SAARLAND UNIVERSITY ARTIFICIAL INTELLIGENCE
Prof. Jörg Hoffmann and Prof. Wolfgang Wahlster
Dr. Álvaro Torralba, Daniel Gnad, Marcel Steinmetz
Christian Bohnenberger, Cosmina Croitoru, Akram Elkorashy, Sophian Guidara,
Daniel Heller, Björn Mathis, Lukas Schaal, Julia Wichlacz

Exercise Sheet 7.

Solutions due Tuesday, **June 21**, 16:00 – 16:15, in the lecture hall.[1]

Exercise 25. (2.5 Points)

Consider the following problem. A fused bulb hangs out of reach from the ceiling. A robot needs to repair the bulb. The room also contains a box. Pushing that box into the correct position, and climbing onto the box, will bring the bulb into reach for the robot.
The exercise is to model this problem as a STRIPS planning task. In doing so, assume the following framework. The robot is currently at position "A", the fused bulb is at position "B", and the box is at position "C". The robot and box are at the same height "Low", the fused bulb is at height "High". By climbing onto the box, the robot changes from "Low" to "High"; vice versa when climbing off the box. The actions available for the robot are "*Go*" from one place to another (only possible if the robot is at "Low"), "*Push*" an object from one place to another (only possible if the robot and object are at "Low"), "*ClimbUp*" onto or "*ClimbDown*" from an object, and "*Repair*" to fix an object. The robot needs to be at the same place and height as an object in order to repair it.
Note that the robot can only push an object or climb onto an object if both of them are at the same location. Furthermore, in case of pushing an object the robot changes to the destination location as well.

(a) Write a STRIPS formalization of the initial state and goal descriptions.

(b) Write a STRIPS formalization of the five actions. In doing so, please do make use of "object variables", i.e., write the actions up in a parameterized way. State, for each parameter, by which objects it can be instantiated.

In both (a) and (b), make use of the following predicates: (do not use any other predicates)

- $At(x, y)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ is at position $y \in \{A, B, C\}$.

[1]Solutions in paper form only, and solution submission only at the stated time at the stated place. At most 3 authors per solution. All authors must be in the same tutorial group. All sheets of your solution must be stapled together. At the top of the first sheet, you must write the names of the authors and the name of your tutor. Your solution must be placed into the correct box for your tutorial group. If you don't comply with these rules, 3 points will be subtracted from your score for this sheet.

- $Height(x, y)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ is at height $y \in \{Low, High\}$.
- $Pushable(x)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ can be pushed.
- $Climbable(x)$: To indicate that the robot can climb on object $x \in \{Box, Bulb, Robot\}$.
- $Repaired(x)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ is repaired.

(Solution)

(a) Initial state description:

$$\begin{aligned} I &= \{At(Robot, A), At(Bulb, B), At(Box, C), \\ &\quad Height(Robot, Low), Height(Box, Low), Height(Bulb, High), \\ &\quad Pushable(Box), Climbable(Box)\} \end{aligned}$$

Goal description: $G = \{Repaired(Bulb)\}$

(b) Action descriptions:

- $Go(x, y) =$
  $$\begin{aligned} &pre : \{At(Robot, x), Height(Robot, Low)\} \\ &add : \{At(Robot, y)\} \\ &del : \{At(Robot, x)\} \end{aligned}$$
  for all $x, y \in \{A, B, C\}$.
- $Push(x, y, z) =$
  $$\begin{aligned} &pre : \{At(Robot, y), Pushable(x), At(x, y), Height(Robot, Low), Height(x, Low)\} \\ &add : \{At(x, z), At(Robot, z)\} \\ &del : \{At(x, y), At(Robot, y)\} \end{aligned}$$
  for all $x \in \{Box, Bulb, Robot\}$, $y, z \in \{A, B, C\}$.
- $ClimbUp(x, y) =$
  $$\begin{aligned} &pre : \{At(Robot, y), At(x, y), Climbable(x), Height(Robot, Low), Height(x, Low)\} \\ &add : \{Height(Robot, High)\} \\ &del : \{Height(Robot, Low)\} \end{aligned}$$
  for all $x \in \{Box, Bulb, Robot\}$, $y \in \{A, B, C\}$.
- $ClimbDown() =$
  $$\begin{aligned} &pre : \{Height(Robot, High)\} \\ &add : \{Height(Robot, Low)\} \\ &del : \{Height(Robot, High)\} \end{aligned}$$

- $Repair(x, y, z) =$

$$\begin{aligned}&pre: \{At(Robot, y), At(x, y), Height(Robot, z), Height(x, z)\}\\&add: \{Repaired(x)\}\\&del: \{\}\end{aligned}$$

  for all $x \in \{Box, Bulb, Robot\}$, $y \in \{A, B, C\}$, $z \in \{High, Low\}$.

