFloor} = \langle \top, \text{floorClean} \rangle \)
- \( \text{washDishes} = \langle \top, \text{dishesClean} \rangle \)
- \( \text{doHousework} = \langle \top, \text{floorClean} \land \text{dishesClean} \rangle \)

Cost functions:
- \( \text{cost}_{\text{vacuumFloor}} = [\neg \text{floorClean}] \cdot 2 \)
- \( \text{cost}_{\text{washDishes}} = [\neg \text{dishesClean}] \cdot (1 + 2 \cdot [\neg \text{haveDishwasher}]) \)
- \( \text{cost}_{\text{doHousework}} = \text{cost}_{\text{vacuumFloor}} + \text{cost}_{\text{washDishes}} \)

(Question: How much can applying action \( \text{washDishes} \) cost?)

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”

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

State-Dependent Action Costs
Compilation II: “Purely Sequential Action Decomposition”

Example
Assume we own a dishwasher:
\[
\text{cost}_{\text{doHousework}} = 2 \cdot \neg \text{floorClean} + \neg \text{dishesClean}
\]

FloorClean: 0
DishesClean: 0

\[\neg \text{floorClean}: 2 \quad \neg \text{dishesClean}: 1\]

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 \(\text{doHousework}\) in the SDAC setting is replaced by an application of the action sequence
\[
\text{doHousework}_1(\neg fC), \text{doHousework}_2(\neg dC)
\]

in the compiled setting.
State-Dependent Action Costs
Compilation II: “Purely Sequential Action Decomposition”

Properties (ctd.):
- plan costs preserved
- blow-up in search space
  E.g., in a state where \( \neg fC \) and \( \neg dC \) hold, should we apply \( \text{doHousework}_1(\neg fC) \) or \( \text{doHousework}_2(\neg dC) \) first?
  \( \leadsto \) 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)? \( \leadsto \) Compilation III

State-Dependent Action Costs
Compilation III: “EVMDD-Based Action Decomposition”

Example

\[
\begin{align*}
\text{cost}_{\text{doHousework}} &= [\neg \text{floorClean}] \cdot 2 + \\
& \quad [\neg \text{dishesClean}] \cdot (1 + 2 \cdot [\neg \text{haveDishwasher}])
\end{align*}
\]

\[
\begin{tikzpicture}
  \node (f0) at (0,0) {\text{floorClean: 0}};
  \node (d0) at (1,0) {\text{dishesClean, haveDishwasher: 0}};
  \node (h0) at (2,0) {\text{~floorClean: 2}};
  \node (d1) at (1,-1) {\text{dishesClean, ~haveDishwasher: 1}};
  \node (d2) at (1,-2) {\text{dishesClean, ~haveDishwasher: 3}};
  \node (h1) at (2,-1) {\text{~dishesClean, haveDishwasher: 1}};
  \node (h2) at (2,-2) {\text{~dishesClean, ~haveDishwasher: 3}};

  \path[->] (f0) edge node[above] {\text{~floorClean: 2}} (h0);
  \path[->] (d0) edge node[above] {\text{~dishesClean: 1}} (h0);
  \path[->] (h0) edge node[above] {\text{~haveDishwasher: 2}} (h2);
  \path[->] (d1) edge node[above] {\text{~dishesClean: 1}} (h1);
  \path[->] (h1) edge node[above] {\text{~haveDishwasher: 2}} (h2);
\end{tikzpicture}
\]

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.)

\[
\begin{align*}
\text{floorClean: 0} & \quad \text{dishesClean: 0} \\
\text{~floorClean: 2} & \quad \text{~dishesClean: 1} \\
& \quad \text{~haveDishwasher: 2}
\end{align*}
\]

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 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!
State-Dependent Action Costs
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]

EVMDDs
Edge-Valued Multi-Valued Decision Diagrams

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

Properties:
✓ satisfy all requirements for Compilation III, even (almost) uniquely determined by them
✓ already have well-established theory and tool support
✓ detect and exhibit additive structure in arithmetic functions

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

Two perspectives on EVMDDs:
- graphs specifying how to decompose action costs
- data structures encoding action costs (used independently from compilations)

Example (EVMDD Evaluation)
\[ \text{cost}_a = xy^2 + z + 2 \]
\[ D_x = \{0, 1\}, \quad D_y = \{0, 1, 2\} \]

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

We see: \( z \) independent from rest, \( y \) only matters if \( x \neq 0 \).

