Planning and Optimization

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

University of Basel
Fall Semester 2021

Exercise Sheet B

Due: October 28, 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-b` *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 B.1 (2+2+2+2 marks)(Lecture B1)

Look up the following 4 planners in planner abstracts of the deterministic tracks of the International Planning Competition (IPC) 2014 and 2018. Categorize each of them in a similar fashion as the examples in lecture B1. That is, list their problem class (satisficing or optimal), algorithm class (explicit search, SAT planning or symbolic search), the design choices of their respective class (for example the search direction for explicit search or the encoding for a SAT solver) and other aspects that stand out.

(a) Dynamic Gamer (2014)

(b) Fast Downward Stone Soup 2014

(c) OLCFF (2018)

(d) Freelunch (2018)

You can find planner abstracts on the competition websites reachable from the ICAPS website (`https://www.icaps-conference.org/competitions/`).

Exercise B.2 (4 marks)(Lectures B2, B3, and B4)

Consider the STRIPS planning task $\Pi = \langle V, O, I, \gamma \rangle$ where

- $V = \{A, B, C, D\}$;
- $O = \{o_1, o_2, o_3, o_4\}$ with
  - $o_1 = \langle A, \neg B \land C \rangle$,
  - $o_2 = \langle B, A \land D \rangle$,
  - $o_3 = \langle A \land D, B \rangle$, and
  - $o_4 = \langle A, B \land \neg C \rangle$;
- $I = \{A \mapsto \top, B \mapsto \top, C \mapsto \bot, D \mapsto \bot\}$
- $\gamma = C \land D$

Provide the breadth-first search tree resulting from regression search for this problem up to depth 3. You may prune parts of the search by marking duplicates and subsumptions as such. Provide the resulting plan if you find a solution.

Exercise B.3 (2+1 marks)(Lecture B3 for (a), Lecture B4 for (b))

(a) What is the regression $regr(\varphi, o_i)$ of the formula $\varphi = (a \lor b)$ through the following operators o_i? Simplify all results as much as possible.

- $o_1 = \langle c, a \rhd b \rangle$
- $o_2 = \langle c, \neg a \rhd b \rangle$
- $o_3 = \langle d, a \land \neg b \rangle$
- $o_4 = \langle c, \neg a \land \neg b \rangle$

(b) Based on your result to (a), which of the four operators are potentially useful in a regression search that expands search state φ, and which ones should be pruned? Justify your answer briefly.

Exercise B.4 (3+3 marks)(Lecture B3)

(a) Provide a family of planning tasks Π_n such that the size of Π_n is polynomial in n, and such that a breadth-first search with regression expands only a polynomial number of search nodes in n, whereas a breadth-first search with progression needs to expand an exponential number of search nodes in n. Assume the progression search prunes all duplicate states and the regression prunes a state if its formula logically entails the formula of its parent. Explain why your family of tasks satisfies the requirements.

(b) Provide a family of planning tasks Π_n such that the size of Π_n is polynomial in n, and such that a breadth-first search with progression expands only a polynomial number of search nodes in n, whereas a breadth-first search with regression needs to expand an exponential number of search nodes in n. Assume the same pruning as in exercise (a). Explain why your family of tasks satisfies the requirements.

Exercise B.5 (1+4+1+1)(Lecture B5)

In this exercise, your task is to define a sequential SAT-encoding for the planning task $\langle \{a, b, c\}, \{a \mapsto 1, b \mapsto 0, c \mapsto 0\}, \{o_1, o_2\}, \neg a \land c \rangle$ with $o_1 = \langle a, b \land (b \rhd \neg a) \rangle$ and $o_2 = \langle a \land b, c \rangle$.

(a) Provide the clauses that encode the initial state and the goal (use time T for the latter).

(b) Provide all clauses that encode the transitions for some time step i. Simplify the clauses and omit those that simplify to $\top$ (you don't need to provide intermediate results for the simplification). Annotate each remaining clause as precondition clause, positive or negative effect clause or positive or negative frame clause.

(c) The clauses from Exercises B.5a) and B.5b) do not suffice for a complete SAT-encoding of the planning task. Provide all missing clauses, again parametrized for some time step i.

(d) What is the smallest horizon T for which the resulting formula for the given planning task is satisfiable? Justify your answer.

Exercise B.6 (3+5 marks)(Lecture B6)

For this exercise, you need to have minisat (`minisat.se/MiniSat.html`) installed. We provide a script in the repository to do so. Simply run the following command: `./install-minisat.sh`

(a) The file `pyperplan/src/search/sat.py` already contains a complete implementation of a SAT search using a sequential encoding. Comment out the lines that add the positive frame clauses to the set of clauses. Explain why this is possible without making the SAT search compute incorrect solutions. Furthermore, investigate what effect on performance this change has experimentally. To do so, compare the runtime of the program with and without these clauses on the tasks in the directories `blocks`, `gripper` and `logistics`. You don't have to run the search longer than two minutes. Provide the runtimes in a table with one row for each task and with one column for each of the encodings. Describe the differences you observe and explain them.

You can run the code with the command
`./pyperplan/src/pyperplan.py -s sat-seq gripper/prob01.pddl`
(The domain file will be automatically inferred.)

Please note that the wallclock time printed by pyperplan can be quite off. Instead, please use the linux built-in `time` command by prepending it to the above command to obtain system wallclock time in seconds (example output: "real 0m32,177s").

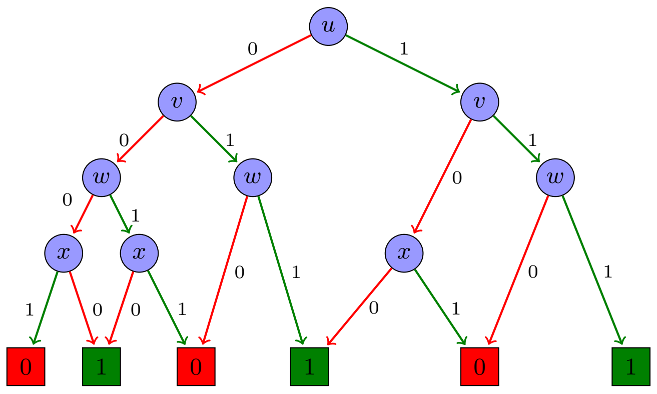
(b) The file `pyperplan/src/search/sat.py` contains an incomplete method `build_parallel_model`. Please complete the implementation using the parallel encoding presented in the lecture. You don't have to change any of the other existing methods for this task.

Test your implementation on the same tasks as in part (a), using the command
`./pyperplan/src/pyperplan.py -s sat-par gripper/prob01.pddl`.
As for part (a), provide a table with runtimes. What is the effect of the parallel encoding compared to the sequential one that you used in part (a)? Plase explain the reason for this effect.

Exercise B.7 (2+2 marks)(Lecture B7)

For both parts, you do not need to show intermediate results, but partial marks may be awarded for wrong results with correct intermediate steps.

(a) Consider the following ordered BDD:



Provide the equivalent reduced ordered BDD.

(b) Provide a reduced ordered BDD for the formula

$$\varphi = ((u \lor v) \land \neg w) \lor (u \land (v \lor (\neg w \land u))) \lor (u \land v \land \neg w) \lor (v \land w)$$

You may choose any order.

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.