# Planning and Optimization

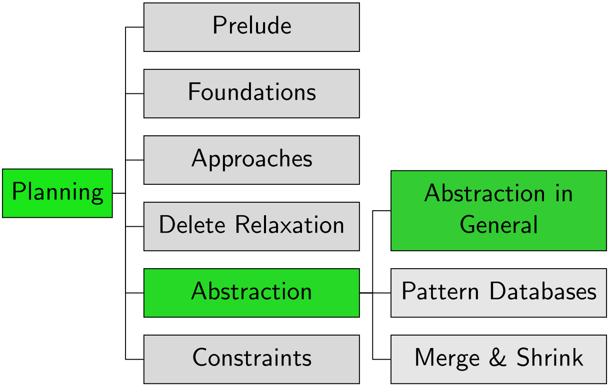
## E3. Abstractions: Introduction

Malte Helmert and Gabriele Röger

Universität Basel

November 6, 2024

# Content of the Course



# Introduction

# Coming Up with Heuristics in a Principled Way

## General Procedure for Obtaining a Heuristic

Solve a simplified version of the problem.

Major ideas for heuristics in the planning literature:

- delete relaxation
- abstraction
- critical paths
- landmarks
- network flows
- potential heuristics

Heuristics based on abstraction are among the most prominent techniques for optimal planning.

# Abstracting a Transition System

Abstracting a transition system means dropping some distinctions between states, while preserving the transition behaviour as much as possible.

- An abstraction of a transition system $\mathcal{T}$ is defined by an abstraction mapping $\alpha$ that defines which states of $\mathcal{T}$ should be distinguished and which ones should not.
- From $\mathcal{T}$ and $\alpha$, we compute an abstract transition system $\mathcal{T}^\alpha$ which is similar to $\mathcal{T}$, but smaller.
- The abstract goal distances (goal distances in $\mathcal{T}^\alpha$) are used as heuristic estimates for goal distances in $\mathcal{T}$.

# Abstracting a Transition System: Example

example from domain-specific heuristic search:

## Example (15-Puzzle)

A 15-puzzle state is given by a permutation $\langle b, t_1, \dots, t_{15} \rangle$ of $\{1, \dots, 16\}$, where $b$ denotes the blank position and the other components denote the positions of the 15 tiles.

One possible abstraction mapping ignores the precise location of tiles 8–15, i.e., two states are distinguished iff they differ in the position of the blank or one of the tiles 1–7:

$$\alpha(\langle b, t_1, \dots, t_{15} \rangle) = \langle b, t_1, \dots, t_7 \rangle$$

The heuristic values for this abstraction correspond to the cost of moving tiles 1–7 to their goal positions.

# Abstraction Example: 15-Puzzle

| | | | |
|---|---|---|---|
| 9 | 2 | 12 | 6 |
| 5 | 7 | 14 | 13 |
| 3 | 4 | 1 | 11 |
| 15 | 10 | 8 | |

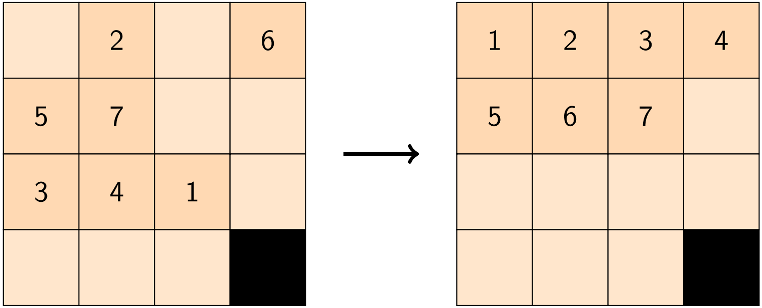
$\longrightarrow$

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | |

real state space:

- $16! = 20922789888000 \approx 2 \cdot 10^{13}$ states
- $\frac{16!}{2} = 10461394944000 \approx 10^{13}$ reachable states

# Abstraction Example: 15-Puzzle




abstract state space:

- $16 \cdot 15 \cdot \ldots \cdot 9 = 518918400 \approx 5 \cdot 10^8$ states
- $16 \cdot 15 \cdot \ldots \cdot 9 = 518918400 \approx 5 \cdot 10^8$ reachable states

# Computing the Abstract Transition System

Given $\mathcal{T}$ and $\alpha$, how do we compute $\mathcal{T}^\alpha$?

