Planning and Optimization

M. Helmert, G. Röger
F. Pommerening

University of Basel
Fall Semester 2017

Exercise Sheet E

Due: December 5, 2017

The files required for this exercise are in the directory `exercise-e` *of the course repository (*`https://bitbucket.org/aibasel/planopt-hs17`*). All paths are relative to this directory. Update your clone of the repository with* `hg pull -u` *to see the files.*

Exercise E.1 (4+4+4+3 marks)

(a) In the files `fast-downward/src/search/planopt_heuristics/critical_paths.*` you can find an incomplete implementation of the h^m heuristic. Complete the implementation of the methods `current_cost` and `compute_heuristic_basic` according to the algorithm in slides E1.

The methods in the file `set_utils.h` *might be useful for manipulating partial states (represented as sets of proposition IDs).*

(b) The implementation from exercise (a) is inefficient. Use a profiler to find out why and implement an improved version in the method `compute_heuristic_improved`. Clearly document the improvements you made and describe why you made them.

One way to profile the code is to use the tool callgrind from the profiling suite valgrind. You can call it on your implementation as follows

```
./fast-downward.py --translate /path/to/task.pddl
valgrind --tool=callgrind ./builds/release32/bin/downward \
         --search "astar(planopt_hm(m=2, improved=false))" < output.sas
```

Change the flag "improved" to true if you want to test your improved implementation. Examples for possible improvements: precomputation of data that is computed again and again; memoization; better choices to select the candidates in the h^m algorithm; adding indirection to avoid expensive copies, ...

(c) Design, run, and report on an experiment that compares the built-in implementation (called with `hm(m=2)`), your basic implementation, and your improved implementation of h^2. Your experiment should demonstrate that the three implementations compute the same heuristic function and that your improvement from exercise (b) significantly improved the performance of the computation.

(d) Consider the task $\Pi = \langle V, I, O, G \rangle$ with the set notation $V = \{w, x, y, z\}$, $I = \{w, x\}$, $O = \{o_1, o_2\}$, $o_1 = \langle \{x\}, \{y\}, \{w\}, 1 \rangle$, $o_2 = \langle \{y\}, \{z\}, \emptyset, 1 \rangle$, and $G = \{x, y, z\}$. Construct Π^2 and show that there is an admissible heuristic h such that $h^*_\Pi(I) < h_{\Pi^2}(I^2)$.

Exercise E.2 (4+3+3+2+4 marks)

(a) In the files `fast-downward/src/search/planopt_heuristics/justification_graph.*` you can find an incomplete implementation of the justification graph used by the LM-cut heuristic. Complete the implementation of the constructor and the methods `mark_goal_zone` and `find_cut_edges` by following the comments in the code. Then compute the LM-cut algorithm in the method `compute_heuristic` of the file `lmcut.cc`.

The $h^{\max}$ computation is integrated into the class `DeleteRelaxedNormalFormTask` *for efficiency. The classes for operators and propositions track their $h^{\max}$ achievers and costs. Calling the method* `compute_hmax` *on the task will update these stored values.*

You can call your heuristic as `planopt_lmcut()` *. If you compare it to the built-in implementation, note that the result of each heuristic evaluation depends on the precondition-choice function that was used. The two implementations are not guaranteed to pick the same achievers, so they might give different results. However, in most tasks the heuristic value of the initial state should be similar.*

(b) Draw the justification graphs generated by LM-cut in the heuristic computation for the initial state of the task in the directory `lmcut`. Label each node of the graph with the name of the represented proposition and its $h^{\max}$ value. Label transitions with the correct operator and its current cost. Also mark the goal zone and the cut in each graph.

You can do this exercise manually but it may be easier to adapt your algorithm from exercise (a) to print the necessary information during the computation.

(c) In the files `fast-downward/src/search/planopt_heuristics/and_or_graph.*` you can find an incomplete implementation of the algorithm discovering landmarks in AND/OR graphs. Complete the method `compute_landmarks` to compute the set of landmarks for reaching the given node in the graph.

You can use your implementation to compute disjunctive action landmarks and causal fact landmarks for I in $sRTG(\Pi^+)$ by calling the planner as `./fast-downward.py /path/to/task --compute-landmarks`*. The output lists all disjunctive action landmarks and the causal fact landmarks that are not already true in the initial state. The task in the directory* `lmcount` *contains the example task from the lecture for debugging.*

(d) Use your implementation from exercise (c) to compute action and fact landmarks for the task in the directory `blocks`.

