Planning and Optimization

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

University of Basel
Fall Semester 2021

Exercise Sheet C

Due: November 11, 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-c` *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 C.1 (1.5+0.5+0.5+0.5 marks)(Lecture C1 for (a), Lecture C2 for (b), (c) and (d))
Consider the planning task $\Pi = \langle V, I, O, \gamma \rangle$ with

- $V = \{A, B, C, D\}$
- $I = \{A \mapsto \mathbf{F}, B \mapsto \mathbf{F}, C \mapsto \mathbf{T}, D \mapsto \mathbf{T}\}$
- $O = \{o_1, o_2\}$ with $o_1 = \langle \neg A, (\neg B \rhd \neg C) \wedge \neg D \rangle$ and $o_2 = \langle \neg B, B \wedge D \rangle$
- $\gamma = \neg C \wedge D$.

(a) Transform Π to a propositional planning task $\Pi' = \langle V', I', \{o_{1'}, o_{2'}\}, \gamma' \rangle$ in positive normal form and provide the delete relaxation Π^+ of Π'.
Reminder: We covered positive normal form in Chapter A6.

(b) Provide the on-sets $\mathrm{on}(I'[\![o_{1'}]\!])$ and $\mathrm{on}(I'[\![o^+_{1'}]\!])$.

(c) Provide a state $s' \neq I'[\![o_{1'}]\!]$ from Π' that is reachable from I' and dominates $I'[\![o_{1'}]\!]$.

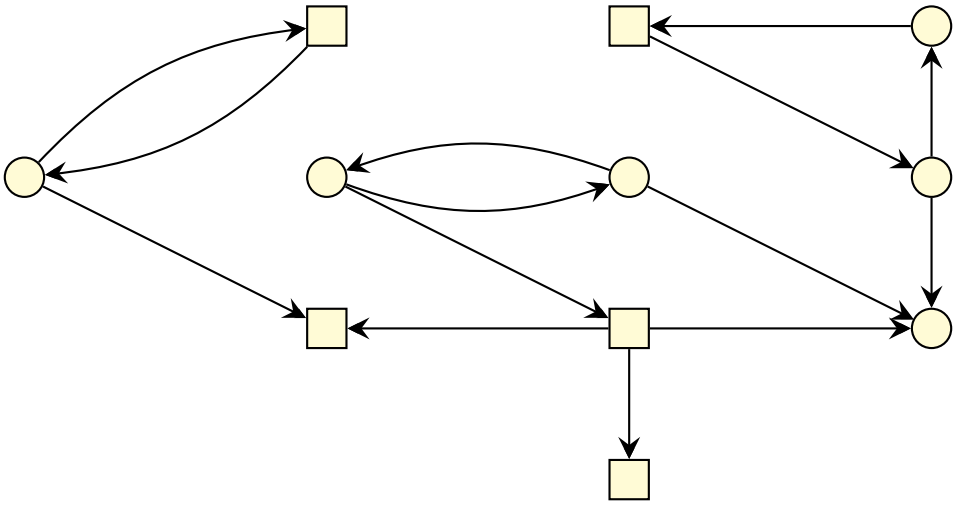
(d) Provide a state $s' \neq I'[\![o^+_{1'}]\!]$ from Π^+ that is reachable from I' and dominates $I'[\![o^+_{1'}]\!]$.

Exercise C.2 (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 in which relaxed operator o^+ is applicable. Then $s[\![o^+]\!]$ dominates s.

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

Exercise C.3 (0.5+0.5+0.5+0.5 marks)(Lecture C3)
Consider the following AND/OR graph:



Use the four smaller versions of this AND/OR graph below to annotate the nodes with truth values **T** and **F** as given by the following:

(a) Mark all forced true and forced false nodes.

(b) Mark all nodes with their truth value in the most conservative valuation.

(c) Mark all nodes with their truth value in the least conservative valuation.

(d) Mark all nodes with their truth value in a consistent valuation that is different from the most conservative and the least conservative valuation.

Exercise C.4 (1+1+1+1 marks)(Lecture C2 for (a) and (b), Lecture C5 for (c), Lecture C6 for (d))
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) A task Π with 2 operators, such that Π^+ has an optimal plan cost of 4.

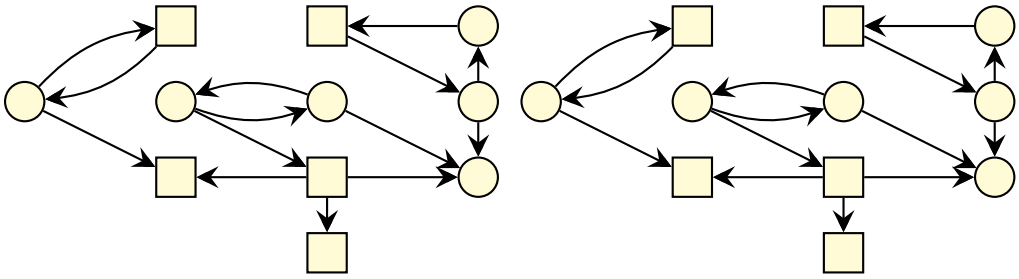
(b) A task Π with 2 variables, such that Π^+ has an optimal plan cost of 3.

