Planning and Optimization

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

University of Basel
Fall Semester 2020

Exercise Sheet C

Due: November 1, 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 C.1 (4 marks)(Lecture C2)

The delete relaxation of SAS^+ is different from propositional tasks because there is no notion of a *negative* effect. Instead of ignoring negative effects, we can consider the variables to have sets of values where new values can only be added. For example, a variable describing the position of an agent will have the value $\{A\}$ if the agent is at A. Moving from A to B in the delete relaxation *adds* the value B to this set, so afterwards the variable has the value $\{A, B\}$ and the agent is considered to be at A and B.
Define this idea formally for arbitrary SAS^+ tasks Π. This should include a definition for Π^+ and o^+; for *satisfying a formula* ($s \models \chi$); for *domination* of states (s dominates s'); and a definition of *applying a relaxed operator* ($s[\![o^+]\!]$) in such tasks. Prove that the following results also hold with your definitions:

- Domination Lemma: Let s and s' be states and χ a formula without negation symbols. If $s \models \chi$ and s' dominates s then $s' \models \chi$.
- Monotonicity lemma: Let s be a state and o an operator. Then $s[\![o^+]\!]$ dominates s.

Refer to the proofs presented in the lecture for parts of the proofs that remain the same.

Exercise C.2 (4 marks)(Lecture C2)

Provide unit-cost planning tasks in STRIPS with the following characteristics. In cases where this is not possible, explain why and use positive normal form instead of STRIPS. If this is also not possible, justify why no such task can exist:

- A task Π_1 with 2 operators, such that Π_1^+ has an optimal plan cost of 4.
- A task Π_2 with 2 variables, such that Π_2^+ has an optimal plan cost of 3.
- A task Π_3 with at least one plan with cost 3, where Π_3^+ is unsolvable.
- An unsolvable task Π_4 such that Π_4^+ is solvable.
- An infinite family of planning tasks $P = \{P_1, P_2, \ldots\}$ (the definition of P_i is parametrized by the value of the integer parameter i) such that the optimal plan cost of P_i is twice the cost of an optimal plan for P_i^+ for all i.

Exercise C.3 (1+2+1 marks)(Lecture C5)

Take the simple instance of the *Visitall* domain (from the International Planning Competition) in the directory `visitall-untyped`, and make sure you understand the problem.

(a) What is the optimal solution value $h^*(I)$? What is the value of $h^+(I)$?

(b) Draw the full Relaxed Task Graph corresponding to the instance, and label each node with the final cost that results from (manually) applying the algorithm seen in class for computing $h^{\max}$. What is the value of $h^{\max}(I)$?

(c) Label the full Relaxed Task Graph from part (b) with the costs that result from h^{add}. Take care that it is clear which values belong to part (b) and which to part (c).

Exercise C.4 (2+2+4+2+3+2 marks)(Lectures C3,C4,C5,C5,C6,C6)

(a) The files `fast-downward/src/search/planopt_heuristics/and_or_graph.*` contain an implementation of an AND/OR graph. Implement the so-called *generalized Dijkstra's algorithm* in the method `most_conservative_valuation` to find the most conservative valuation of a given AND/OR graph by following the approach outlined in the code comments.

The example graphs from the lecture are implemented in the method `test_and_or_graphs`*. You can use them to test and debug your implementation by calling Fast Downward as* `./fast-downward.py --test-and-or-graphs`.

(b) The files `fast-downward/src/search/planopt_heuristics/relaxed_task_graph.*` contain a partial implementation of a relaxed task graph for STRIPS tasks. Complete it by constructing the appropriate AND/OR nodes and edges between them in the constructor. Also complete the method `is_goal_relaxed_reachable` by querying the AND/OR graph.

You can use the heuristic `planopt_relaxed_task_graph()` *(which prunes states that are not relaxed solvable) to test your implementation.*

(c) Modify the construction of the relaxed task graph by setting the variable `direct_cost` of each operator effect node you create to the actual cost of the operator.

Then implement the method `weighted_most_conservative_valuation` for AND/OR graphs to compute h^{add} by following the approach outlined in the code comments. Use a comment to point out the change you would have to make to turn this into a computation for $h^{\max}$.

Finally, implement the method `additive_cost_of_goal` of the relaxed task graph class to return the $h^{\texttt{add}}$ value of the task based on the implementation above.

(d) The heuristic `planopt_add()` uses your implementation from exercise (c) as heuristic values. Use it in an eager greedy search on the instances in the directory `sokoban`. Which of the instances are relaxed solvable? Which ones can you solve with this heuristic within the resource limits? Compare the heuristic values of the initial state with the cost of an optimal relaxed plan, the discovered plan and an optimal plan.

You can compute optimal relaxed plans by explicitly creating the delete relaxation of the task and solving it with an optimal search algorithm. This can be done with the Fast Downward options `--search "astar(lmcut())" --translate-options --relaxed`*. This is not the ideal way of computing optimal relaxed plans, so it will not complete on all instances. If the search does not complete, the last reached f-layer is a lower bound to the optimal relaxed solution cost.*

The values of `planopt_add()` *and the built-in implementation of Fast Downward (*`add()`*) should match, so you can use the built-in implementation for debugging exercise 2(c).*

(e) Modify your solution of exercise (c) so that every time you reduce the cost of an OR node, the ID of the responsible successor is stored in the `achiever` field of the OR node.

Then implement the method `ff_cost_of_goal` by collecting all best achievers. Start from the goal node and recursively collect all successors of each encountered AND node and the stored best achiever from each encountered OR node. Return the sum of direct costs of all collected nodes.

(f) The heuristic `planopt_ff()` uses your implementation from exercise (e) as heuristic values. Use it in an eager greedy search on the instances in the directory `sokoban` and compare the heuristic values of the initial state with the cost of an optimal relaxed plan, the discovered plan and an optimal plan. Also compare the results to the results of exercise 2(d).

The values of `planopt_ff()` *and the built-in implementation of Fast Downward (*`ff()`*) are not guaranteed to match, but should lead to similar results on these benchmarks.*

Exercise C.5 (3 marks)(Lecture C6)

Consider the following paper:

> Katz, M., Hoffmann, J. and Domshlak, C. (2013). Who Said we Need to Relax All Variables? *Proceedings of ICAPS 2013*, 126–134.

Describe the core idea of the red-black relaxation. What is the connection between the delete relaxation as defined in the lecture and the delete relaxation as described in the paper? What are the extreme cases of the red-black relaxation? Compare the cost of the optimal plan in the red-black relaxation to h^+ and $h^\star$.

Note: To answer this question, it is sufficient to read and understand the first $2\frac{1}{2}$ pages, up to (and excluding) the section titled "Bounding Variable Number and Size". Your answer **must not** exceed half a page (we correct only the first half page and ignore the rest).

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.