Planning and Optimization

M. Helmert, G. Röger
C. Büchner, T. Keller, S. Sievers

University of Basel
Fall Semester 2021

Exercise Sheet D

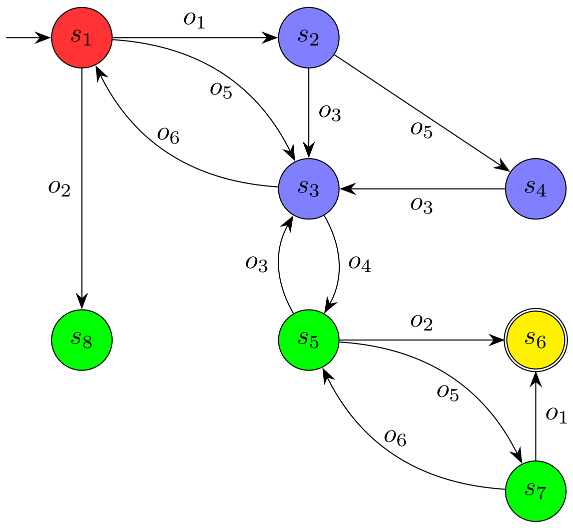
Due: November 25, 2021

Important: for submission, consult the rules at the end of the exercise. Non-adherence to these rules might lead to a penalty in the form of a deduction of marks or, in the worst case, that your submission will not be corrected at all.

The files required for this exercise are in the directory `exercise-d` *of the course repository (*`https://github.com/aibasel-teaching/planopt-hs21`*). All paths are relative to this directory. Update your clone of the repository with* `git pull` *to see the files. In the virtual machine,* `/vagrant/plan-opt-hs21` *is the repository.*

Exercise D.1 (1+1+1+1.5+2+1.5 marks)(Lecture D2)

Consider the transition system $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$ with S, L, T, s_0 and $S_\star$ as depicted below and with $c(o_i) = i$ for all $1 \leq i \leq 6$.



(a) Graphically provide a transition system $\mathcal{T}_1$ such that $\mathcal{T}_1 \sim \mathcal{T}$ and $\mathcal{T}_1 \neq \mathcal{T}$ and provide the functions φ and λ that are used in the definition of isomorphic transition systems.

(b) Graphically provide a transition system $\mathcal{T}_2$ such that $\mathcal{T}_2 \overset{G}{\sim} \mathcal{T}$ and $\mathcal{T}_2 \not\sim \mathcal{T}$. Provide the function φ that is used in the definition of graph-equivalent transition systems and argue why $\mathcal{T}_2 \not\sim \mathcal{T}$.

(c) Consider the abstraction α that maps all states depicted in the same color to the same abstract state, i.e., $\alpha(s_1) = s_r$, $\alpha(s_2) = \alpha(s_3) = \alpha(s_4) = s_b$, $\alpha(s_5) = \alpha(s_7) = \alpha(s_8) = s_g$ and $\alpha(s_6) = s_y$. Graphically provide $\mathcal{T}^\alpha$ and give h^α.

(d) Assume you may change the abstraction α from part (c) by mapping one concrete state to another (already existing) abstract state. If you care about having some positive effect on the heuristic quality, which change do you make? Justify your answer. (There are multiple reasonable options.)

(e) Provide an abstraction β of $\mathcal{T}$ such that $|S^\beta| = 4$ and such that there is no abstraction $\beta' \neq \beta$ with $|S^{\beta'}| = 4$ and $h^{\beta'}(s_1) > h^\beta(s_1)$. Graphically provide the transition system $\mathcal{T}^\beta$.

(f) Consider the abstraction α from part (c). Provide an abstraction α' of $\mathcal{T}$ that is a coarsening of α such that $\alpha' = \alpha'' \circ \alpha$ with $h^{\alpha'}(s_1) = h^{\alpha}(s_1)$. Provide the function α''.

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

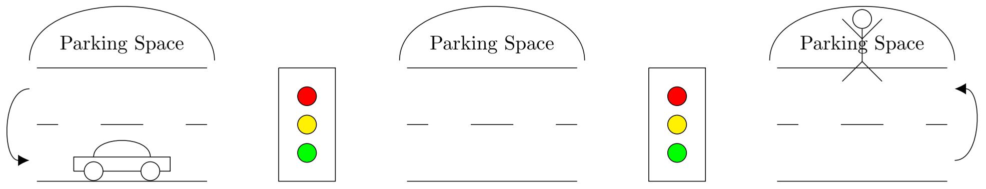
(a) Prove the following claim from the lecture: let α_1 and α_2 be abstractions of a transition system $\mathcal{T}$. If no label of $\mathcal{T}$ affects both $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$, then α_1 and α_2 are orthogonal.