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

EVMDDs
Arithmetic operations on EVMDDs

Given arithmetic operator $\otimes \in \{+,-,\cdot,\ldots\}$, EMVDDs $E_1, E_2$.
Compute EVMDD $E = E_1 \otimes E_2$.

Implementation: procedure $\text{apply}(\otimes, E_1, E_2)$:

- **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

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 $a = (\chi, e)$, $\text{cost}_a = xy^2 + z + 2$.

Auxiliary variables:

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

Replace $a$ by new auxiliary actions (similarly for other actions).
EVMDD-Based Action Compilation

Example (EVMDD-based action compilation, ctd.)

\[ a^x = (\chi \land \sigma = 0 \land \alpha = 0), \]
\[ \alpha = 0 \]
\[ a^1 x = 0 = (\alpha = 1 \land x = 0, \alpha : \geq 3), \quad \text{cost} = 2 \]
\[ a^2 x = 0 = (\alpha = 2 \land y = 0, \alpha : \geq 3), \quad \text{cost} = 0 \]
\[ a^3 x = 1 = (\alpha = 1 \land z = 1, \alpha : \geq 4), \quad \text{cost} = 1 \]
\[ a^4 = (\alpha = 4, \chi \land \sigma : 0 \land \alpha : 0), \quad \text{cost} = 0 \]

Proposition
\[ \Pi' \] has only state-independent costs.

Proof. By construction.

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
\[ \Pi' \] only state-independent costs.

Proof. By construction.

EVMDD-Based Action Compilation

Definition (EVMDD-based action compilation)

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

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

\[ V' = V \cup \{ \sigma \} \cup \{ \alpha_a | a \in O \}, \]
\[ I' = I \cup \{ \sigma \mapsto 0 \} \cup \{ \alpha_a \mapsto 0 | a \in O \}, \]
\[ O' = \bigcup_{a \in O} EAC(a), \text{ and} \]
\[ \gamma' = \gamma \land (\sigma = 0) \land \bigwedge_{a \in O}(\alpha_a = 0). \]
EVMDD-Based Action Compilation

Proof (ctd.)

By construction, \( \text{cost}(a_0^n) + \cdots + \text{cost}(a_i^n) = \text{cost}_a(s_{i-1}) \).

Moreover, the sequence \( a_0, \ldots, a_i \) is applicable in \( s_{i-1} \cup \{ \sigma \mapsto 0 \} \cup \{ \alpha_a \mapsto \alpha_a \} \) and leads to \( s_i \cup \{ \sigma \mapsto 0 \} \cup \{ \alpha_a \mapsto \alpha_a \} \).

Therefore, by induction, \( \pi' = a_0^n, a_1^n, \ldots, a_{i-1}^n, a_i^n \) is applicable in \( s_0 \cup \{ \sigma \mapsto 0 \} \cup \{ \alpha_a \mapsto \alpha_a \} \) (and leads to a goal state). Moreover, \( \text{cost}(\pi') = \text{cost}(a_0^n) + \cdots + \text{cost}(a_i^n) = \text{cost}_a(s_0) + \cdots + \text{cost}_a(s_{i-1}) = \text{cost}(\pi) \).

Still to show: \( \Gamma' \) 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.

December 18th, 2017  B. Nebel, R. Mattmüller – AI Planning

Example (Ctd.)