(c) A task Π with at most 4 state variables and initial state I with $h^{\max}(I) > h^{\text{add}}(I)$.

(d) A task Π with at most 4 state variables and initial state I with $h^{\text{add}}(I) > h^{\text{FF}}(I)$.

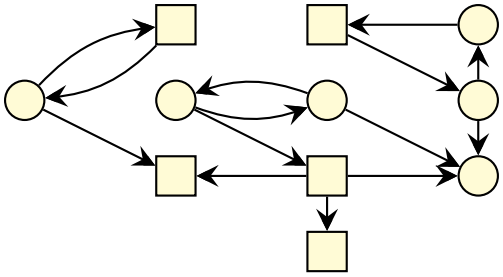
Exercise C.5 (1+2+1 marks)(Lecture C5)
Take the simple instance of the *Visitall* domain (from the International Planning Competition) in the directory `visitall-untyped`, and make sure you understand the problem.

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

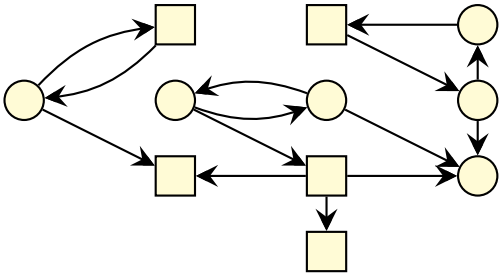


(a)

(b)



(c)



(d)

(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.6 (2+2+2+2+2 marks)(Lecture C2 for (a) and (b), Lectures C3 and C4 for (c), Lectures C5 and C6 for (d) and (e))

In this exercise, you are asked to implement several heuristics based on the delete-relaxation. For simplicity, we restrict planning tasks to STRIPS.

(a) In the file `fast-downward/src/search/planopt_heuristics/h_greedy_relaxed_plan.cc` you can find an incomplete implementation of a heuristic that estimates the goal distance as the cost of a greedily computed relaxed plan. Complete the implementation by applying operators that are applicable in the relaxed task until the goal is reached or until there are no more changes. Do not use operators that do not add anything new.

Use a time limit of 3 minutes, which can be imposed by executing `ulimit -t 180` *in the console after logging into the virtual machine.*

To test your implementation, you can use a greedy search with the heuristic `planopt_greedy_relaxed()`*, as in the following Fast Downward command line:* `--search "eager_greedy([planopt_greedy_relaxed()])"`*. Expected plan costs on some* `castle` *instances: castle-2-2-8-cards 4, castle-3-3-8-cards 9, castle-4-3-5-cards 16.*

(b) Evaluate the heuristic from part (a) in a greedy search by running it on the instances in the directory `castle`. Compare the heuristic values of the initial state with the cost of an optimal relaxed plan, the discovered plan and an optimal plan. To do this comparison, provide a table with one row per instances and, from left to right, the following values: $h^{\text{greedy}}(I)$, $h^{+}(I)$, $h^{*}(I)$ and $c(\pi)$ (cost of the discovered plan).

Discuss the results, in particular w.r.t. the relationship of the different heuristic values/plan costs.

Use a time limit of 3 minutes, which can be imposed by executing `ulimit -t 180` *in the console after logging into the virtual machine.*

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 command line `--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 in three minutes, the last reached f-layer is a lower bound to the optimal relaxed solution cost. In these cases, please provide the value x of the last reached f-layer and write an table entry of the form $\geq x$.*

(c) The file `fast-downward/src/search/planopt_heuristics/relaxed_task_graph.cc` contains 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. Associate operator effect nodes with costs (only needed for later parts of this exercise).

To test your implementation, you can use the heuristic `planopt_relaxed_task_graph()` *(which prunes states that are not relaxed solvable). Expected plan costs on some* `sokoban` *instances: prob01 32, prob02 10, prob03 9.*

(d) 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.

To test your implementation, you can use the heuristic `planopt_ff()`*. Expected plan costs on some* `sokoban` *instances: prob01 32, prob02 10, prob03 13. Furthermore, 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 the* `sokoban` *benchmarks.*

(e) Evaluate the heuristic `planopt_ff()` in a greedy search on the instances in the directory `sokoban`. 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 additive heuristic `planopt_add()`. To do this comparison, provide a table with one row per instances and, from left to right, the following values: $h^{\text{add}}(I)$, $h^{\text{ff}}(I)$, $h^{+}(I)$, $h^{*}(I)$ and $c(\pi)$ (cost of the discovered plan using h^{FF}).

Discuss the results, in particular in the light of the theoretical relationship between these heuristic/plan cost values.

Use a time limit of 3 minutes, which can be imposed by executing `ulimit -t 180` *in the console after logging into the virtual machine.*

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 command line `--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 in three minutes, the last reached f-layer is a lower bound to the optimal relaxed solution cost. In these cases, please provide the value x of the last reached f-layer and write an table entry of the form $\geq x$.*

Exercise C.7 (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 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.