Planning and Optimization

M. Helmert, G. Röger
P. Ferber, T. Keller, S. Sievers

University of Basel
Fall Semester 2020

Exercise Sheet D

Due: November 15, 2020

Important: for submission, consult the rules at the end of the exercise. Non-adherence to the rules will lead to your submission not being corrected.

Exercise D.1 (4 marks)(Lecture D2)

Consider the following graph G depicting a simple transition system. Assume that operators o_i have cost 1, while operators o'_i have cost 5. As usual, an incoming arrow indicates the initial state, and goal states are marked by a double rectangle.

$$\begin{tikzcd}
G: & {} \arrow[r] & s_1 \arrow[r, "o'_1"] \arrow[d, "o_2"'] & s_2 \arrow[r, "o'_5"] & s_3 \arrow[r, "o'_8"] \arrow[d, "o_9"] & \boxed{s_4} & \\
 & & s_5 \arrow[r, "o'_4"] \arrow[ur, "o'_3"] & s_6 \arrow[r, leftrightarrow, "o'_7"] \arrow[ur, "o'_6"] & s_7 \arrow[r, "o'_{10}"] \arrow[ur, "o_{11}"] & s_8 \arrow[u, "o_{12}"'] \arrow[r, leftrightarrow, "o_{13}"] & s_9
\end{tikzcd}$$

Provide the following graphs:

- a graph G_1 which is isomorphic to G but not the same.
- a graph G_2 which is graph equivalent to G but not isomorphic to it.
- a graph G_3 which is a strict homomorphism of G but not graph equivalent to it.
- a graph G_4 which is a non-strict homomorphism of G but not graph equivalent to it.
- a graph G_5 that is the transition system induced by the abstraction α that maps states that are in the column i in the image above to the abstract state s_i. For example, the two states in the first column are mapped to an abstract state t_1, the two states in the second column to an abstract state t_2, and so on.
- a graph G_6 that is the induced transition system of an abstraction β that is a non-trivial coarsening of α.
- a graph G_7 that is the induced transition system of an abstraction γ that is a non-trivial refinement of β but different from α.

In *all graphs*, highlight an optimal path and compute its cost. For graphs G_1–G_4, justify (one sentence is enough) why they don't have the property they are not supposed to have, for example, why G_2 is not isomorphic to G. For graph G_5, justify why the graph is an abstraction of G. For graphs G_6–G_7, justify why the graphs are a coarsening and a refinement.

Exercise D.2 (2 marks)(Lecture D3)

Point out the problems with the following ideas for abstraction mappings in the beleaguered castle domain:

α_1: For each card value v there is one abstract state representing all world states where v is the highest undiscarded value.

α_2: A state is mapped to an abstract state by ignoring the suit of the top card on each tableau pile.

α_3: There are up to $n = 10^6$ abstract states $s_0, \ldots, s_n$. A world state s is mapped to the abstract state s_k, where k is the MD5 hash of s modulo 10^6.

α_4: All states s with $0 \leq h^*(s) < 5$ are mapped to the first abstract state, all states s with $5 \leq h^*(s) < 10$ are mapped to the second abstract state, and so on.

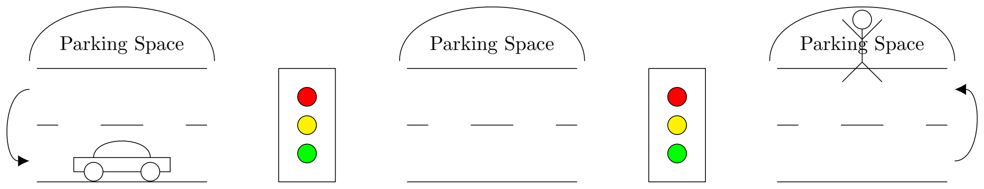
Exercise D.3 (1+4+2 marks)(Lecture D3)

(a) Prove the following claim from the lecture: let α_1 and α_2 be abstractions of a transition system $\mathcal{T}$. If no label of $\mathcal{T}$ affects both $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$, then α_1 and α_2 are orthogonal.