(/Solution)

Exercise 26. (2.5 Points)

Consider the following planning task. A truck (denoted **T**) has to transport packages (denoted P_x) to their goal locations. To do so, it can drive between any two connected locations and (un)load a package at its current position. However, the truck is only able to drive if a package is loaded (empty truck drives are not allowed). Additionally, only a single package can be loaded at the same time. In the initial state, the truck and package P_1 are at position A and the package P_2 is at position B; the goal is to bring P_2 to location C. Figure 1 illustrates the initial state as well as the connections between the locations.

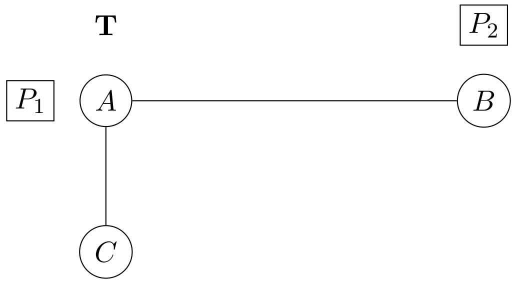


Figure 1: Map showing the initial state of the planning task.

In STRIPS, this task is formalized as follows:

$$\begin{aligned}I &= \{\text{atT}(A), \text{atP}(1, A), \text{atP}(2, B), empty\}\\G &= \{\text{atP}(2, C)\}\\A &= \{drive(x, y, i) \mid \{x, y\} \in \{\{A, B\}, \{A, C\}\}, x \neq y, i \in \{1, 2\}\}\\&\cup\ \{load(i, x) \mid i \in \{1, 2\}, x \in \{A, B, C\}\}\\&\cup\ \{unload(i, x) \mid i \in \{1, 2\}, x \in \{A, B, C\}\}\end{aligned}$$

$$\begin{aligned} \mathit{drive}(x,y,i) &= \mathit{pre}: \{\text{atT}(x), \text{atP}(i,T)\} \\ &\quad \mathit{add}: \{\text{atT}(y)\} \\ &\quad \mathit{del}: \{\text{atT}(x)\} \end{aligned}$$

1. Formalize the preconditions, add list, and delete list of the load and unload actions in STRIPS.
2. How can the task be modified to allow for *up to* two packages being loaded into the truck at the same time? (the truck should be able to load 0, 1, or 2 packages, but still only be able to drive if at least one package is loaded).

If you need to add new facts to P or change existing ones, describe why you do that and what the new facts are used for. Write down the initial state and the new load/unload actions in STRIPS syntax.
(Solution)

1.

$$\begin{aligned} \mathit{load}(i,x) &= \mathit{pre}: \{\text{atP}(i,x), \text{atT}(x), \mathit{empty}\} \\ &\quad \mathit{add}: \{\text{atP}(i,T)\} \\ &\quad \mathit{del}: \{\text{atP}(i,x), \mathit{empty}\} \\ \mathit{unload}(i,x) &= \mathit{pre}: \{\text{atP}(i,T), \text{atT}(x)\} \\ &\quad \mathit{add}: \{\text{atP}(i,x), \mathit{empty}\} \\ &\quad \mathit{del}: \{\text{atP}(i,T)\} \end{aligned}$$

2. Remove the *empty* fact, add 3 facts “capacityX” with $X \in \{0,1,2\}$, so that each of them is true if the truck has exactly X capacity left. In the initial state, *capacity*2 is true.

$$I = \{\text{atT}(A), \text{atP}(1,A), \text{atP}(2,B), \mathit{capacity}2\}$$

$$\begin{aligned} \mathit{load}_1(a,x) &= \mathit{pre}: \{\text{atP}(a,x), \text{atT}(x), \mathit{capacity}1\} \\ &\quad \mathit{add}: \{\text{atP}(a,T), \mathit{capacity}0\} \\ &\quad \mathit{del}: \{\text{atP}(a,x), \mathit{capacity}1\} \end{aligned}$$

$$\begin{aligned} \mathit{load}_2(a,x) &= \mathit{pre}: \{\mathrm{atP}(a,x), \mathrm{atT}(x), \mathit{capacity}2\} \\ &\quad \mathit{add}: \{\mathrm{atP}(a,T), \mathit{capacity}1\} \\ &\quad \mathit{del}: \{\mathrm{atP}(a,x), \mathit{capacity}2\} \end{aligned}$$

