Planning and Optimization

G. Röger, T. Keller
G. Francès

University of Basel
Fall Semester 2018

Exercise Sheet D

Due: November 13, 2018

The files required for this exercise are in the directory **`exercise-d`** *of the course repository (`https://bitbucket.org/aibasel/planopt-hs18`). All paths are relative to this directory. Update your clone of the repository with* **`hg pull -u`** *to see the files. For the runs with Fast Downward, set a time limit of 1 minute and a memory limit of 2 GB. Using Linux, such limits can be set with* `ulimit -t 60` *and* `ulimit -v 2000000`*, respectively.*

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

(a) 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.

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

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.

(b) 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.

(c) 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.

(d) 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 $\mathcal{T}(\Pi)^{\pi_P}$.

(e) 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.2 (4+3+2+3+2 marks)

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`.
The bash scripts in the directory `scripts` *can be extended to run the experiments that you will need to answer some of the questions.*

(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 method `test_projections`. *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 use the built-in implementation of Fast Downward to debug your code as explained in exercise (c).

(c) Use the heuristic `pdb(pattern=greedy(1000))` to find a good pattern with at most 1000 abstract states for each instance in the directory `castle`. Then run your implementation from exercise (b) using the heuristic `planopt_pdb(pattern=`P`)`. For each instance use the same pattern P used by the built-in implementation.

Compare the two implementations and discuss the preprocessing time, the search time, and the number of expanded states excluding the last f-layer (printed as “Expanded until last jump”). Repeat the experiment for 100000 abstract states and compare the results.

(d) In the files `fast-downward/src/search/planopt_heuristics/canonical_pdbs.*` you can find an incomplete implementation of the canonical pattern database heuristic. Complete the implementation in the methods `build_compatibility_graph` and `compute_heuristic` to create the compatibility graph for a given pattern collection and for computing the heuristic value given the maximal cliques of that graph.

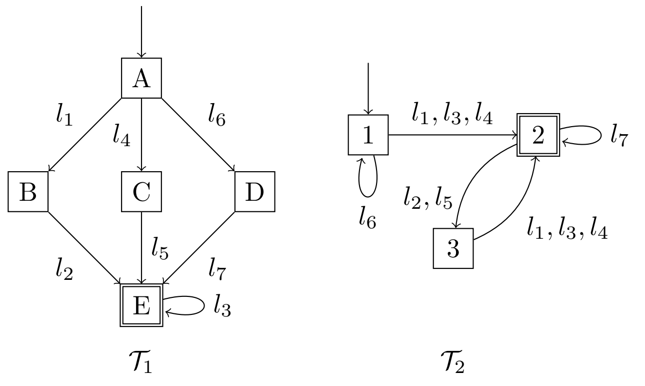
You can use the built-in implementation of Fast Downward to debug your code as explained in exercise (e).

(e) Use the heuristic `cpdbs(patterns=combo(1000))` to find a good pattern collection with at most 1000 abstract states for each instance in the directory `nomystery-opt11-strips`. Then run your implementation from exercise (d) using the heuristic `planopt_cpdbs(patterns=`C`)`. For each instance use the same pattern collection C used by the built-in implementation.

Compare the two implementations and discuss the total time, and the number of expanded states excluding the last f-layer (printed as "Expanded until last jump"). Also compare your results to using a single pattern database heuristic with up to 1000 abstract states as in exercise (c).

Exercise D.3 (5+3 marks)

(a) Consider a set $X = \{\mathcal{T}_1, \mathcal{T}_2\}$ of abstract transition systems with identical label set $L = \{l_1, \dots, l_7\}$ and cost function c such that $c(l_1) = c(l_4) = c(l_6) = 1$ and $c(l_2) = c(l_3) = c(l_5) = c(l_7) = 2$. $\mathcal{T}_1$ and $\mathcal{T}_2$ are depicted graphically below.



- Determine a mapping $\lambda : L \mapsto L'$ that maps all $\mathcal{T}_1$-combinable labels with identical cost to the same (new) label and all labels l that are not $\mathcal{T}_1$-combinable with another label to l. Let c' be the cost function that allows exact label reduction with $\langle \lambda, c' \rangle$. Graphically provide $\mathcal{T}_1^{\langle \lambda, c' \rangle}$ and $\mathcal{T}_2^{\langle \lambda, c' \rangle}$.
- Graphically provide the transition systems $\mathcal{T}_1'$ and $\mathcal{T}_1''$ that result from shrinking $\mathcal{T}_1^{\langle \lambda, c' \rangle}$ with the following shrinking strategies:
  - $\mathcal{T}_1'$ results from applying f-preserving shrinking, and
  - $\mathcal{T}_1''$ results from applying bisimulation-based shrinking.
- Graphically provide the transition systems $\mathcal{T}_1' \otimes \mathcal{T}_2^{\langle \lambda, c' \rangle}$, $\mathcal{T}_1'' \otimes \mathcal{T}_2^{\langle \lambda, c' \rangle}$, and $\mathcal{T}_1 \otimes \mathcal{T}_2$. How do they compare with respect to size and heuristic value of the initial state?

(b) Let X and X' be collections of transition systems. Why is $h(s) = h^*_{\mathcal{T}_{X'}}(\sigma(s))$ not necessarily an admissible heuristic for $\mathcal{T}_X$ if the transformation from X to X' is not safe? Discuss the question for each of the following reasons why a transformation with functions σ and λ can be unsafe:

- $c'(\lambda(l)) > c(l)$ for at least one $l \in L$
- there is a transition $\langle s, l, t\rangle$ of $\mathcal{T}_X$ such that $\langle \sigma(s), \lambda(l), \sigma(t)\rangle$ is not a transition of $\mathcal{T}_{X'}$, or
- there is a goal state s of $\mathcal{T}_X$ such that $\sigma(s)$ is not a goal state of $\mathcal{T}_{X'}$.

The exercise sheets can be submitted in groups of three students. Please submit one single copy of the exercises per group (only one member of the group does the submission), and provide all student names on the submission.