**Requirement**

We want to obtain an admissible heuristic.
Hence, $h^*(\alpha(s))$ (in the abstract state space $\mathcal{T}^\alpha$) should never overestimate $h^*(s)$ (in the concrete state space $\mathcal{T}$).

An easy way to achieve this is to ensure that all solutions in $\mathcal{T}$ are also present in $\mathcal{T}^\alpha$:

- If $s$ is a goal state in $\mathcal{T}$, then $\alpha(s)$ is a goal state in $\mathcal{T}^\alpha$.
- If $\mathcal{T}$ has a transition from $s$ to $t$, then $\mathcal{T}^\alpha$ has a transition from $\alpha(s)$ to $\alpha(t)$ with the same cost.

# Computing the Abstract Transition System: Example

## Example (15-Puzzle)

In the running example:

- $\mathcal{T}$ has the unique goal state $\langle 16, 1, 2, \dots, 15 \rangle$.
  - $\rightsquigarrow$ $\mathcal{T}^\alpha$ has the unique goal state $\langle 16, 1, 2, \dots, 7 \rangle$.
- Let $x$ and $y$ be neighbouring positions in the $4 \times 4$ grid. $\mathcal{T}$ has a transition from $\langle x, t_1, \dots, t_{i-1}, y, t_{i+1}, \dots, t_{15} \rangle$ to $\langle y, t_1, \dots, t_{i-1}, x, t_{i+1}, \dots, t_{15} \rangle$ for all $i \in \{1, \dots, 15\}$.
  - $\rightsquigarrow$ $\mathcal{T}^\alpha$ has a transition from $\langle x, t_1, \dots, t_{i-1}, y, t_{i+1}, \dots, t_7 \rangle$ to $\langle y, t_1, \dots, t_{i-1}, x, t_{i+1}, \dots, t_7 \rangle$ for all $i \in \{1, \dots, 7\}$.
  - $\rightsquigarrow$ Moreover, $\mathcal{T}^\alpha$ has a transition from $\langle x, t_1, \dots, t_7 \rangle$ to $\langle y, t_1, \dots, t_7 \rangle$ if $y \notin \{t_1, \dots, t_7\}$.

# Practical Requirements

# Practical Requirements for Abstractions

To be useful in practice, an abstraction heuristic must be efficiently computable. This gives us two requirements for $\alpha$:

- For a given state $s$, the abstract state $\alpha(s)$ must be efficiently computable.
- For a given abstract state $\alpha(s)$, the abstract goal distance $h^*(\alpha(s))$ must be efficiently computable.

There are a number of ways of achieving these requirements:

- pattern database heuristics (Culberson & Schaeffer, 1996)
- domain abstractions (Hernádvölgyi and Holte, 2000)
- merge-and-shrink abstractions (Dräger, Finkbeiner & Podelski, 2006)
- Cartesian abstractions (Ball, Podelski & Rajamani, 2001)
- structural patterns (Katz & Domshlak, 2008)

# Practical Requirements for Abstractions: Example

**Example (15-Puzzle)**

In our running example, $\alpha$ can be very efficiently computed: just project the given 16-tuple to its first 8 components.

To compute abstract goal distances efficiently during search, the most common approach is to precompute all abstract goal distances prior to search by performing a backward uniform-cost search from the abstract goal state(s). These distances are then stored in a table (requires $\approx$ 495 MiB RAM).

During search, computing $h^*(\alpha(s))$ is just a table lookup.

This heuristic is an example of a pattern database heuristic.

# Multiple Abstractions

# Multiple Abstractions

- One important practical question is how to come up with a suitable abstraction mapping $\alpha$.
- Indeed, there is usually a huge number of possibilities, and it is important to pick good abstractions (i.e., ones that lead to informative heuristics).
- However, it is generally not necessary to commit to a single abstraction.