(b) Let Π be a SAS$^+$ planning task that is not trivially unsolvable and does not contain trivially inapplicable operators, and let P be a pattern for Π. Prove that $\mathcal{T}(\Pi|_P) \overset{\text{G}}{\sim} \mathcal{T}(\Pi)^{\pi_P}$, i.e., $\mathcal{T}(\Pi|_P)$ is graph-equivalent to $\mathcal{T}(\Pi)^{\pi_P}$.

(c) Discuss the theorem from exercise (d). First, discuss why it is relevant. Why would we need to define $\Pi|_P$, if we already saw that π_P is a valid abstraction of $\mathcal{T}(\Pi)$, and hence we could use h^{π_P} as our heuristic? Second, discuss why is it important to exclude trivially unsolvable tasks or trivially inapplicable operators.

Exercise D.4 (3+2+1+2)(Lecture D4)



In this exercise, we work with the shown driving scenario. There are 3 street segments s_1, s_2, s_3. You can drive from one segment to an adjacend segment if the traffic light is green and if you are on the correct lane, i.e., if you are on the bottom lane you can drive only to the right. You can switch lanes only at the end of the road. It is your goal to pick up a passenger and then to park in the middle segment.
More formally, the scenario is defined by the following FDR $\Pi = \langle V, O, I, \delta \rangle$ task:

- $V = \{\mathit{Segment}, \mathit{Lane}, \mathit{Passenger}, \mathit{TrafficLight}_1, \mathit{TrafficLight}_2\}$, with
  $\mathit{dom}(\mathit{Segment}) = \{s_1, s_2, s_3\}$
  $\mathit{dom}(\mathit{Lane}) = \{b, t, p\}$
  $\mathit{dom}(\mathit{Passenger}) = \{s_1, s_2, s_3, \mathit{car}\}$
  $\mathit{dom}(\mathit{TrafficLight}_i) = \{r, y, g\}$ for $i \in \{1, 2\}$
- $I = \{\mathit{Segment} = s_1, \mathit{Lane} = b, \mathit{Passenger} = s_3, \mathit{TrafficLight}_1 = r, \mathit{TrafficLight}_2 = y\}$
- $O =$

$$\begin{aligned}
&\{\langle \mathit{Segment} = s_i \wedge \mathit{Lane} = b \wedge \mathit{TrafficLight}_i = g, \mathit{Segment} := s_i + 1\rangle \mid i \in \{1, 2\}\} \cup \\
&\{\langle \mathit{Segment} = s_i \wedge \mathit{Lane} = t \wedge \mathit{TrafficLight}_{i-1} = g, \mathit{Segment} := s_i - 1\rangle \mid i \in \{2, 3\}\} \cup \\
&\{\langle \mathit{Segment} = s_1 \wedge \mathit{Lane} = t, \mathit{Lane} := b\rangle\} \cup \\
&\{\langle \mathit{Segment} = s_3 \wedge \mathit{Lane} = b, \mathit{Lane} := t\rangle\} \cup \\
&\{\langle \mathit{Lane} = t, \mathit{Lane} := p\rangle\} \cup \\
&\{\langle \mathit{Segment} = s_i \wedge \mathit{Passenger} = s_i, \mathit{Passenger} = \mathit{car}\rangle \mid i \in \{1, 2, 3\} \cup \\
&\{\langle \top, \mathit{TrafficLight}_i = x\rangle \mid i \in \{1, 2\} \text{ and } x \in \{r, y, g\}\}
\end{aligned}$$

- $\delta = \mathit{Segment} = s_2 \wedge \mathit{Lane} = p \wedge \mathit{Passenger} = \mathit{car}$

(a) Provide the syntactic projection $\Pi|_P$ of Π for the pattern $P = \{\mathit{Segment}, \mathit{Lane}\}$.

