# Evolutionary process discovery

**Dr. Laura Genga (TU/e) – Dr. Marco S. Nobile (TU/e) – Ya Song (TU/e)**

## Processes and Petri Nets

The financial institute PETRINAS manages loan requests from its customers according to a not well-formalized process model. The process starts when a request is received. Then, the request passes preliminary assessments, aimed at verifying whether the applicant meets the requirements. The request also undergoes frauds detection. As soon as an application is not eligible, the process is terminated. The user receives an offer and can communicate whether he/she intends to accept it. If this is not the case, the process ends. Otherwise, a contract is drawn and finally sent to the customer. The actual sequence of these actions is unknown to the institute.

A sequence of actions can be formalized and investigated using a Petri Net (PN). A PN is a directed bipartite graph characterized by two sets of nodes: the **places** (used to represent the status of the process) and the **transitions** (corresponding to the activities described above) connecting the places. Places can only be connected to transitions, and transitions can only be connected to places.

PNs can be effective to represent and study processes. For example, the PN shown in Fig. 1 represents one possible action: receiving the request, a transition which brings from place p1 to place p2.

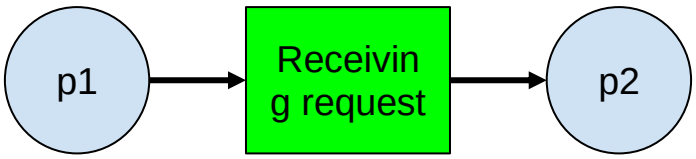


**Fig. 1** - Example of Petri net with two places and one transition.

Given a set of $M$ places and a set of $N$ transitions, we can represent the PN's connections using a matrix $A$ of dimension $M \times N$.

The $m$-th row in the matrix denotes the $m$-th place. The $n$-th column represents the $n$-th transition. Then, the element $a_{m,n}$ of the matrix encodes the connection between the $m$-th place and the $n$-th transition.

- if $a_{m,n} = -1$ then an arc connects the $m$-th place to the $n$-th transition;
- if $a_{m,n} = 1$ then an arc connects the the $n$-th transition to the $m$-th place.

As an example, consider the PN in Fig. 2.

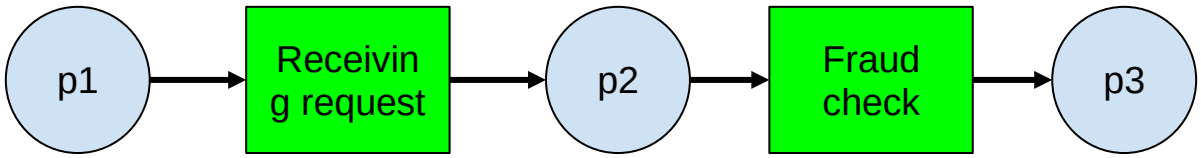


**Fig. 2** - Example of Petri net with three places and two transitions.

The PN in Fig. 2 can be represented by the following $3 \times 2$ matrix:

```
-1   0
 1  -1
 0   1
```

Given a PN encoded as a matrix, we can perform a simulation of a process by starting from an arbitrary place (we will use p1 as default) and "follow the arcs". If we keep track of the transitions (i.e., activities) followed during the simulation we get a simulated **trace**. For instance, the PN in Fig. 2 can produce the following trace:

*"Receiving request", "Fraud check"*

The company PETRINAS has collected a dataset composed of 100 traces of this kind from their processes (contained in the file `dataset_a2.txt` that you can find in Canvas) and is now interested in creating and investigating the associated PN. The company knows that there are 10 clear activities + 2 "invisible activities" in their process, which corresponds to an unknown matrix with dimension $12 \times 12$ to be discovered. Considering that each element of the matrix can take values {-1, 0, 1} then the number of possible PNs to be tested is $3^{144} \approx 5 \times 10^{68}$which is clearly not feasible by means of an exhaustive search.

## Genetic Algorithms to the rescue

We can explore this huge search space using a **Genetic Algorithm** (GA). The candidate solutions (i.e., the PN matrices) can be encoded as **linearized lists of integer numbers**. For instance, the PN in Fig. 2 can be encoded as: [-1, 0, 1, -1, 0, 1].

A very simple fitness function for a candidate solution can be based on the **simulation** of the experimental traces. Specifically, given a candidate PN whose fitness value must be assessed, we can iterate on each trace of the dataset and see whether the PN can actually simulate that trace correctly. For each correct step taken the PN in a trace, we accumulate +1 in a counter. As soon as the simulation is blocked (because we cannot take the next expected transition from the place that we reached in the PN), the simulation of that trace terminates and we start over with the following trace. When no more traces can be simulated, the counter is returned as the fitness value for that PN.
The pseudocode of the aforementioned fitness function is shown below:

```
fun fitness(A):
   counter = 0
   for trace in dataset:
      current_place = p1
      for activity in trace:
         if A[current_place][activity] == -1:
            counter += 1
            current_place = get_new_place(A, activity)
         else:
            break
   return counter
```

If we try to optimize this fitness function, we find out that the GA tend to converge to a complex PN with several arcs. In order to mitigate this phenomenon, we can introduce a penalty factor in the formula. For instance, we can penalize the number arcs in the PN, in order to promote smaller solutions (see, e.g., De Medeiros *et al*., "Using Genetic Algorithms to Mine Process Models"). To do so, we can change the last line of the function as follows:

```
return w*counter-(1-w)*count_nonzero(A)
```

where `count_nonzero()` is a function that returns the total number of elements in A that are not equal to 0. By using this approach, PNs with a lot of arcs get penalized. In the equation, the contribution of the correct traces and the regularization terms is balanced with a real parameter `w` in [0,1]. The greater the value, the higher the contribution of the traces to the final fitness value. You can find this **fitness function and the simulation code already implemented** in the `support.py` file on Canvas (in this assignment, we set `w=0.9`).