$$\begin{aligned} \mathit{unload}_1(a,x) &= \mathit{pre}: \{\mathrm{atP}(a,T), \mathrm{atT}(x), \mathit{capacity}0\} \\ &\quad \mathit{add}: \{\mathrm{atP}(a,x), \mathit{capacity}1\} \\ &\quad \mathit{del}: \{\mathrm{atP}(a,T), \mathit{capacity}0\} \end{aligned}$$

$$\begin{aligned} \mathit{unload}_2(a,x) &= \mathit{pre}: \{\mathrm{atP}(a,T), \mathrm{atT}(x), \mathit{capacity}1\} \\ &\quad \mathit{add}: \{\mathrm{atP}(a,x), \mathit{capacity}2\} \\ &\quad \mathit{del}: \{\mathrm{atP}(a,T), \mathit{capacity}1\} \end{aligned}$$

(/Solution)

Exercise 27. (3 Points)

In this exercise, we consider the problem of collecting objects of different colors, while moving along a road map that takes the form of a directed full binary tree of depth 3 (see pictures below). Each tree node might contain objects having any of three different colors (red, green, blue); moving to that node immediately collects all the objects present. The corresponding STRIPS planning task $\Pi = (P, A, I, G)$ looks as follows:

$$\begin{aligned} P &= \{at(x) \mid x \in \{A, \dots, O\}\} \cup \{got(c) \mid c \in \{red, green, blue\}\} \\ A &= \{move(x,y) \mid (x,y) \text{ is an edge in the tree}\} \\ I &= \{at(A)\} \\ G &= \{got(red), got(green), got(blue)\} \end{aligned}$$

Here, the action $move(x,y)$ is specified as follows:

- In case that the destination y does not contain any objects: $move(x,y) =$

$$\begin{aligned} &\mathit{pre}: \{at(x)\} \\ &\mathit{add}: \{at(y)\} \\ &\mathit{del}: \{at(x)\} \end{aligned}$$

  for all x, y in the tree with an edge from x to y.

- In case that the destination y does contain objects (shown here for red and blue; other cases are defined accordingly): $move(x, y) =$

$$\begin{aligned} &pre : \{at(x)\} \\ &add : \{at(y), got(red), got(blue)\} \\ &del : \{at(x)\} \end{aligned}$$

  for all x, y in the tree with an edge from x to y, where y contains both a red object and a blue object.

Note in the figures below that the edges are directed, i.e., you can pass them only in one direction.
Perform Greedy Best-First Search as specified in Chapter 14 of the lecture. As the heuristic function, use the following goal counting heuristic: For state s, $h(s)$ is the number of goal facts $p \in G$ that are not true in s. (In other words: $h(s) = |G \setminus s|$.) Whenever you need to break ties, i.e., when $h(s) = h(s')$ is minimal for at least two states s and s' in the open list, do so alphabetically. That is, choose the state the name of whose graph node (the node x where the state contains the fact $at(x)$) comes first in the alphabet.
Draw the search space, showing all states generated during the search, including an edge from each expanded state to the child nodes generated when expanding it. Identify each state s by the name of the respective graph node (the node x for which "$at(x)$" is true in s), as well as the subset of colors already collected in the state (those colors x for which "got(x)" is true in s). Annotate each state with its heuristic value, as well as a number indicating the expansion order. (For generated states that are not expanded, use "–" as the expansion order number.)

(a) Consider the road map in Figure 2.

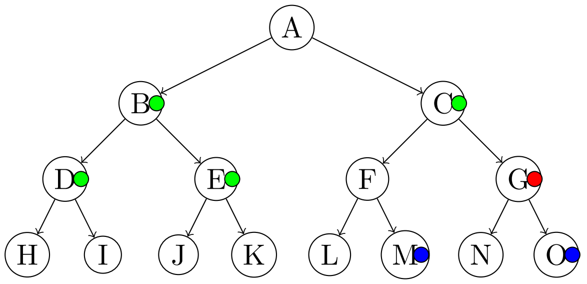


Figure 2: Tree for Exercise 27(a).

(b) Consider the road map in Figure 3.

(c) Consider the road map in Figure 4.