Compilation of \( a \):

\[
\begin{align*}
 a^x &= (u = 0 \land \sigma = 0 \land \alpha_a = 0, \\
 \alpha_a &= 0, \\
 a^1, x = 0 &= (\alpha_a = 1 \land x = 0, \alpha_a = 3), \\
 \alpha_a &= 1, \\
 a^1, x = 1 &= (\alpha_a = 1 \land x = 1, \alpha_a = 2), \\
 \alpha_a &= 1, \\
 a^2, y = 0 &= (\alpha_a = 2 \land y = 0, \alpha_a = 3), \\
 \alpha_a &= 2, \\
 a^2, y = 1 &= (\alpha_a = 2 \land y = 1, \alpha_a = 3), \\
 \alpha_a &= 2, \\
 a^2, y = 2 &= (\alpha_a = 2 \land y = 2, \alpha_a = 3), \\
 \alpha_a &= 2, \\
 a^3, z = 0 &= (\alpha_a = 3 \land z = 0, \alpha_a = 4), \\
 \alpha_a &= 3, \\
 a^3, z = 1 &= (\alpha_a = 3 \land z = 1, \alpha_a = 4), \\
 \alpha_a &= 3, \\
 a^3, z = 2 &= (\alpha_a = 3 \land z = 2, \alpha_a = 4), \\
 \alpha_a &= 3, \\
 a^4 &= (\alpha_a = 4, u = 1 \land \sigma = 0 \land \alpha_a = 0), \\
 \alpha_a &= 4 \\
\end{align*}
\]

\( \text{cost} = 2 \)

Example

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

\[
\begin{align*}
 a &= (u = 0, u = : 1), \\
 b &= (u = 1, u = : 2), \\
 \text{cost}_a &= xy^2 + z + 2, \\
 \text{cost}_b &= z + 1.
\end{align*}
\]

Optimal plan for \( \Pi \):

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

December 18th, 2017  B. Nebel, R. Mattmüller – AI Planning

Example (Ctd.)

Compilation of \( b \):

\[
\begin{align*}
 b^x &= (u = 1 \land \sigma = 0 \land \alpha_b = 0, \\
 \alpha_b &= 0, \\
 b^1, z = 0 &= (\alpha_b = 1 \land z = 0, \alpha_b = 2), \\
 \alpha_b &= 1, \\
 b^1, z = 1 &= (\alpha_b = 1 \land z = 1, \alpha_b = 2), \\
 \alpha_b &= 1, \\
 b^2 &= (\alpha_b = 2, u = 2 \land \sigma = 0 \land \alpha_b = 0), \\
 \alpha_b &= 2 \\
\end{align*}
\]

\( \text{cost} = 1 \)

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

\[
\pi' = a^x, a^1, x = 1, a^2, y = 2, a^3, z = 0, a^4, b^x, b^1, z = 0, b^2.
\]

\( \text{cost} = 2 + 0 + 4 + 0 + 0 = 6 \)

December 18th, 2017  B. Nebel, R. Mattmüller – AI Planning
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.

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

Question: Are they also informative in SDAC planning?

Assume we want to compute the additive heuristic $h_{add}$ in a task with state-dependent action costs.

But what does an action $a$ cost in a relaxed state $s^*$?

And how to compute that cost?
Relaxed SAS$^+$ Tasks

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

- Operators are already in effect normal form.
- We do not need to impose a positive normal form, because all conditions are conjunctions of facts, and facts are just variable-value pairs and hence always positive.
- Hence $a^+ = a$ for any operator $a$, and $\Pi^+ = \Pi$.
- For simplicity, we identify relaxed states as just variable-value pairs and hence always positive.

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

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

Action Costs in Relaxed States

Example

Assume $s^+$ is the relaxed state with

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

What should action $a$ with $\text{cost}_a = xy^2 + z + 2$ cost in $s^+$?

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

For simplicity, we identify relaxed states as just variable-value pairs and hence always positive. A relaxed state $s^+$ is applicable in a relaxed state $s^+$ if all precondition facts of $a$ are contained in $s^+$.
Action Costs in Relaxed States

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

**Example**

\[
\begin{align*}
\text{s}^* & \quad a^* [2] \quad \text{t}^* \\
\end{align*}
\]

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 \( \text{cost}_a \).

