Planning and Optimization

M. Helmert, T. Keller
S. Eriksson, F. Pommerening, S. Sievers

University of Basel
Fall Semester 2019

Exercise Sheet B

Due: October 20, 2018

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. The files required for this exercise are in the directory `exercise-b` *of the course repository (*`https://bitbucket.org/aibasel/planopt-hs19`*). All paths are relative to this directory. Update your clone of the repository with* `hg pull -u` *to see the files.*

Exercise B.1 (3+5 marks)

(a) Consider the propositional planning task $\Pi = \langle V, I, O, \gamma \rangle$ with

$$
\begin{aligned}
V &= \{a, b, c, d, e\} \\
I(a) &= \mathbf{T} \\
I(v) &= \mathbf{F} \quad \text{for all } v \in V \setminus \{a\} \\
O &= \{o_1, o_2, o_3, o_4\} \\
\gamma &= e
\end{aligned}
$$

and

$$
\begin{aligned}
o_1 &= \langle \top, b \wedge d \rangle \\
o_2 &= \langle \neg e, a \wedge \neg b \rangle \\
o_3 &= \langle c, (d \rhd e) \rangle \\
o_4 &= \langle a \wedge \neg b, c \wedge \neg a \rangle
\end{aligned}
$$

Plot the search space explored by a progression and by a regression breadth-first search through this task. In the regression search simplify the state formula as much as possible at every node of the search tree. Do not expand the node further if that formula is unsatisfiable or logically entails the state formula of a previously expanded node. In the progression search do not expand a node if its state is a duplicate of a previously expanded state.

(b) Look up the following 5 planners in planner abstracts of the International Planning Competition (IPC) 2008, 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, SAT or symbolic), the design choices of their respective class (for example the search direction for explicit search) and other aspects that stand out.

1. OLCFF
2. MAPlan
3. HSP^*_0
4. Madagascar
5. Symple

You can find planner abstracts on the competition websites reachable from the ICAPS website (`http://icaps-conference.org/index.php/Main/Competitions`*).*

Exercise B.2 (5+2+3+3+3 marks)

In this exercise we consider the solitaire game Beleaguered Castle (`http://justsolitaire.com/Beleaguered_Castle_Solitaire/`). It consists of a deck of cards stacked face-up in several tableau piles. For each suit in the deck there is a discard pile consisting only of the ace initially. There are three types of legal moves:

- The top card of a tableau pile can be moved on top of another tableau pile if the top card of the target pile has a value that is one higher. The suit of both cards does not matter for this move. For example, 2♣ can be moved on 3♡, 10♣ on J♠, or Q♢ on K♢.
- The top card of a tableau pile can be moved to an empty tableau pile. This is allowed for all cards (not just for kings as in other solitaire games).
- The top card of a tableau pile can be moved to the discard pile for the matching suit if the top card on the discard pile has a value one lower. For example, if 7♡ was discarded last, then 8♡ can be discarded next. Discarded cards can never be moved again.

The objective of the game is to move all cards to their corresponding discard pile. We consider a generalization of the game with m tableau piles $\mathit{Tableaus} = \{t_1, \ldots, t_m\}$ and any set of cards *Cards*. For a given card $c \in \mathit{Cards}$ we use $\mathit{suit}(c)$ and $\mathit{value}(c)$ to refer to its suit and numerical value. The set of discard piles contains one discard pile for each suit: $\mathit{Discards} = \{\mathit{discard}_s \mid s = \mathit{suit}(c) \text{ for some } c \in \mathit{Cards}\}$. The set of all piles is $\mathit{Piles} = \mathit{Tableaus} \cup \mathit{Discards}$.

(a) In the file `regression/strips_regression.py` you will find a partial implementation of a breadth-first regression search for STRIPS tasks. Complete the missing parts. Only regress a formula through an operator if that operator adds at least one proposition of the formula. Ignore search states with an unsatisfiable formula or a formula that is equivalent to the formula of a previously expanded state. You don't have to ignore formulas that imply the formula of a previously expanded state but are not equivalent (i.e., represent strict subsets of states).

Test your implementation by solving the given instance of "Beleaguered Castle". You can find a grounded PDDL domain in STRIPS and a small instance in the directory `castle`.

How many states are generated and expanded? Have a look at some of the generated states and explain why so many states are expanded.

(b) Extend your code from exercise (a) with mutex-based pruning by completing the following steps:

- Complete the method `create_mutexes` by mapping each proposition to a set of propositions that are mutually exclusive with it. Normally, such mutex groups would be discovered automatically from the planning task but here you can manually add mutex groups for the specific instance. More specifically, use the mutex groups $L_c = \{c\text{-}on\text{-}x \mid x \in \mathit{Piles} \cup \mathit{Cards} \setminus \{c\}\}$ adapted to the instance.
- Before starting the search, call `create_mutexes` and store the result.
- Before inserting a node into the queue, loop over all propositions in the formula. For each proposition check if the set of propositions mutex with it intersects the formula. If it does, there are two mutually exclusive propositions in the formula and it does not have to be added to the queue.

Repeat the experiment from exercise (a) and discuss the differences.

(c) The file `regression/general_regression.py` contains a partial implementation of general regression of a formula through an effect. Complete the implementation. Regress the goal through each operator and list the resulting formulas. Simplify the formulas as much as possible (within your implementation or manually). If a formula violates mutexes, list at least one of the violated mutexes (without proof).

Test your implementation and use it on the task `vampire/p01_grounded.pddl`. Provide the sequence of operators and formulas that you obtain through regressing the goal of the task. Underline violated mutexes.

(d) 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.

(e) 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 (d).

Exercise B.3 (5+3 marks)

Pyperplan (`https://bitbucket.org/malte/pyperplan/src/default/`) is a lightweight STRIPS planner written in Python. While it doesn't come with as strong performance as Fast Downward, it is very easy to extend and modify.

(a) In the file `pyperplan/src/search/bdd_bfs.py` you can find an incomplete implementation of a BDD-based breadth-first search. Complete it by using the utility methods in the file `pyperplan/src/search/bdd.py`. Do not modify anything else than the file `pyperplan/src/search/bdd_bfs.py` (and do not modify the constructor of `BDDSearch` yet, this is for part (b)). Test your search on the tasks in the directory `blocks` and make sure that it can find valid plans.

You can run the code with the command
`./pyperplan/src/pyperplan.py -s bdd blocks/domain.pddl blocks/p1.pddl`

(b) The constructor of `BDDSearch` contains a commented out alternative variable order for the variables within the BDD. Change the order by commenting out the old order and including the new order instead. Print the number of total BDD nodes after adding each operator and after each expansion step (use the provided method `print_bdd_nodes()`). Compare the two variable orders on a small task and discuss the results.

Exercise B.4 (3+5 marks)

For this exercise, you need to have minisat (`minisat.se/MiniSat.html`) installed. The simplest option is to install the package pysat, which will also install several other SAT solvers. You can install everything you need by running the following command: `./install-pysat.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 one minute.

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 printed wallclock time can be quite off. Instead, please use the linux built-in `time` command by prepending it to the above command to obtain real CPU runtimes 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`.
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.