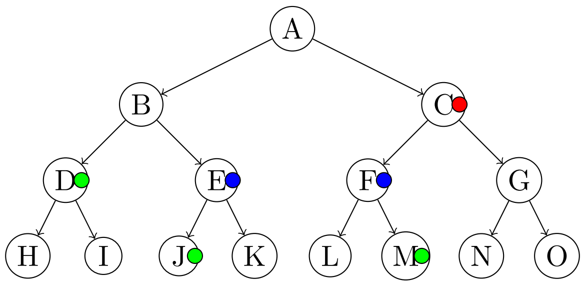


Figure 3: Tree for Exercise 27(b).

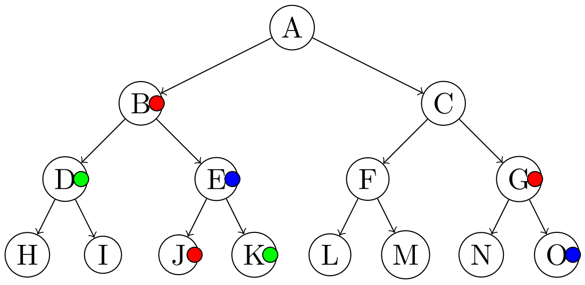
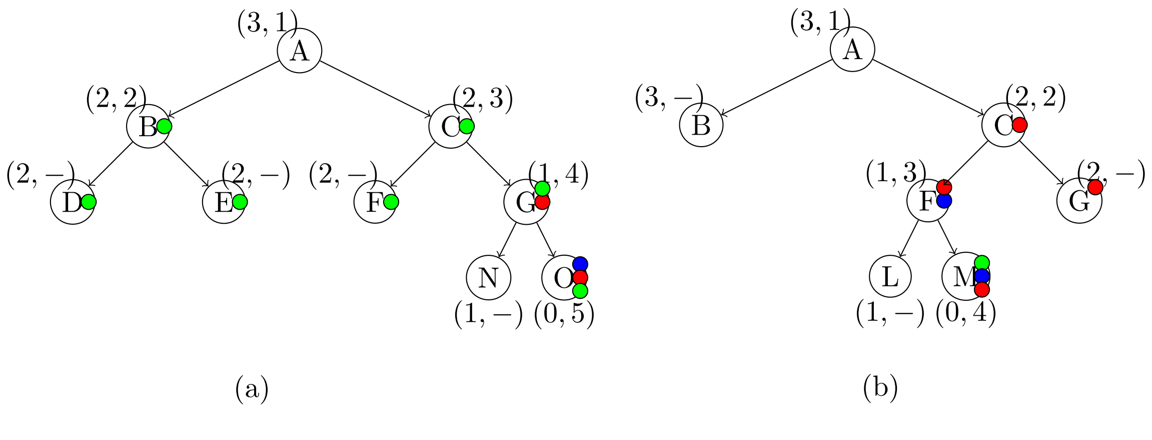
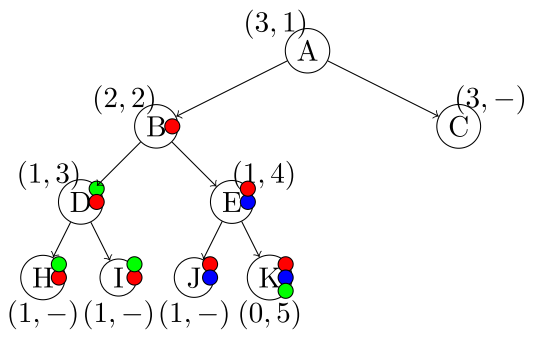


Figure 4: Tree for Exercise 27(c).

(d) What can you say about the behavior of search on the three different road maps of (a), (b), (c)? What properties of the road maps and of the heuristic function h used are responsible for that behavior?

(Solution)
All nodes in the following search space graphs are annotated with a pair of numbers (a, b), where a is the heuristic value and b the number of that node in the expansion order.





(c)

(d) Search in (a) is similar to breadth-first search, until it reaches G, which has a red object. In (b), it leads more or less directly to the goal, except for generating B, G, and L. In (c), search completely discards the right branch, because there is only an object in B in the first layer. In this branch, it performs search similar to breadth-first search, not expanding all states in the last layer. Responsible for this is the distribution of colored objects, in combination with the heuristic which greedily draws the search down tree branches where colors can be collected immediately.

(/Solution)

Exercise 28. (2 Points)

As specified in the lecture, PolyPlanLen is the problem of deciding, given a STRIPS planning task Π and an integer B bounded by a polynomial in the size of Π, whether or not there exists a plan for Π of length at most B.
Prove that PolyPlanLen is **NP**-hard. Tip: Use a polynomial reduction from SAT.