AI Planning

20. Symmetry Reduction

How to Not Try “the Same Thing” Over and Over Again

Álvaro Torralba, Cosmina Croitoru

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources
Agenda

1. Introduction
2. Symmetry Basics
3. Finding Symmetries
4. Exploiting Symmetries
5. Conclusion
**Optimal Planning**: Admissible heuristics estimate distance to goal \( h(s) \) and avoid the expansion of nodes whose \( g(s) + h(s) > f^*(I) \)

They are a very effective technique but not the only one!

Sometimes, even almost perfect heuristics are not good enough![Helmert and Röger (2008)]

**Definition (Almost Perfect Heuristic)**. A heuristic is almost perfect if it differs from the perfect heuristic \( h^* \) only by an additive constant:

\[
(h^* - c)(s) = \max(h^*(s) - c, 0).
\]

**Definition (Search effort \( N^c(\Pi) \))**. Let \( \Pi \) be a planning task. We denote \( N^c(\Pi) \) to the number of states \( s \) with \( g(s) + (h^* - c)(s) < h^*(\Pi) \).

\( A^* \) with \( h^* - c \) will expand at least \( N^c(\Pi) \) nodes.

**Theorem**. There exist families of planning tasks \( \{\mathcal{T}_1, \ldots, \mathcal{T}_n\} \) where \( \mathcal{T}_i \) is a planning task of size \( i \) and \( N^c(\mathcal{T}_i) \) grows exponentially in \( i \) even for small \( c \).

**Proof**. Gripper (see next slide), Miconic, Blocksworld, \ldots
Motivation: Example Gripper

Gripper:
\[ T_n \text{ carry } n \text{ balls from } L \text{ to } R. \]

Empirical results:

<table>
<thead>
<tr>
<th>Balls</th>
<th>( h^*(I) )</th>
<th>( N^1(\Pi) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>11</td>
<td>125</td>
</tr>
<tr>
<td>06</td>
<td>17</td>
<td>925</td>
</tr>
<tr>
<td>08</td>
<td>23</td>
<td>5885</td>
</tr>
<tr>
<td>10</td>
<td>29</td>
<td>34301</td>
</tr>
<tr>
<td>12</td>
<td>35</td>
<td>188413</td>
</tr>
<tr>
<td>14</td>
<td>41</td>
<td>991229</td>
</tr>
<tr>
<td>16</td>
<td>47</td>
<td>5046269</td>
</tr>
</tbody>
</table>

Reachable states: \[ S_n = 2(2^n + 2n2^{n-1} + n(n - 1)2^{n-2}) \]

Theorem. Let \( n \in \mathbb{N}_0 \) with \( n \geq 3 \). If \( n \) is even, then \( N^1(T_n) = \frac{S_n}{2} - 3 \). If \( n \) is odd then \( N^1(T_n) = S_n - 3 \).

Proof sketch. If \( n \) is even: basically all states with an even number of balls in each room are part of an optimal plan. If \( n \) is odd, all states are part of an optimal plan.
To the rescue: pruning methods

1. **State Pruning**: Reduces the search effort by not checking parts of the search space. Particular nodes are pruned.

2. **Action Pruning**: Reduces the search effort by considering only some of the applicable actions. Particular edges are pruned.

**We cover 3 different methods for pruning:**

- (State) **Symmetry reduction**: → This Chapter
- (State) **Dominance pruning**: → Chapter 19
- (Action) **Partial-order reduction**: → Chapter 18
Symmetries in a Nutshell: Example
Symmetries in a Nutshell: Wrap-Up

### Basic Algorithm

1. **Pre-search:** Find Symmetries
   - Find (some) generators of the automorphism group that fixes goal
   - Plug them into an *effective* symmetry detection black box

2. **Search:** Use Symmetries
   - Run $A^*$. When a successor node $s'$ is generated, check if a symmetrical node $s$ was already found. If yes, then
     - (a) Update $g(s) := \min\{g(s), g(s')\}$
     - (b) Prune $s'$

3. **Post-search:**
   - Extract plan going backwards from the goal using symmetries
Our Agenda for This Chapter

1. **Symmetry Basics**: Formal definition of symmetries and their associated structures; proving their basic properties.