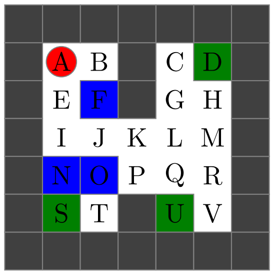
(b) Draw the transition system induced by the task $\Pi|_P$. Do not label the transitions. If there are multiple transitions from one state to another, draw only one arrow for all transitions.

(c) Calculate for every state in the transition system of (b) its distance to the goal.

(d) A PDB stores the goal distance of all abstract states in a one dimensional lookup table and uses a perfect hash function to calculate for a given state its table index. Provide the lookup table for the pattern $P = \{\mathit{Segment}, \mathit{Lane}\}$. Use the distances calculated in (c). Annotate every table entry with its associated abstract state. Order the variables in the pattern as follows: *Segment*, *Lane*

Exercise D.5 (3+3+1 marks)(Lecture D5)

In the *Sokoban* domain, a worker has to push boxes to goal positions, but cannot pull them. The figure below illustrates an example problem. The red dot denotes the initial position of the worker, the blue cells denote the initial positions of the boxes, and the green cells denote the goal positions of the boxes, where it does not matter which box is finally located at which goal position. The letters (A – V) are only shown to indicate the cells.



Consider the SAS$^+$ representation of this Sokoban problem with variables pos_w, pos_{b1}, pos_{b2}, pos_{b3} (which denote the positions of the worker and the three boxes), $atgoal_{b1}$, $atgoal_{b2}$, $atgoal_{b3}$ (which indicate whether the boxes are at goal positions), and $content_A$, ..., $content_V$ (which denote the content of the individual cells). Formally, the variable domains are defined as follows:

- $dom(pos_w) = dom(pos_{b1}) = dom(pos_{b2}) = dom(pos_{b3}) = \{A, \dots, V\}$
- $dom(atgoal_{b1}) = dom(atgoal_{b2}) = dom(atgoal_{b3}) = \{\mathbf{T}, \mathbf{F}\}$
- $dom(content_A) = \dots = dom(content_V) = \{empty, w, b1, b2, b3\}$

The initial state is defined by the set consisting of the following mappings:

- $pos_w \mapsto A$, $pos_{b1} \mapsto F$, $pos_{b2} \mapsto O$, $pos_{b3} \mapsto N$, $atgoal_{b1} \mapsto \mathbf{F}$, $atgoal_{b2} \mapsto \mathbf{F}$, $atgoal_{b3} \mapsto \mathbf{F}$
- $content_F \mapsto b1$, $content_O \mapsto b2$, $content_N \mapsto b3$, $content_A \mapsto w$
- $content_x \mapsto empty$ for all $x \in \{A, \dots, V\} \setminus \{A, F, N, O\}$

The goal is given by the formula $atgoal_{b1} = \mathbf{T} \wedge atgoal_{b2} = \mathbf{T} \wedge atgoal_{b3} = \mathbf{T}$. The operators (*move* and *push*) are defined as usual (recall that it is not allowed to pull boxes). We call cells c and c' *adjacent* if c' is either above, below, left or right to c (i.e., diagonal cells are *not* adjacent).

- *move* operators: For adjacent cells c and c', the worker can move from c to c' if the worker is currently at c and c' is empty. After moving, c is empty and the worker is at c'.
- *push* operators: For cells c, c', c'' such that c is adjacent to c' in direction X iff c' is adjacent to c'' in direction X for $X \in \{\text{above}, \text{below}, \text{left}, \text{right}\}$, the worker can push a box bi from c' to c'' if the worker is at c, the box is at c' and c'' is empty. After pushing, c is empty, the worker is at c', and the box is at c''. $atgoal_{bi}$ is set depending on whether the box moved from a nongoal to a goal position or vice versa.

