Business Process Optimization Competition '25

Overview

The goal of the project is to build efficient planning and scheduling mechanisms for an artificial hospital. To do this, the competitors implement:

1. A 'plan' policy function that assigns patients to the moment in time at which they will be admitted to the hospital for treatment. You can think of it as an appointment book planning problem.
2. A 'schedule' policy function that schedules how many resources of different types (e.g. operating rooms, hospital beds, ...) are available at which time. This is in essence a staffing problem.

The objective is to create these functions in such a way that they both minimize the cost of running the hospital and the cycle time it takes to treat the patients. For both cost and cycle time it holds that 'lower is better'. The tradeoff lies therein that having more resources available reduces the cycle time of patients (due to less waiting), but at the same time increases the operational costs due to resource wages.

Different types of patients are treated in the hospital. Patients will go through a treatment process that roughly follows the steps illustrated in Figure 1. The figure illustrates that, once admitted, patients have an intake interview. After that they go to surgery and then to a nursing ward. Complications of their surgery may arise, upon which they need to go back into surgery. However, note that the figure is incomplete. To understand in detail which types of patients there are, which process steps they go through, and which resources are needed in which step, you will need to do process mining on your own.

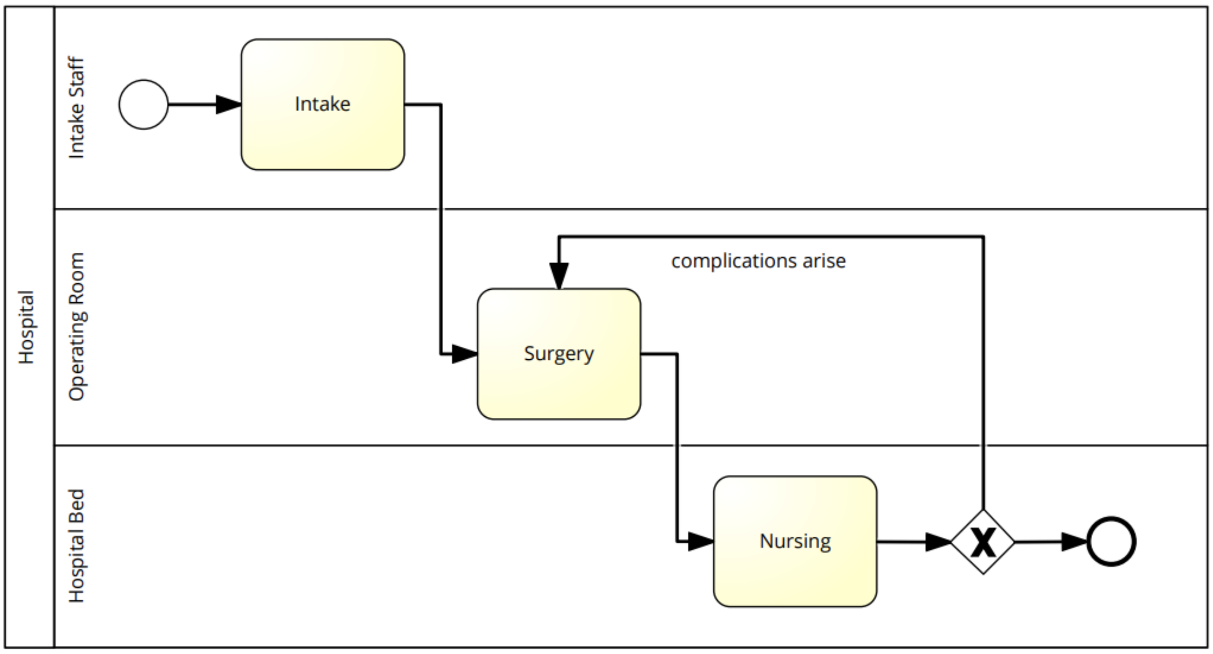

Figure 1. The treatment process.

The assignment

Students implement two policy functions in Python: ‘plan’ and ‘schedule’.

The quick way to get started is to use the example planner that is provided with the assignment code (__example__.py). The example runs out-of-the-box and contains example implementations of both policy functions that are relatively easy to understand and contain some useful pointers for how you can proceed.