2. **Finding Symmetries**: We take a look at algorithms that detect symmetric states from the definition of the planning task.

3. **Exploiting Symmetries**: We discuss how symmetries can be used during the search and how the solution plan can be extracted afterwards.
What Is A Symmetry?

**What is a symmetry?** In abstract terms, an object is symmetric with respect to an operation if the operation preserves some property of the object. Here, we will consider automorphisms of the state space.

**Definition (Automorphism).** An automorphism of a graph \((S, E)\) is a permutation \(\sigma : S \rightarrow S\) of the vertices of the graph that preserves the structure of the graph, i.e., for every two vertices \(s_1, s_2 \in S\), we have that \((s_1, s_2) \in E\) iff \((\sigma(s_1), \sigma(s_2)) \in E\).

→ Since transitions in our state space have a cost, we require the action costs to be the same.

**Definition (Automorphism in \(\Theta\)).** An automorphism of a state space \(\Theta\) is a permutation \(\sigma : S \rightarrow S\) of the states that preserves the structure of \(\Theta\), i.e., for every two states \(s_1, s_2 \in S\), we have that \(s_1 \xrightarrow{a} s_2 \in T\) iff \(\sigma(s_1) \xrightarrow{a'} \sigma(s_2) \in T\) with \(c(a) = c(a')\).
Example of Automorphisms

Question!

Which of the following permutations are automorphisms?

(A): ARR-BRR  (B): A-B
(C): XYZ-XZY  (D): L-R

→ (A): No, because then $\sigma(\text{ALR}) \rightarrow \sigma(\text{ARR}) = \text{ALR} \rightarrow \text{BRR} \notin T$
→ (B): No, because $\text{ARL} \rightarrow \text{RRL}$ but $\text{BRL} \nrightarrow \text{RRL}$
→ (C): No, because $\text{ARL} \rightarrow \text{RRL}$ but $\text{ALR} \nrightarrow \text{RLR}$
→ (D): Yes! It corresponds to a vertical symmetry.
→ Horizontal Symmetry: Corresponds to (B) and (C) together.
Definitions: Symmetry Group and Orbit

**Definition (Symmetry Group).** A set of automorphisms \( \Sigma = \{\sigma_1, \ldots, \sigma_n\} \) generates a group \( G \), where \( G \) is the set of all permutations that can be obtained by composing elements of \( \Sigma \).

**Definition (Orbit).** The orbit of a vertex \( s \) with respect to a group \( G \), denoted by \( G(s) \), is the set of vertices to which elements in \( G \) map \( s \).

\[
G(s) = \{\sigma(s) \mid \sigma \in G\}
\]

**Definition (Symmetric states).** Let \( G \) be a symmetry group of the state space of a planning task. Then, we say that \( s, t \) are symmetric, \( s \sim_G t \) iff they belong to the same orbit \( G(s) = G(t) \).
Example: Orbits

Question!

How many orbits are in the group generated by \( \{\sigma_1, \sigma_2\} \)?

(A): 4  
(B): 5  
(C): 8  
(D): 10

\( \rightarrow (5): \) LRR-LLL, LLL-RRR, BLL-ALL-ARR-BRR, LRL-LLR-RRL-RLR,
Automorphisms characterize the transitions of our state space. However, they do not take into account the initial state and goals of the problem. What we want is to consider symmetric states equivalent so that if we have considered a state in the search, we can skip all its symmetric states. In forward search we need to preserve goal distance: \[ s \sim_G t \implies h^*(s) = h^*(t) \]

→ Symmetry methods guarantee that for every plan valid for \( s, \pi \), there exists an equivalent plan for \( t, \sigma(\pi) \) and vice versa.

\[ ^1 \text{In domains with 0-cost actions we also need to take into account the optimal number of 0-cost actions.} \]
Definition (Stabilizer).

Let $G$ be a symmetry group. The **point-stabilizer** of a vertex $s$ with respect to $G$, denoted $G_s$, is a subgroup of $G$ that contains all the permutations that fix $s$. $G_s = \{ \sigma \mid \sigma \in G, \sigma(s) = s \}$.

