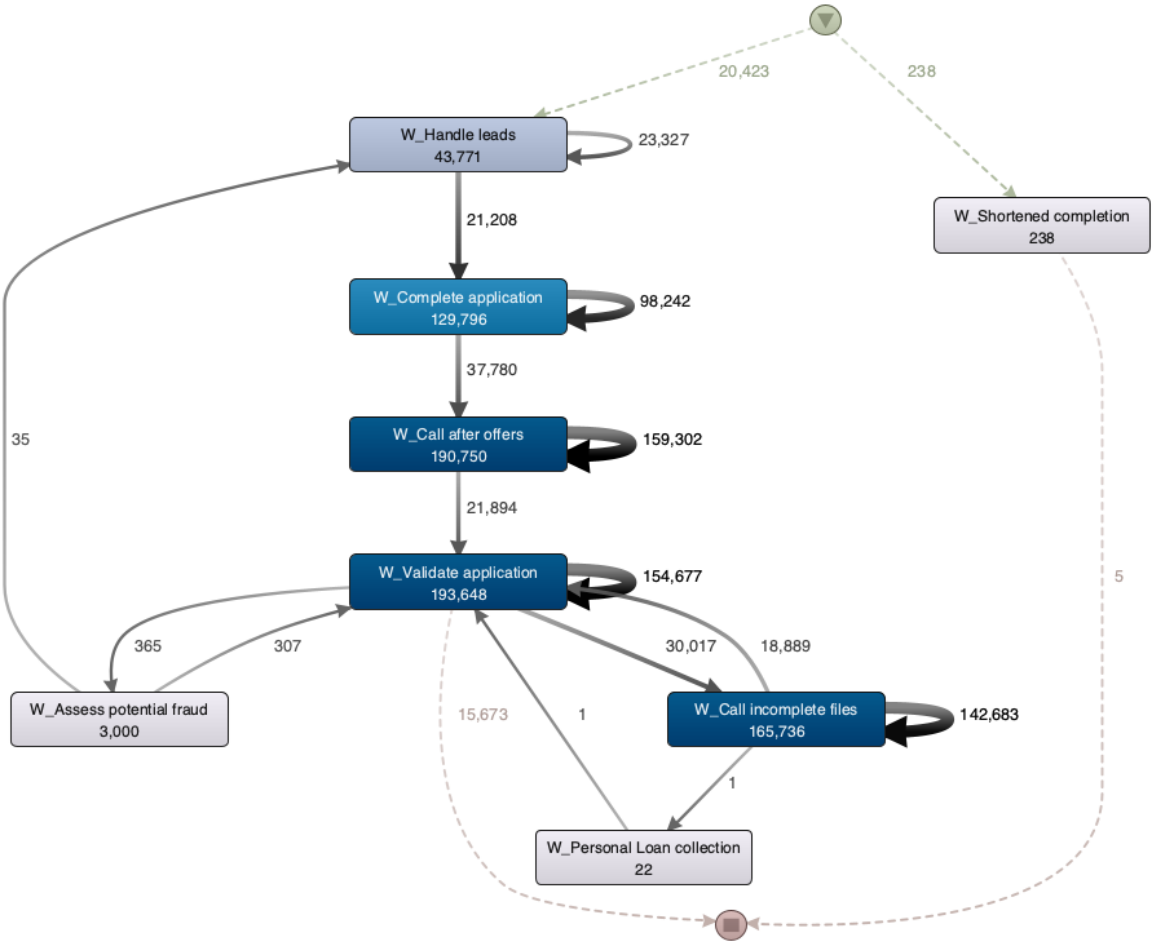
Business Process Optimization Competition

The goal of this competition is to develop a technique that assigns people to tasks in such a way that the total cycle time of the cases is minimized. The technique can be applied to a simulated company, which is based on the financial services institute from the BPI 2017 challenge (van Dongen, 2017). The group that achieves the lowest average cycle time per case on this simulator wins the competition.

A model of the process, mined from the simulator using Disco[1], is shown below.



The figure illustrates how customer cases passed through the process in a particular simulation run. Note that it only shows the most frequent paths to keep the figure readable.

The figure illustrates that - for this simulation run - after a case arrived, for the vast majority of the cases (20,423), first a task of the type 'W_handle leads' became active and had to be performed. When a task becomes active for a customer case, a resource must be assigned to perform that task. When the resource completes the task, the next task becomes active. For this simulation run in 23,327 cases a task of the same type became active after performing a 'W_handle leads' task and had to be performed. In 21,208 cases, a task of the type 'W_Complete application' became active. A case ends when no tasks become active for it anymore. The figure illustrates that - for this simulation run - in most cases this is after some task of the type 'W_validate application' was performed.

[1] https://fluxicon.com/disco/

Tasks must be executed by resources that are shared between all cases and all tasks in the process. A resource can only work on a single task at a time. Task types have a 'resource pool'. This is the pool of resources that are authorized to perform tasks of that type. A resource cannot perform a task for which it is not in the resource pool. As a consequence, a task (and the case that the task belongs to) may incur a waiting time, because it may have to wait for a resource to become available that can perform it.

Resources work a number of hours per day. When they arrive they will stay for a couple of hours. Once a task is assigned to them, they will complete that task, but they may leave after performing a task. Different resources may be more or less effective on tasks of different types in terms of the time they need to perform the task.

The cycle time of a case is the time between the moment at which the case arrives and the moment at which it completes. These moments are the same as the moments at which the first task of the case becomes active and the moment at which the last task of the case completes. Consequently, the cycle time includes both the time resources take to execute tasks for the case and the time the case has to wait for a resource to become available to perform a task.