The ‘plan’ policy function is invoked each time a resource or patient becomes available. It is given three parameters. The first is the list of patients who have not yet been planned for admission to the hospital. The second is the list of patients that have been planned, but can still be replanned, because they have not yet been admitted. The third is the current simulation time. You implement the function such that it returns a list of patients and the time at which to admit them. Not each patient has to be planned at the moment the ‘plan’ function is invoked. If a patient is not in your result list, it will simply be in the list of patients to plan again next time the function is invoked. Patients must be planned for a timeslot 24 hours before their admittance. This means that if you try to plan a patient for admittance at a moment in time that is less than 24 hours ahead, you will receive an error. For example, if the plan policy function is invoked with a list of patient identifiers [100, 101, 102] and a time t, it could return a list of tuples [(100, t+24), (101, t+168)], meaning that patient 100 will be admitted tomorrow at the same time, patient 101 will be admitted next week at the same time, and the planning of patient 102 is postponed till the next time the plan function is invoked. Returning [(100, t+23)] would raise an exception, because then patient 100 is planned less than 24 hours ahead.

The ‘schedule’ policy function is invoked each day at 18:00 (in simulation time) and can be used to schedule how many resources of which type should be available at which (future) moment in time. The function must return a list of tuples (<type>, <time>, <number>). Each tuple should contain:

- A resource type.
- The number of resources of that type that should be available.
- The moment in (simulation) time from which that number of resources should be available.

Note that a week has 168 hours and the simulation starts at t=0, which is Monday 2018-01-01 00:00. There is a maximum number of resources of each type that is available. For example, there are at most 5 operating rooms available. Your schedule must observe the following constraints:

1. You must schedule for tomorrow 8:00 or later (i.e. at least 14 hours ahead, considering that scheduling is done each day at 18:00).
2. You must not schedule more resources than the maximum number of resources of a type.
3. If you are scheduling less than one week ahead on the same day (i.e., less than 158 hours ahead), you can only increase - not decrease - the number of resources.

For example, if your schedule function is invoked at time t=18, you could return [("OR", t+14, 5), ("OR", t+158, 3)]. Note that t=18 is 18:00 on Monday 1 January 2018. This is the first moment in time at which the schedule function will be invoked. The returned answer means that from t+14, i.e. from 8:00 on Tuesday 2 January, 5 operating rooms will be available. This will continue to be the case until t+158, i.e. 8:00 on Monday 8 January, from which moment only 3 operating rooms will be available. This number of operating rooms will continue to be available indefinitely, but at the next invocation of the schedule function you can change that.

The next invocation of the planning function is at time t=42, or Tuesday 2 January 2018 at 18:00. Suppose that at this time we return [("OR", t+2, 5), ("OR", t+14, 6), ("OR", t+38, 3)]. This will return errors for multiple reasons:

1. We are scheduling for t+2, which is not allowed because it is less than 14 hours ahead.
2. We are scheduling 6 operating rooms for t+14, which is not allowed, because the maximum number of operating rooms available is 5.
3. We are scheduling 3 resources for t+38, i.e. on Thursday 4 January at 8:00, but note that this is less than a week ahead, meaning we cannot decrease the number of resources to 3, from the 5 that we originally planned to be available on Thursday when we planned ("OR", t+14, 5) during the scheduling function invocation at t=18.

Evaluation

Each solution is scored according to 4 KPIs:

- Waiting time for admission (WTA)
- Waiting time in hospital (WTH)
- Nervousness (NERV)
- Personnel cost (COST)

The waiting time for admission is the total time patients have to wait from the moment they arrive in the process until the moment they start their intake starts. Note that patients who never receive their intake will still add to this number.

The waiting time in the hospital is the sum of waiting times for each task. This does not include the waiting times of the emergency room patients, since you have no control over them.