The **set-stabilizer** of a set of vertices $S$ with respect to $G$, denoted $G_S$, is a subgroup of $G$ that contains all the permutations that fix $S$. $G_S = \{ \sigma \mid \sigma \in G \text{ s.t. } \sigma(s) = t \text{ for all } s, t \in S \}.$

→ We are interested in stabilizing the goal! We call $G_G$ a **goal-stabilizer group** if is the subgroup that stabilizes the set of goal states.

→ Goal-stabilizer groups preserve $h^*$. 
Proposition (**Goal Stabilizer preserves $h^*$**). Let $\Pi$ be an FDR planning task and $G$ a *goal-stabilizer group* of $\Theta_\Pi$. Then, 
$s \sim_G t \implies h^*(s) = h^*(t)$.

**Proof.** Let $s, t$ be symmetric states, i.e., $s \sim_G t$. We show that for every plan for $s (t)$, $\pi$, exists a symmetric plan for $t (s)$ of the same cost.

By induction on the length of the plan, $|\pi|$. **Base case**, $|\pi| = 0$. Then $s$ is a goal state and since $G$ is goal-stabilizer, $t$ is a goal state.

**Inductive case.** $|\pi| = n > 0$. Let $a$ be the first action in $\pi$. Then, $s' = a(s)$ is the next state in the plan for $s$. Since $s$ and $t$ are symmetric, there exists $\sigma(a)$ such that $t \xrightarrow{\sigma(a)} t'$ where $t' \sim s'$. By induction, $h^*(s') = h^*(t')$. Since $a$ and $\sigma(a)$ are symmetric, $c(a) = c(\sigma(a))$. 
And Now for Something Completely Different

- $V$: $D_1, D_2, D_3 : \{Hand, Thrown\}; T_1, T_2 : \{0, 1, 2, 3\}$.
- Initial state $I$: $D_1 = D_2 = D_3 = Hand$, $T_1 = T_2 = 0$
- Goal $G$: $T_1 = 3$ or $T_2 = 3$.
- Actions $A$:
  - $\text{throwFirst}(x, y)$: 
    - pre $D_x = Hand$, $T_1 = 0$, $T_2 = 0$;
    - eff: $T_y = 1$, $D_x = Thrown$
  - $\text{throwNext}(x, y)$: 
    - pre $D_x = Hand$, $T_y > 0$;
    - eff $T_y = T_y + 1$, $D_x = Thrown$
Question!

Find a symmetry goal-stabilizer group: $G = \{\sigma_1, \ldots, \sigma_k\}$.
How many orbits are in the group?

$G = \{\sigma_1, \sigma_2, \sigma_3\}$
$
\sigma_1 = T_1 \leftrightarrow T_2, \\
\sigma_2 = D_1 \leftrightarrow D_2 \text{ and} \\
\sigma_3 = D_2 \leftrightarrow D_3.
$

This induces 4 orbits, one for each goal distance. For example,

$THH10 = \sigma_1(\sigma_2(\sigma_3(HHT01)))$. 
How to Find Symmetries?

Symmetry groups are characterized as automorphisms (i.e., permutations of states in our state space). However:

1. How to find a set of automorphisms?
   → look for **structural symmetries** in the problem description.

2. How to succinctly represent them?
   → permutations of facts (like in the example of slide 19).

Procedure:

1. Define Problem Description Graph, a graph that represents the problem.
2. Use state-of-the-art algorithms to find automorphisms in that graph.
**Structural Symmetry**

**Definition (Structural Symmetry).** Let $\Pi = (V, A, I, G)$ be an FDR planning task. A permutation $\sigma$ on $V \cup A \cup (\bigcup_{v \in V} D_v)$ is a structural symmetry if

- $\sigma(V) = V$
- $\sigma(A) = A$, and for all $a \in A$:
  - $\text{pre}(\sigma(a)) = \sigma(\text{pre}(a))$
  - $\text{eff}(\sigma(a)) = \sigma(\text{eff}(a))$
  - $c(\sigma(a)) = c(a)$
- $\sigma(G) = G$

**Theorem (Structural symmetries are goal-stabilizer).** Let $\Pi$ be an FDR planning task and $\sigma_1, \ldots, \sigma_k$ structural symmetries of $\Pi$. Then $\{\sigma_1, \ldots, \sigma_k\}$ is a goal-stabilizer group of $\Theta_\Pi$.