Technical details

You can run the simulator using:

```
simulator = Simulator(your_planner)
result = simulator.run()
```

In this code your_planner is an instance of a class that you implement yourself and that must implement the following two functions:

```
def plan(available_resources, unassigned_tasks, resource_pool):
```

A function that you can use to assign resources to tasks according to a plan that you can make yourself. The function is called each time a task becomes available to execute or a resource becomes available. Your function must return a list of pairs (resource, task) where resource is a member of available_resources and task is a member of unassigned_tasks. The resource will then immediately start executing the task. Assigning a task that is already assigned or an unavailable resource (i.e. tasks or resources that are not in the provided lists) leads to an error. The function also has a parameter resource_pool, which is a dictionary that assigns task types to the list of resources that can perform it. A task must be assigned to a resource from the pool of resources that can perform it. If it is assigned to a resource that cannot perform it, an error will be returned. See the API reference below on how to extract task types from tasks.

```
def report(event):
```

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. More information on the event data structure is provided in the API reference below.

The result returned by the simulate function is a pair (cycle_time, message), where cycle_time is the average cycle time of the executed cases or None, if there was an error in the simulation of the problem. message is free text that explains what happened in natural language. Note that the cycle time of unfinished cases also adds to the average. The cycle time of an unfinished case is calculated from the moment the unfinished case starts till the moment the simulator ends. Also, note that a case is considered to start at the moment it arrives and its first task is activated. Depending on how you do process mining on the simulator, this may be different from how your process mining tool reports the cycle time.

Notes

The company uses a lot of temp workers and at each time of the day only a limited number of resources is available. Resources will always complete their task when it is assigned to them, but may leave in between executing tasks. Due to the use of temp workers, it is uncertain when the same resource will come back again.

Resources have different efficiency in terms of the time they take to perform a task of a particular type.

By default, the simulator always executes the same dataset. However, two datasets are provided to you. You can test your approach on the other dataset as well, by passing it as an argument to the simulator (see API Reference below).

The competition will be decided based on yet other datasets, also using multiple replications of the simulation to determine if the difference in performance between solutions is significant.

IMPORTANT. It is explicitly forbidden to use information in your code from the .pickle files that are provided or from properties or methods from the simulator.py file, other than the ones specified in the API Reference below. Doing so will lead to a desk-reject of the submission.

Submission

Submit your solution by 12 August 2022 by e-mail to:
r.m.dijkman@tue.nl

Send in your submission as a single .zip file that includes:

- Your code in Python. This can contain multiple files. Make sure there is one file that ends with:
  ```
  simulator = Simulator(your_planner)
  result = simulator.run()
  ```
  This is the file from which we will call your code to evaluate your solution.
- A .pdf file with a brief description (max. 1 A4) of the main technique or techniques that you used for planning.
- A .txt file with the names, affiliations, and e-mail addresses of the people who make the submission.

Evaluation process

Your solution will be checked between 12 August and 12 September 2022. On 12 September 2022 the winner of the competition will be announced at the BPO workshop (https://sites.google.com/view/bpo2022) at Castle Münster, Germany.

Prize

A prize of 1,000 euro is awarded for the best solution. If there are multiple submissions with the same result, the prize is shared between those submissions. The best three submissions receive a certificate with their ranking. All submissions receive a certificate of participation.

API Reference

```
class Event:
  # An instance of class Event has the following attributes:
  # The unique identifier of the case
  # to which this event applies.
  int case_id
  # An instance of the class Task to which this event applies.
  # None if the event does not apply to a task.
  Task task
  # The simulation time (in hours since the start of the simulation)
  # at which the event occurred.
  float timestamp
  # The resource to which this event applies.
  # None if the event does not apply to a resource.
  str resource
  # The type of event represented.
  # Can be one of EventType.
  EventType lifecycle_state

class EventType(Enum):
  # The event represents the arrival of a new case.
  CASE_ARRIVAL
  # The event represents the moment at which a task becomes ready
  # to be executed by a resource.
  TASK_ACTIVATE
  # The event represents the moment at which executing a task
  # by a resource starts.
  START_TASK
  # The event represents the moment at which executing a task
  # by a resource completes.
  COMPLETE_TASK
  # The event represents the moment at which the case completes,
  # i.e. the moment at which the last task of the case completes.
  COMPLETE_CASE

class Task:
  # An instance of class Task has the following attributes:
```

```
  # The unique identifier of the task.
  int id
  # The unique identifier of the case to which the task belongs.
  int case_id
  # The name of the type of the task.
  # This is one of the names in the figure above:
  # W_Handle leads, W_Complete application, ...
  str task_type

class Simulator:
  # The simulator you can use to evaluate your solution.
  # Only use the following methods:
  # Instantiate the simulator with your own planner.
  # planner: an instance of your planner that implements
  #   the two methods ‘plan’ and ‘report’ as specified above.
  # str instance_file: optionally, the name of the problem instance
  #   you want to simulate.
  #   Defaults to: "BPI Challenge 2017 - instance.pickle"
  #   Can also be set to: "BPI Challenge 2017 - instance 2.pickle"
  def __init__(self, planner, instance_file)
  # Can be invoked to run the simulator instance on your planner.
  # When ready returns a pair (cycle_time, message).
  # float cycle_time: the average cycle time of simulated cases.
  #   None if an error occurred.
  # str message: a message describing in natural language what
  #   happened.
  def run(self)
```

References

van Dongen, Boudewijn (2017): BPI Challenge 2017. 4TU.ResearchData. Dataset. https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b