(b) Prove the following claim from the lecture: Let Π be a SAS$^+$ planning task, and let P be a pattern for Π. Prove that $\mathcal{T}(\Pi|_P) \overset{G}{\sim} \mathcal{T}(\Pi)^{\pi_P}$.

Hint: it might be useful to first define the transition systems $\mathcal{T}(\Pi)$, $\mathcal{T}(\Pi|P)$, and $\mathcal{T}(\Pi)^{\pi_{\{p\}}}$, then write down the conditions for graph-equivalence and then prove the claim by proving the conditions applied to the transition systems.

(c) Discuss the theorem from exercise (b). First, discuss why it is relevant, i.e.: 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 it is important that Π (as a SAS$^+$ task) only has *satisfiable* conditions as the goal and in preconditions of operators.

Exercise D.3 (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 unit-cost FDR $\Pi = \langle V, I, O, \gamma \rangle$ task:

- $V = \{\mathit{Segment}, \mathit{Lane}, \mathit{Passenger}, \mathit{TrafficLight_1}, \mathit{TrafficLight_2}\}$, with
  $dom(\mathit{Segment}) = \{s_1, s_2, s_3\}$
  $dom(\mathit{Lane}) = \{b, t, p\}$
  $dom(\mathit{Passenger}) = \{s_1, s_2, s_3, car\}$
  $dom(\mathit{TrafficLight_i}) = \{r, y, g\}$ for $i \in \{1, 2\}$
- $I = \{\mathit{Segment} = s_1, \mathit{Lane} = b, \mathit{Passenger} = s_3, \mathit{TrafficLight_1} = r, \mathit{TrafficLight_2} = y\}$
- $O =$
  $$\begin{aligned}&\{\langle \mathit{Segment} = s_i \wedge \mathit{Lane} = b \wedge \mathit{TrafficLight_i} = g, \mathit{Segment} := s_{i+1}, 1\rangle \mid i \in \{1,2\}\} \cup\\ &\{\langle \mathit{Segment} = s_i \wedge \mathit{Lane} = t \wedge \mathit{TrafficLight_{i-1}} = g, \mathit{Segment} := s_{i-1}, 1\rangle \mid i \in \{2,3\}\} \cup\\ &\{\langle \mathit{Segment} = s_1 \wedge \mathit{Lane} = t, \mathit{Lane} := b, 1\rangle\} \cup\\ &\{\langle \mathit{Segment} = s_3 \wedge \mathit{Lane} = b, \mathit{Lane} := t, 1\rangle\} \cup\\ &\{\langle \mathit{Lane} = t, \mathit{Lane} := p, 1\rangle\} \cup\\ &\{\langle \mathit{Segment} = s_i \wedge \mathit{Passenger} = s_i, \mathit{Passenger} = car, 1\rangle \mid i \in \{1,2,3\}\} \cup\\ &\{\langle \top, \mathit{TrafficLight_i} = x, 1\rangle \mid i \in \{1,2\} \text{ and } x \in \{r,y,g\}\}\end{aligned}$$
- $\delta = \mathit{Segment} = s_2 \wedge \mathit{Lane} = p \wedge \mathit{Passenger} = car$

(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 $P = \{\mathit{Segment}, \mathit{Lane}\}$.

- Show how you calculated the indices for the states.
- Use the goal distances calculated in (c).
- Annotate every table entry with its associated abstract state.
- Order the variables in the pattern as *Segment*, *Lane* and the values of *Segment* as s_1, s_2, s_3 and the values of *Lane* as b, t, p

Exercise D.4 (4+3+2 marks)(lecture D4)

Note: to simplify implementation details, for the exercises in this part you can assume that the planning tasks that you have to deal with possess a simplified structure. In particular, you can assume that they are SAS$^+$ *tasks with the additional restrictions that (i) for every operator o, the set of state variables that appear in pre(o) is the same as the set of state variables that appear in eff(o), and (ii) the goal formula mentions all the state variables of the problem, which implies that there is one single goal state. The tasks are converted to this simplified form automatically without you having to do anything about it, so you can safely assume that conditions (i) and (ii) always hold. This simplified form, by the way, is called* Transition Normal Form *(TNF), and is useful to make the proofs of theorems and implementation of algorithms easier. You can find more details about the way TNF tasks are represented in the code in file* `fast-downward/src/search/planopt_heuristics/tnf_task.h`.

(a) In the files `fast-downward/src/search/planopt_heuristics/projection.*` you can find an incomplete implementation of a class projecting a TNF task to a given pattern. Complete the implementation by projecting the initial state, the goal state and the operators.