Consider the pattern collection $\mathcal{C}$ that consists of exactly the following patterns:

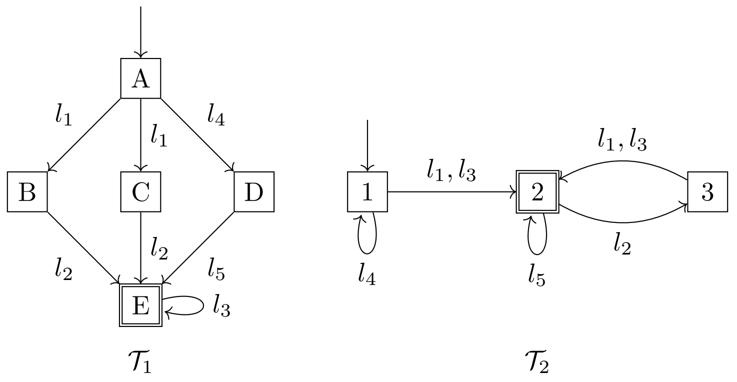
- $P_1 = \{atgoal_{b1}, pos_{b1}\}$
- $P_2 = \{atgoal_{b1}, content_H, content_M\}$
- $P_3 = \{atgoal_{b3}, pos_w\}$
- $P_4 = \{atgoal_{b3}, content_A\}$
- $P_5 = \{atgoal_{b2}, pos_{b2}, atgoal_{b3}\}$
- $P_6 = \{pos_{b1}, content_D, content_E\}$
- $P_7 = \{atgoal_{b2}, pos_{b2}\}$

(a) Simplify the collection by removing trivial patterns and causally irrelevant variables from patterns.

(b) Construct the compatibility graph for $\mathcal{C}$ and determine the maximal cliques.

(c) Provide the canonical heuristic $h^{\mathcal{C}}$ and simplify it with help of the Dominated Sum Theorem if possible.

Exercise D.6 (2+3 marks)(Lecture D8)
Consider a set $X = \{\mathcal{T}_1, \mathcal{T}_2\}$ of abstract transition systems with identical label set $L = \{l_1, \ldots, l_7\}$ and cost function c such that $c(l_1) = 1$ and $c(l_2) = c(l_3) = c(l_4) = c(l_5) = 2$. $\mathcal{T}_1$ and $\mathcal{T}_2$ are depicted graphically below.



(a) In the lecture, we have seen that merge-and-shrink is a powerful framework for computing abstractions through the means of applying transformations to factored transition systems. *Shrinking* is one type of such transformations, and it means to apply an abstraction to a single transition system of the factored transition system. In practice, given a transition system and a size limit imposed on it, the question is *how* to come up with a good abstraction of the factor. This is what a *shrink strategy* does: it is an algorithm that computes an abstraction of a given transition system so that its new size is guaranteed to obey a given limit.

For this exercise, we ignore the size limit and consider the following two simple shrink strategies. (Maximal) h-preserving shrinking abstracts all states with the same h-value to the same abstract state. f-preserving shrinking abstracts all states that have the same h-value and the same g-value, hence the name.

Graphically provide the transition systems $\mathcal{T}_1'$ and $\mathcal{T}_1''$ that result from shrinking $\mathcal{T}_1$ with h-preserving and f-preserving shrinking, respectively.

(b) Graphically provide the transition systems $\mathcal{T}_1' \otimes \mathcal{T}_2$, $\mathcal{T}_1'' \otimes \mathcal{T}_2$, and $\mathcal{T}_1 \otimes \mathcal{T}_2$. How do they compare with respect to size and heuristic value of the initial state?