Cost Computation for Relaxed States

**Example**

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

- Computing \( \text{cost}_a(s^*) = \min_{s \in S} \text{cost}_a(s) \) for all \( s \) consistent with \( s^* \) minimizing over all start-end-paths in EVMDD following only edges consistent with \( s^* \).
- Observation: Minimization over exponentially many paths can be replaced by top-sort traversal of EVMDD, minimizing over incoming arcs consistent with \( s^* \) at all nodes!
Cost Computation for Relaxed States

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

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

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

Proof.
Homework?

Relaxation Heuristics

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

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

$$ h^{add}(s) = h^{add}_g(\text{GoalFacts}) $$

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

$$ h^{add}_s(\text{fact}) = \begin{cases} 0 & \text{if fact} \in s \\ \min \text{achieve} \ a \ \text{of} \ \text{fact} \ h^{add}_s(\text{pre}(a)) + \text{cost}_a & \text{otherwise} \end{cases} $$

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

Relaxations with SDAC

Example

- $a = \langle \top, x = 1 \rangle$ \hspace{1cm} $\text{cost}_a = 2 - 2y$
- $b = \langle \top, y = 1 \rangle$ \hspace{1cm} $\text{cost}_b = 1$

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

- $h^{add}_s(y = 1) = 1$
- $h^{add}_s(x = 1) = ?$

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

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

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

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

Remark: We can use EVMDDs to compute $C^s_\alpha$ 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 $h^{\text{add}}$ values from the previous RPG layer.

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

RPG Compilation

December 18th, 2017 B. Nebel, R. Mattmüller – AI Planning

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 $h^{\text{add}}$ heuristic in $\Pi$ gives the same value for $s \cup \{x \mapsto 0\} \cup \{a \mapsto 0 \mid a \in O\}$ as the generalization of $h^{\text{add}}$ to SDAC tasks defined above gives for $s$ in $\Pi$.

Computing $h^{\text{add}}$ for SDAC:

- Option 1: Compute classical $h^{\text{add}}$ on compiled task.
- Option 2: Compute $C^s_\alpha$ directly. How?

  - Plug EVMDDs as subgraphs into RPG
  - Efficient computation of $h^{\text{add}}$

Option 2: RPG Compilation Option 2: Computing $C^s_\alpha$

Evaluate nodes:

- $\text{cost}_a = xy^2 + z + 2$
- Variable nodes become $\lor$-nodes
- Weights become $\land$-nodes
- Augment with input nodes
- Ensure complete evaluation
- Insert $h^{\text{add}}$ values
- $\land$: $\Sigma(\text{parents}) + \text{weight}$
- $\lor$: $\text{min}(\text{parents})$
- $C^s_\alpha = \text{value}$

December 18th, 2017 B. Nebel, R. Mattmüller – AI Planning
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 a SDAC planning task. Then the classical additive RPG evaluation of the RPG constructed using EVMDDs as above computes the generalized additive heuristic $h_{add}$ defined before.

Abstractions

Question: Why consider abstraction heuristics?

Answer:
- admissibility
- $\Rightarrow$ optimality

Abstraction Heuristics for SDAC

Question: What are the abstract action costs?

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

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

Problem: exponentially many states in minimization

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

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

Definition (Cartesian abstraction)
A set of states $s^{abs}$ is Cartesian if it is of the form
\[ D_1 \times \cdots \times D_n, \]
where $D_i \subseteq D_i$ for all $i = 1, \ldots, n$.
An abstraction is Cartesian if all abstract states are Cartesian sets.