# Combining Multiple Abstractions

Maximizing several abstractions:

- Each abstraction mapping gives rise to an admissible heuristic.
- By computing the maximum of several admissible heuristics, we obtain another admissible heuristic which dominates the component heuristics.
- Thus, we can always compute several abstractions and maximize over the individual abstract goal distances.

Adding several abstractions:

- In some cases, we can even compute the sum of individual estimates and still stay admissible.
- Summation often leads to much higher estimates than maximization, so it is important to understand under which conditions summation of heuristics is admissible.

# Maximizing Several Abstractions: Example

## Example (15-Puzzle)

- mapping to tiles 1–7 was arbitrary
  $\rightsquigarrow$ can use any subset of tiles
- with the same amount of memory required for the tables for the mapping to tiles 1–7, we could store the tables for nine different abstractions to six tiles and the blank
- use maximum of individual estimates

# Adding Several Abstractions: Example

| | | | |
|---|---|---|---|
| 9 | 2 | 12 | 6 |
| 5 | 7 | 14 | 13 |
| 3 | 4 | 1 | 11 |
| 15 | 10 | 8 | |

| | | | |
|---|---|---|---|
| 9 | 2 | 12 | 6 |
| 5 | 7 | 14 | 13 |
| 3 | 4 | 1 | 11 |
| 15 | 10 | 8 | |

- 1st abstraction: ignore precise location of 8–15
- 2nd abstraction: ignore precise location of 1–7

⇝ Is the sum of the abstraction heuristics admissible?

# Adding Several Abstractions: Example

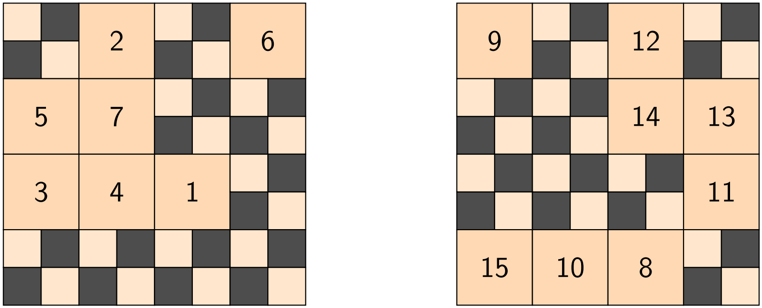
| | | | |
|---|---|---|---|
| | 2 | | 6 |
| 5 | 7 | | |
| 3 | 4 | 1 | |
| | | | |

| | | | |
|---|---|---|---|
| 9 | | 12 | |
| | | 14 | 13 |
| | | | 11 |
| 15 | 10 | 8 | |

- 1st abstraction: ignore precise location of 8–15
- 2nd abstraction: ignore precise location of 1–7

⇝ The sum of the abstraction heuristics is not admissible.

# Adding Several Abstractions: Example




- 1st abstraction: ignore precise location of 8–15 and blank
- 2nd abstraction: ignore precise location of 1–7 and blank

$\leadsto$ The sum of the abstraction heuristics is admissible.

# Outlook

# Our Plan for the Next Lectures

In the following, we take a deeper look at abstractions and their use for admissible heuristics.

In the next two chapters, we formally introduce abstractions and abstraction heuristics and study some of their most important properties.

Afterwards, we discuss some particular classes of abstraction heuristics in detail, namely

- pattern database heuristics and
- merge-and-shrink abstractions.

# Summary

# Summary

- Abstraction is one of the principled ways of deriving heuristics for planning tasks and transition systems in general.
- The key idea is to map states to a smaller abstract transition system $\mathcal{T}^\alpha$ by means of an abstraction function $\alpha$.
- Goal distances in $\mathcal{T}^\alpha$ are then used as admissible estimates for goal distances in the original transition system.
- To be practical, we must be able to compute abstraction functions and determine abstract goal distances efficiently.
- Often, multiple abstractions are used.
  They can always be maximized admissibly.
- Adding abstraction heuristics is not always admissible.
  When it is, it leads to a stronger heuristic than maximizing.