**Proof Intuition** Since $\sigma(V) = V$; $\sigma$ is a permutation ($\sigma(s)$ is a state). Since $\sigma(A) = A$ (and pre, eff, c); $\sigma$ is an automorphism of $\Pi$ Since $\sigma(G) = G$; $\sigma$ is a goal-stabilizer subgroup.
Definition (Problem Description Graph). Let $\Pi = (V, A, I, G)$ be an FDR planning task. The problem description graph of $\Pi$ is the undirected colored graph $PDG(\Pi)$ with four types of vertices colored of different colors:

1. one for each variable $N_v$, $v \in V$,
2. one for each value of a variable $N_d$, $d \in D_v$, and
3. two for each action corresponding to its preconditions and effects: $N_{pre_a}$ and $N_{eff_a}$, $a \in A$.

There is an arc between:

1. $N_v$ and $N_d$ iff $d \in D_v$,
2. $N_{pre_a}$ and $N_{eff_a}$ for every $a \in A$, and
3. $d$ and $pre_a$ or $eff_a$ iff $d \in pre_a$ or $d \in eff_a$.  

Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 20: Symmetry Reduction 23/43
Problem Description Graph: Example

\[
a_1 = \text{load}(T_1, L) \quad a_2 = \text{move}(T_1, L, R) \quad a_3 = \text{move}(T_2, L, R)
\]
Using the PDG

**Definition (State-permutation induced by PDG-permutation).** Let \( \Pi = (V, A, I, G) \) be an FDR planning task. Let \( \pi \) be a permutation of \( \text{PDG}(\Pi) \). The *induced permutation* on the state space of the planning task, \( \pi' : S \rightarrow S \) is defined as \( \pi'(s) = \{ \pi(v), \pi(d) \mid \langle v, d \rangle \in s \} \).

**Theorem (PDG automorphisms to state space automorphisms).** Let \( \Pi \) be an FDR planning task. Let \( \sigma \) be an automorphism of \( \text{PDG}(\Pi) \) and let \( \sigma' \) be the state-permutation induced by \( \sigma \). Then \( \sigma' \) is a goal-stabilizer automorphism of the state space of \( \Theta_\Pi \).

**Proof Intuition** The PDG-permutation renames operators, variables and values in a way that preserves the semantics.
Finding automorphisms in the PDG

- **On the bad side** – “Is the graph automorphism problem in P, or NP, or neither?” Is an open problem in CS. It is harder than graph isomorphism which is also not known to be in P.

- **On the good side** We compute the automorphisms **only once before starting the search** – We already in PSPACE, so how bad can it be?

- Open-source software tools that are available for this task:

\(^2\)Also produce Canonical Labeling
Exploiting Symmetries: Basic Idea

Basic Idea

1. **Pre-search:** Find Symmetries
   Find (some) generators of the automorphism group that fixes goal

2. **Search:** Use Symmetries
   Run heuristic search. When a successor node $s'$ is generated, check if a symmetrical node $s$ was already found. If yes, then
   (a) If $g(s) > g(s')$, then update $g$, parent, and achieving action of $s$ to those of $s'$ and reopen $(s)$
   (b) Prune $s'$

3. **Post-search:**
   Non-standard plan extraction
Finding Symmetric States

Given state $s$, is $s$ symmetric to another previously known state?

**Problem 1.** Finding symmetric states is intractable. Given a pair of states $s, t$ and a symmetry group $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ decide whether $G(s) = G(t)$ is NP-hard.

**Problem 2.** Comparing $s$ against all previously known states may be very expensive.

→ Each orbit is implicitly represented by one of its states, called the **canonical state**, e.g., the lexicographically **smallest** state in the orbit. Whenever we generate $s$, replace it for the representative of its orbit. Duplicate elimination takes care of the rest.

→ Given $s$, finding the canonical state of $G(s)$ is still NP-hard (see Problem 1)!