Nervousness scores replannings, because replannings are undesirable to a patient. Each time replanning takes place, the score is increased by how much less than 14 days before the original planning point the patient is replanned (because replanning on short notice is undesirable). Note that patients which have not been taken in by the end of the simulation lead to a penalty. More precisely, the nervousness is calculated as max((168*2-(t_p-t_r)),0) where t_r is the time at which the replanning takes place and t_p is the originally planned time of the intake (i.e., the time for which the patient was scheduled before replanning took place). Therefore, the penalty for replanning is 0 if the replanning takes place more than 2 weeks (168*2) in advance and increases the closer the (originally planned) intake would have taken place.

The personnel cost is the total cost of personnel. Personnel costs 1 per hour you scheduled them to work. However, if they are planned less than one week ahead, they will cost 2, and overtime (a person still needs to finish a task when actually scheduled to go home) costs 3. Note that overtime cannot be completely avoided: it is common that personnel still needs to finish a task (e.g., a surgery) when they actually want to go home.

To compare submissions for the assignment, each KPI will be normalized, by calculating it as the percentage improvement over the benchmark, using the formula 100% * (<your_value> - <benchmark_value>)/<benchmark_value>. KPIs will then be weighed, such that personnel cost counts 3x where the other KPIs count 1 time, because employing extra personnel can be used to improve all of the other KPIs. So, the formula for computing the final score is:

(normalized(WTA) + normalized(WTH) + normalized(NERV) + 3*normalized(COST))/6

Technical Details

You can run the simulator using:

```
problem = HealthcareProblem()
simulator = Simulator(your_planner, problem)
result = simulator.run(365*24)
```

In this code `your_planner` is an instance of a class that you implement yourself. This class must subclass `planners.Planner` and implement the following function.

```
def plan(self, cases_to_plan, cases_to_replan, simulation_time):
```

> Takes a list of case identifiers of patients that can be planned, a list of case identifiers of patients that have already been planned, but can still be replanned, and the current simulation time. Returns a list of tuples (<case identifier>, <time>), representing the moment in time at which the case the given case identifier will be admitted to the hospital

```
def schedule(self, simulation_time):
```

> Takes the current simulation time. Returns a list of triples (<type>, <time>, <number>) indicating how many resources of which type should be available from which moment in time.

```
def report(self, case_id, element, timestamp, resource, lifecycle_state, data=None):
```

> A function that you can use to collect information from the simulator. It is called each time one of the following happens: a new case arrives, a task becomes ready to perform, performing a task started, performing a task completed, a case completes. The simulator does not expect any return value. This function is just there for you to collect data on what is happening in the process, so you can also have it do nothing.
>
> There are three ready-made reporters that you can use for your convenience in the reporter.py library. One simply prints each event to the standard output, one exports them as an event log that can be analyzed using your favorite process mining tool, and one aggregated information on resource usage which be visualized in the form of a graph after the simulation ends.

An example planner is implemented in the __example__.py file for your reference.

Important Notes

It is explicitly forbidden to use any variables that you did not create yourself, based on information that you obtained via the plan, schedule, or report functions. Doing so will lead to a desk rejection of the submission.

Your code must simulate one year (365*24 hours) of cases within 2 hours on a typical desktop computer. Code that takes longer to run will be rejected.

By default, the simulator always simulates a year (365*24 hours) of cases. However, this may take quite long, which may be prohibitive when you are coding or debugging. For that reason, you can also pass the number of hours you want to simulate to the run method.

Submission

Submit your assignment in Moodle by August 31, 2025. If you'd like to participate in the annual BPO competition (which is this assignment) submit your solution by 15 August 2025 by e-mail to: bpo.competition@bpm.cit.tum.de

For any submission, send in your submission as a single .zip file that includes:

1. Your code in Python. This can contain multiple files. Make sure there is one file that ends with:

   ```
   problem = HealthcareProblem()
   simulator = Simulator(your_planner, problem)
   result = simulator.run(365*24)
   ```

   This is the file from which we will call your code to evaluate your solution.
2. A .pdf file with a brief description (max. 1 A4) of the main technique or techniques that you used for planning.
3. A .txt file with a group name (optional), person names, affiliations (optional), and e-mail addresses of the people who make the submission.

Evaluation process

Your submitted solution will be checked and graded. If you compete, the winner will be announced at the BPM 2025 conference and the ranking will be updated after that.