Is the number of action landmarks an admissible heuristic for this task? Is the number of fact landmarks (without those already true in the initial state) an admissible estimate? Under which conditions are these statements true in general?

(e) Design an exercise about landmark orders for next year's exercise sheet. The goal of your exercise should be that someone who did not understand the difference between different landmark orders and what they are used for in LM-count, will understand it after completing the exercise. Your answer should include an exercise question, a model solution, and a brief discussion on why the exercise achieves the learning objective.

Exercise E.3 (3+4+2 marks)
Consider the following TNF task $\Pi = \langle V, I, O, \gamma \rangle$ with

- Variables $V = \{a, b, c\}$ with $dom(a) = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $dom(b) = \{1, 2, 3, 4, 5, 6, 7\}$, and $dom(c) = \{1, 2, 3, 4, 5\}$,
- Initial state $I = \{a \mapsto 1, b \mapsto 1, c \mapsto 1\}$,
- Operators $O = \{o_1, \ldots, o_{14}\}$ where
  - $o_1 = \langle (a = 1) \wedge (b = 1) \wedge (c = 1), (a := 2) \wedge (b := 2) \wedge (c := 2) \rangle$
  - $o_2 = \langle (a = 2) \wedge (b = 2) \wedge (c = 2), (a := 2) \wedge (b := 4) \wedge (c := 3) \rangle$
  - $o_3 = \langle (a = 2) \wedge (b = 2) \wedge (c = 2), (a := 3) \wedge (b := 3) \wedge (c := 2) \rangle$
  - $o_4 = \langle (a = 2) \wedge (b = 3) \wedge (c = 2), (a := 4) \wedge (b := 3) \wedge (c := 4) \rangle$
  - $o_5 = \langle (a = 3), (a := 2) \rangle$
  - $o_6 = \langle (b = 3), (b := 4) \rangle$
  - $o_7 = \langle (a = 3) \wedge (b = 3) \wedge (c = 3), (a := 5) \wedge (b := 5) \wedge (c := 5) \rangle$
  - $o_8 = \langle (a = 4) \wedge (b = 4) \wedge (c = 4), (a := 5) \wedge (b := 5) \wedge (c := 5) \rangle$
  - $o_9 = \langle (a = 5), (a := 6) \rangle$
  - $o_{10} = \langle (a = 6) \wedge (b = 5), (a := 5) \wedge (b := 6) \rangle$
  - $o_{11} = \langle (a = 6) \wedge (b = 6), (a := 7) \wedge (b := 6) \rangle$
  - $o_{12} = \langle (a = 7) \wedge (b = 7), (a := 6) \wedge (b := 7) \rangle$
  - $o_{13} = \langle (a = 7) \wedge (b = 6), (a := 8) \wedge (b := 7) \rangle$
  - $o_{14} = \langle (a = 8) \wedge (b = 7), (a := 7) \wedge (b := 7) \rangle$

  and $cost(o) = 1$ for all $o \in O$,
- Goal $\gamma = (a = 6) \wedge (b = 7) \wedge (c = 5)$.

(a) Provide the LP solved by the flow heuristic for I as an input file for the solver *lp-solve*. Use each atom as the constraint name for its flow constraint so it is easy to see which constraint belongs to which atom. Then solve the LP and provide the objective value and values for all variables Count_o in the discovered solution.

On Ubuntu you can install lp-solve with `sudo apt install lp-solve`. *Its input format is described on* `http://lpsolve.sourceforge.net/5.5/lp-format.htm` *and in the example file in the directory* `lp`. *You can run lp-solve using* `lp_solve /path/to/file.lp`.

(b) Draw the transition systems of the three projections to a, b, and c. For operators that do not mention the variable, include just one representative self-loop at the goal state to keep the transition system concise. Write the value of each operator in the optimal solution from exercise (a) next to its name in the edge labels and highlight edges with a non-zero value. What do you notice in the abstractions? Discuss your observations.

(c) Dualize the LP from exercise (a). For this exercise, you can treat all flow constraints as $\geq$-constraints. For 1 Bonuspoint, dualize the general form of the LP solved by a flow heuristic.

The exercise sheets can be submitted in groups of two students. Please provide both student names on the submission.