→ **We approximate**! Use a greedy algorithm to find a lexicographically smaller state, but not necessarily the smallest.
Representative states

\[
\text{function } \text{FindRepresentativeGreedy} \ (G = \sigma_1, \ldots, \sigma_k, \text{ state } s) \\
\text{while } \exists \sigma \in G, \text{ s.t., } \sigma(s) < s \ \text{ do} \\
\quad s \leftarrow \sigma(s) \\
\text{return } s
\]

\text{FindRepresentativeGreedy} \ induced \ G_C, \ a \ new \ subgroup \ of \ G.

The orbit \( G_C(s) \) is formed by all states \( s' \) s.t.
\text{FindRepresentativeGreedy}(G, s') = s.

The equivalence relation is defined as \( s \sim_{G_C} t \) iff \( G_C(s) = G_C(t) \).

\[
\rightarrow \ s \sim_{G_C} t \implies s \sim_G t \ \text{so the approximation is still safe.}
\]
\[
\rightarrow \ s \sim_{G_C} t \iff s \sim_C t \ \text{so the approximation is less powerful.}
\]
A* with Symmetries

1. **Search:** Run A*. When a successor node $s'$ is generated, check if a symmetrical node $s$ was already found (by checking if FindRepresentativeGreedy($s'$) is a duplicate). If yes, then
   (a) If $g(s) > g(s')$, then update $g$, parent, and achieving action of $s$ to those of $s'$ and reopen ($s$)
   (b) Prune $s'$

2. **Post-search:** Non-standard plan extraction

- $s$ and $s'$ are symmetric. Should we prune $s$ or $s'$?
  → We prune $s'$ because $s$ might have already been expanded so we cannot easily prune it anymore.
- Is it possible that $g(s) > g(s')$? → Yes, $h^*(s) = h^*(s')$ but $g^*(s)$ and $g^*(s')$ may differ as well as $h(s)$ and $h(s')$.
- The new values of $g$, parent and achieving action of $s$ are not “true”, but we know that they are for some symmetric state $s'$ and that suffices for plan extraction (see next slides)
A* with Symmetries: Example
Solution Reconstruction with Symmetries

The plan returned by $A^*$ with symmetries is not valid, but a valid plan can be reconstructed by "undoing" the symmetries.

```plaintext
function Trace-forward ($\pi = \langle (\epsilon, s_0), (a_1, s_1), \ldots, (a_m, a_m) \rangle$)
let $\sigma_i \in \Gamma_{S_*}$ be such that $\sigma_i(s_i) = s_{i-1}[a_i]$ for $0 < i \leq m$
$\sigma := \sigma_{id}$
$\rho := \langle \epsilon \rangle$
for $i := 1$ to $m$ do:
    $s := \sigma(s_{i-1})$, $\sigma := \sigma \circ \sigma_i$, $s' := \sigma(s_i)$
    append to $\rho$ a cheapest action $a$ such that $s[a] = s'$
endfor
return $\rho$
```
Solution Reconstruction with Symmetries: Example

\[ \pi = (\epsilon, HHH00), (\text{throw}(D_1, T_1), HTH10), (\text{throw}(D_1, T_1), THT02), (\text{throw}(D_2, T_2), TTT30) \]

\[ \sigma_1 = \{D_1 \leftrightarrow D_2\} \]

\[ \sigma_2 = \{D_2 \leftrightarrow D_3, T_1 \leftrightarrow T_2\} \]

\[ \sigma_3 = \{T_1 \leftrightarrow T_2\} \]

→ The figure shows a search tree after the goal has been found. All nodes that were pruned due to symmetries are omitted. Dashed edges represent the transitions that were substituted because a better path was found to a symmetric state. For example, \(HHH00 \xrightarrow{t_{2,1}} HTH10\) is substituted by \(HHH00 \xrightarrow{t_{1,1}} TTH10\) because \(HTH10\) and \(TTH10\) are detected as symmetric and the second has lower cost.³

³ In the example, \(g(HTH10) = g(TTH10)\) but we assume that the substitution has occurred anyway in order to show how the solution reconstruction works.
A symmetry $\sigma$ is an automorphism, i.e., a permutation of the states that preserves the structure (goal-distance) of the state space.

A set of permutations $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ defines a group $G$, as is the set of all permutations that can be obtained by composing elements in $\Sigma$. The orbit of an state $s$, $G(s)$ is the set of states $s'$ such that exists $\sigma \in G$, $\sigma(s) = s'$.