For the assignment, you have to perform the following tasks. The first task is to deal with this problem with GA:

1. (5 points) implement the GA using deap.algorithms.eaSimple method in DEAP. Please use ['cxTwoPoint', 'mutFlipBit', 'selTournament'] as the [Crossover, Mutation, Selection] operators, record the default parameter of your initial GA and plot the best fitness at each iteration ;

Then you have to **investigate the performance** of the GA: in task 2 and 3 that you will do, check the **distribution of the fitness values** of the best solution found across 30 runs by creating **boxplots**. Also, check the **average best fitness**(ABF) at each iteration (over 30 runs) to verify possible premature convergence by creating line plots. Please check the examples of these two figures on the lecture 2 slides.

2. (5 points) **compare the performance of possible variants of your GA**, e.g., using different crossover, mutation, and selection operators. Test the variants in the table below, and investigate both the impact to the **optimization performances** and the

overall **running time**, keep the best choice according to ABF for the later questions**.** You can refer to the operators described in DEAP documentation: Operators;

| Crossover | Mutation | Selection | ABF | Time (Seconds) |
|---|---|---|---|---|
| cxTwoPoint | mutFlipBit | selTournament | | |
| cxTwoPoint | mutFlipBit | selRoulette | | |
| cxTwoPoint | mutShuffleIndexes | selTournament | | |
| cxTwoPoint | mutShuffleIndexes | selRoulette | | |
| cxOrdered | mutFlipBit | selTournament | | |
| cxOrdered | mutFlipBit | selRoulette | | |
| cxOrdered | mutShuffleIndexes | selTournament | | |
| cxOrdered | mutShuffleIndexes | selRoulette | | |

3. (5 points) investigate the **impact of mutation and crossover probabilities** and try to determine the **best choice of hyper-parameters** for this problem, keep the best choice for the later questions. You are suggested to test mutation and crossover probabilities in the table below;

| Average Best Fitness | | crossover probabilities | | | |
|---|---|---|---|---|---|
| | | 0.2 | 0.4 | 0.6 | 0.8 |
| mutation probabilities | 0.2 | | | | |
| | 0.4 | | | | |
| | 0.6 | | | | |
| | 0.8 | | | | |

4. (3 points) the representation that we are using gives the possibility to connect multiple places to the same transition, which might be undesirable for PETRINAS. In this task, you are suggested to exploit tool decoration to handle such type of **constraint** and "fix" the candidate solution after a mutation and/or crossover to have **at most 2 non-zero entries for each column**;
5. (2 points) use the function `matrix_to_graphviz()` provided in the file `support.py` to **plot the best Petri net** both before and after using tool decorator (best with respect to the fitness function) and **briefly comment it** for PETRINAS. The function receives as argument a candidate solution, in the form of a linearized matrix. The function creates a file `output.pdf` containing the plot. In order to use the `matrix_to_graphviz()` you have to **install the required libraries**: pm4py and graphviz (see the instructions in the appendix of this file);

**Summarize all your findings into a PDF report**:

- explain **which tests** you have performed;
- show the **figures** and motivate which was the **best configuration** of the GA according to your tests.

Remember: submit both the **report** and the **source** Python code. If one of the two files is missing, the score will be 0 points.

The max length of the report is: **6 pages**.

If you have any comments, questions, or doubts about the assignment, please **open a thread on Canvas Discussions**: other students may be looking for the same answer. Also, feel free to answer your colleagues. **We will not answer private emails.**

Finally, please remember that this is just an exercise about GAs: it is not important if the PNs do not look completely reasonable as long as they are optimal with respect to the fitness function.

## Appendix A: how to install pm4py and graphviz

To install **pm4py** simply use pip:

```
pip install pm4py
```

To install **graphviz** please follow the instructions on the website:

https://graphviz.org/download/

There are executable installation files for the main operating systems. In order to make graphviz available to pm4py, **please make sure that graphviz's executable files are in your system's path**. The installer should ask you about this: please answer yes. You may need to restart your laptop after the installation of graphviz.

## Appendix B: Assignment Rubric

| | |
|---|---|
| Q1 (5 points) | • The code of the initial GA is functional and well-documented;<br>• The plot of the best fitness at each iteration is provided. |
| Q2 (5 points) | • The code for testing different operators is functional and well-documented;<br>• The table is filled out properly;<br>• The boxplot used to show the distribution of the best fitness values is provided;<br>• The line plot of ABF at each iteration is provided. |
| Q3 (5 points) | • The code for hyper-parameters tuning is functional and well-documented;<br>• The table is filled out properly;<br>• The boxplot used to show the distribution of the best fitness values is provided;<br>• The line plot of ABF at each iteration is provided. |
| Q4 (3 points) | • The code for tool decoration is functional and well-documented; |
| Q5 (2 points) | • The plots for the best Petri nets are provided;<br>• The report describes the findings about the best Petri nets and makes some comments. |