The example task from the lecture and two of its projections are implemented in the files `projection_test.*`. *You can use them to test and debug your implementation by calling Fast Downward as* `./fast-downward.py --test-projections`.

(b) In the files `fast-downward/src/search/planopt_heuristics/pdb.*` you can find an incomplete implementation of a pattern database. Complete the implementation by computing the distances for all abstract states as described in the code comments. You can run your implementation using the heuristic `planopt_pdb(pattern=greedy(1000))` in an A* search. It uses a greedy pattern generation algorithm which results in an abstract state space with at most 1000 states.

You can debug your code by comparing it to the built-in PDB implementation of Fast Downward. The according heuristic is `pdb(pattern=greedy(1000))` which should find the same pattern. Make sure the patterns are identical (printed as “Greedy generator pattern”) and that both implementations result in the same amount of expanded states before the last f-layer (printed as “Expanded until last jump”).

Examples from the `castle` directory:

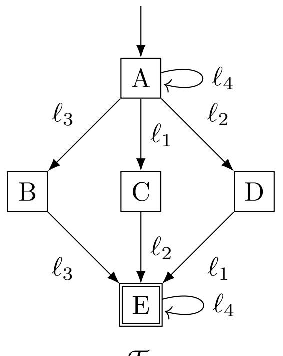
instance	pattern	expanded until last jump
castle-8-5-4-cards	[56, 57, 58, 59, 60, 61, 62, 63, 64]	3718
castle-8-5-10-cards	[57, 58, 59, 60, 61, 62, 63, 64, 65]	744
castle-8-5-13-cards	[56, 57, 58, 59, 60, 61, 62, 63, 64]	6742

(c) Analyze what effect the abstraction size has on the search performance. To do so, run your implementation of the PDB heuristic on all problem instances of the `castle` domain with an abstract state spaces of at most 1000 states as well as at most 100000 states. Then, compare the preprocessing time (i.e., difference between total time and search time), search time, and expanded states before the last f-layer.

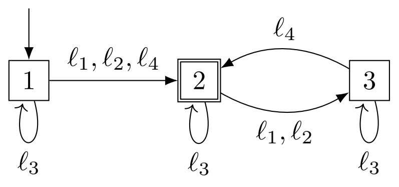
The bash script in the directory `scripts` can be used as a starting point to run the experiments.

Exercise D.5 (2+3 marks)(Lecture D7, D8)

Consider a factored transition system $F = \{\mathcal{T}_1, \mathcal{T}_2\}$ of transition systems with label set $L = \{\ell_1, \ldots, \ell_4\}$ and cost function c where $c(\ell_1) = 1$ and $c(\ell_2) = c(\ell_3) = c(\ell_4) = 2$. $\mathcal{T}_1$ and $\mathcal{T}_2$ are depicted graphically below.



$\mathcal{T}_1$



$\mathcal{T}_2$

(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. Both shrink strategies never combine a non-goal state with a goal state. h-preserving shrinking abstracts all states with the same h-value by mapping them to the same abstract state. f-preserving shrinking combines all states that have the same f-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. (Note that this means to apply both shrinking strategies on the *original* transition system $\mathcal{T}_1$.)

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

Exercise D.6 (5 marks)(Lecture D7, D8)

Label reduction is another transformation in the merge-and-shrink framework next to merging and shrinking. In its current form, called "generalized label reduction", it has been defined in the following paper:

Sievers S., Wehrle M., and Helmert H. (2014). Generalized Label Reduction for Merge-and-Shrink Heuristics *Proceedings of AAAI 2014*, 2358–2366.

Read the relevant parts of the paper and explain, *using your own words (= not copying)*, what label reduction is and how it works. In particular, describe the properties this transformation has (i.e., is it conservative, is it exact) and what conditions are necessary for these properties to hold. Note that the notation of a "conservative" transformation from the lecture is called "safe" in the paper. You can exclude the section "Label Reduction: State of the Art" (which only deals with the previous non-generalized label reduction) and everything starting from the section "Experiments" on page 5. Your write-up should be at most half a page as type set in default LaTeX.

Submission rules:

- Exercise sheets must be submitted in groups of three students. Please submit a 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. Use a file name that does not contain any spaces or special characters other than the underscore "_". If you want to submit handwritten solutions, include their scans in the single PDF. Make sure it is in a reasonable resolution so that it is readable, but ensure at the same time that the PDF size is not astronomically large. Put the names of all group members on top of the first page. Either use page numbers on all pages 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 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. Code that does not compile or which we cannot successfully execute will not be graded.
- For the submission: if the exercise sheet does not include programming exercises, simply upload the single PDF. If the exercise sheet includes programming exercises, upload 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. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to ADAM, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.