A group $G$ defines an equivalence class. $s$ and $t$ are symmetric according to $G$, $s \sim_G t$, iff $G(s) = G(t)$.

In forward search we are interested in groups that stabilize the goal, i.e., goal states are only symmetric to other goal states.

Symmetries can be obtained by computing the automorphisms of the Problem Description Graph (PDG). Those are structural symmetries of the domain and can be succinctly represented.
Remarks

Symmetries are also useful for satisficing planning. [Domshlak et al. (2013)]

Relation between symmetries and heuristics. [Shleyfman et al. (2015)]

**Definition (Invariant Under Structural Symmetries).** A heuristic $h$ is invariant under structural symmetries if $h(s) = h(t)$ for every $s \sim t$.

- If a heuristic is invariant under symmetries we do not need to evaluate more than one state in the same orbit.
- If a heuristic is not invariant under symmetries, it is admissible to evaluate several states from the same orbit and take the maximum.

The coarsest bisimulation captures all symmetries. [Sievers et al. (2015)]

Therefore, we can use bisimulation in M&S to capture all local symmetries to the variables that have been already merged.
Heuristics Invariant Under Structural Symmetries

Classification of heuristics (\(\mathcal{E}\) stands for “with a tie-breaking invariant under symmetries”):

**Non-symmetric**

- \(h^{\text{FF}}\) Hoffmann & Nebel
- \(h^{\text{FF}} / h^{\text{add}}\), \(h^{\text{FF}} / h^{\max}\) Keyder & Geffner
- \(h^{\text{PDB}}\) Culberson & Schaeffer, Edelkamp
- \(h^{M&S}\) Helmert, Haslum & Hoffmann
- \(h^{\text{LM-cut}}\) Helmert & Domshlak

**Symmetric**

- \(h^+\) Hoffmann & Nebel
- \(h^{\max}\) Bonet & Geffner
- \(h^{\text{add}}\) Bonet & Geffner
- \(\mathcal{E} h^{\text{FF}}\) Hoffmann & Nebel

- \(h^m\) Haslum & Geffner
- \(\mathcal{E} h^{\text{LM-cut}}\) Helmert & Domshlak
Remarks: Relation to Dominance

Dominance pruning is strictly more general than symmetry pruning.

Define ≤ such that s ∼ t iff s ≤ t and t ≤ s.

The coarsest bisimulation on Θ finds all the symmetries. The coarsest simulation on Θ is coarser than bisimulation.

So, does symmetry pruning make sense?
Yes! it allows specific ideas that do not work in the general case:

1. Unlike the dominance compositional approach in Chapter 19, the method to compute symmetries finds symmetries over all variables.
2. Orbit search efficiently performs an approximate check. that is not possible in the case of dominance.
Remarks: Orbit Search

In practice, one can just substitute each state by its canonical representative. Then, the pruning is just performed by standard duplicate elimination. We call this orbit search because each search node corresponds to a symmetry orbit.

Orbit Search Algorithm

1. **Pre-search:** Find Symmetries
   
   Find (some) generators of the automorphism group that fixes goal

2. **Search:** Use Symmetries
   
   Run heuristic search. When a successor node $s'$ is generated replace $s'$ by $\text{FindRepresentativeGreedy}(G, s')$. If a duplicate its found update $g$, parent, and achieving action as usual.

3. **Post-search:**
   
   Non-standard plan extraction
Reading

- **Exploiting Problem Symmetries in State-Based Planners** [Pochter et al. (2011)].
  
  Available at: http://www.cs.hujcdi.ac.il/~avivz/pubs/12/PlanningSymmetries_AAAI11.pdf
  
  **Content:** Introduces of symmetries as automorphisms of the state space in planning and the method based on the problem description graph.

- **Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search** [Domshlak et al. (2012)].
  
  Available at: http://iew3.technion.ac.il/~dcarmel/Papers/Sources/icaps12b.pdf
  
  **Content:** Improvement over the work by Pochter et al.. Prove that it is not necessary to stabilize the symmetries with respect to the initial state if the Trace-Forward algorithm is used for solution reconstruction.