[Intseipp and Helmert, 2013]

Intuition: Variables are abstracted independently.
\[ \Rightarrow \text{exploit independence when computing abstract costs!} \]

Example (Cartesian abstraction)
Cartesian abstraction over $x, y$

\[
\begin{align*}
\text{Cost: } x + y + 1, \\
\text{edges consistent with } s^{abs}
\end{align*}
\]

\[
\begin{array}{ccc}
 x = 0 & y = 0 & y = 2 \\
 00 & 01 & 02 \\
 10 & 11 & 12 \\
 x = 1 & y = 1 \\
 20 & 21 & 22 \\
 x = 2 & \\
 \text{cost} = 4 & \text{cost} = 5 \\
\end{array}
\]

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

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

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

Longer answer:
1. For each Cartesian state $s^{abs}$ and each variable $v$, each value $d \in D_v$ is either consistent with $s^{abs}$ or not.
2. This implies: at all decision nodes associated with variable $v$, some outgoing edges are enabled, others are disabled.
   This is independent from all other decision nodes.
3. This allows local minimizations over linearly many edges instead of global minimization over exponentially many paths in the EVMDD when computing minimum costs.
\[ \Rightarrow \text{polynomial in EVMDD size!} \]

If abstraction not Cartesian: two variables can be
- independent in cost function ($\Rightarrow$ compact EVMDD), but
- dependent in abstraction.
\[ \Rightarrow \text{cannot consider independent parts of EVMDD separately.} \]

Example (Non-Cartesian abstraction)
\[ \text{cost: } x + y + 1, \text{cost}(s^{abs}) = 2, \text{local minim.: 1 } \Rightarrow \text{underestimate!} \]
CEGAR and Cartesian Abstractions

Assume the following:

- **Initial abstraction** is one-state abstraction with single abstract state $D_1 \times \cdots \times D_n$.
- **Cartesian abstraction** each refinement step takes one abstract state $s^{ab} = D_1 \times \cdots \times D_i \times D_{i+1} \times \cdots \times D_n$, one variable $v_i$, and splits $s^{ab}$ into $D_1 \times \cdots \times D_{i-1} \times D'_i \times D_{i+1} \times \cdots \times D_n$ 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.

Some questions:

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

Some questions:

- **Q**: How long to stay in refinement loop?
  - **A**: Until one of the following termination criteria is met:
    - No abstract plan exists.
      - **Terminate** with result "unsolvable".
    - Abstract plan $\pi$ is concretizable ($\Rightarrow$ has no flaw).
      - **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, ...).
Consider the following FDR planning task \( \langle V, I, O, \gamma \rangle \):

- \( V = \{ t, p \} \) with
  - \( \mathcal{H} = \{ L, R \} \)
  - \( \mathcal{P} = \{ L, T, R \} \)
- \( I = \{ t \mapsto L, p \mapsto L \} \)
- \( O = \{ \text{pick-in}_i \mid i \in \{ L, R \} \} \)
  \cup \{ \text{drop-in}_i \mid i \in \{ L, R \} \}
  \cup \{ \text{move}_{ij} \mid i, j \in \{ L, R \}, i \neq j \} \), where
  - \( \text{pick-in}_i = (t = i \land p = i, p := T) \)
  - \( \text{drop-in}_i = (t = i \land p = T, p := \bot) \)
  - \( \text{move}_{ij} = (t = i, t := j) \)
- \( \gamma = (p = R). \)

**CEGAR: Flaws**

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

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

Flaw 1: Abstract plan terminates in concrete non-goal state.
Resolution: Split abstraction of last state $s_n$ of concrete trace into (a) part containing $s_n$, but containing no concrete goal state, and (b) rest.

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.

Flaw 3: Concrete and abstract paths diverge.
Resolution: Split abstraction of state $s_{i-1}$ of concrete trace, after which paths diverge when applying operator $o$, into (a) part containing $s_{i-1}$ where applying $o$ always leads to the “wrong” abstract successor state, and (b) rest.

CEGAR: Cost-Mismatch Flaws

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

Flaw 4: Action is more costly in concrete state than in abstract state.
Resolution: Split abstraction of violating concrete state into two parts that differ on the value of a variable that is relevant to the cost function of the operator in question, such that we have different cost values in the two parts.
CEGAR: Cost-Mismatch Flaws

Example (Cost-mismatch flaw)

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

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

SDAC Planning and EVMDDs

Conclusion

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

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