Exercise D.7 (2+2+3 marks)(Lecture D8)

In this exercise, you are asked to implement and evaluate a *shrink strategy* in Fast Downward. In the lecture, we have seen that merge-and-shrink is a powerful framework for computing abstractions through the means of applying transformations to factored transition systems. Shrinking is one type of such transformations, and it means to apply an abstraction to a single transition system of the factored transition system. In practice, given a transition system and a size limit imposed on it, the question is *how* to come up with a good abstraction of the factor. This is what a shrink strategy does: it is an algorithm that computes an abstraction of a given transition system so that its new size is guaranteed to obey a given limit.
In Fast Downward, a shrink strategy is given a transition system and an object that allows to retrieve distances for states of the transition system. The strategy then has to compute an equivalence relation over the states, i.e., a partitioning over states. All states of the same equivalence class (or partition) are then mapped to the same new abstract state. Your task will be to implement the logic of the strategy for computing the state equivalence relation (the state partitioning).

(a) In `fast-downward/src/search/merge-and-shrink/shrink_h_preserving.cc` you find an incomplete implementation of a h-preserving shrink strategy that aims at abstracting all states with the same h-value to the same abstract state. It works as follows: first, partition all states of the transition system according to their h-value. Then, iterate over all partitions in increasing order of their h-value and simply assign all states of a partition to the same equivalence class in the result, as long as this does not violate the given size limit. If the size of the resulting equivalence relation reaches the given size limit, then the strategy cannot turn each partition into a separate equivalence class anymore, but instead, it assigns all states of all remaining partitions to the last equivalence class created (i.e., the last equivalence class possibly holds states that have different h-values due to the size limit).

You can test your strategy using
```
./fast-downward/fast-downward.py --alias mas-h-preserving-x blocks/probBLOCKS-6-0.pddl
```
where $x \in \{1, 10, 100, 1000\}$ denotes the size limit imposed on transition systems.

(b) In `fast-downward/src/search/merge-and-shrink/shrink_random.cc` you find an incomplete implementation of a random shrink strategy that abstracts states uniformly at random. This can be implemented as follows: Iterate over all states in a random order. As long as the size of the resulting equivalence relation has not reached the imposed size limit, assign the state to its own equivalence class, thus increasing the size of the resulting equivalence relation by 1. Once the size limit is reached, assign the state to a random equivalence class (in this case, there are exactly as many equivalence classes as the size limit allows).

You can test your strategy using
```
./fast-downward/fast-downward.py --alias mas-random-x blocks/probBLOCKS-6-0.pddl
```
where $x \in \{1, 10, 100, 1000\}$ denotes the size limit imposed on transition systems.

(c) Evaluate both strategies on the tasks in the directory `blocks`, using all four size limits ($\{1, 10, 100, 1000\}$). You can limit time for each run to 1 minute. For each run, report the runtime of the merge-and-shrink algorithm ("Merge-and-shrink algorithm runtime: "), the total runtime ("Total time: "), and the number of expanded states ("Expanded until last jump: "), which denotes the number of expanded states excluding the last f-layer of the A* search. (On the last f-layer, the number of expanded states only depends on tie-breaking which we don't want to include in our evaluation.) Discuss the results. (In particular, *explain* the results. Note that this can, to some extent, also be done without a working implementation.)

Please structure your table as follows:

Shrink strategy	$h-1$	$h-10$	$h-100$	$h-1000$	$r-1$	$r-10$	$r-100$	$r-1000$
blocks 6-0, M&S:								
blocks 6-0, Exp:								
blocks 6-0, Total:								
blocks 8-0, M&S:								
…								

Submission rules:

- Exercise sheets must be submitted in groups of two or three students. Please submit one single copy of the exercises per group (only one member of the group does the submission).
- Create a single PDF file (ending .pdf) for all non-programming exercises. If you want to submit handwritten parts, include their scans in the single PDF in a reasonable resolution, so that they are readable but the PDF size is not astronomically large. Put the names of all group members on top of the first page. Use page numbers or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).
- For programming exercises, only create and submit those code textfiles required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it!
- For the submission, you can either upload the single PDF or prepare a ZIP file (ending .zip, .tar.gz or .tgz; not .rar or anything else) containing the single PDF and the code textfile(s) and nothing else. Please do not use directories within the ZIP, i.e., zip the files directly.
- Name all files without spaces.
- Only upload one submission per group. Do not upload several versions, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.