# DAT 530
# Optimizing Task Assignment in a Consultant Company

By Daniel Fylling, Jing Hou

Nov 10th, 2023

# Contents

# 1 Abstract

In this project, we aim to develop a Petri Net model to simulate the process of task assignments in a consultant company and provide indications of the human resources optimization in the process. We will illustrate our model's development from the basic version to a highly refined version designed for the optimal allocation of job tasks to the most suitable employee within the consulting firm.

By analysing the results of the simulation we can offer valuable insights to support decision-making, like determining which employee must be let go during a downsizing stage.

It is believed that the model can be reshaped to fit other real-life scenarios as well, such as to simulate vessel utilization in the offshore industry, e-bike usage in the city or printer availability in the printing business.

# 2 Introduction

This project work is done and presented for the course Discrete Simulation and Performance Analysis (DAT530) at the University of Stavanger. This project aims to investigate the consulting company and simulate the job task assignments which involves allocated job task, human resource, possible candidate profile, and assigned job task requirements in a real-life consultant company. In this report, the basic Petri Net theory and GPenSIM in detail are not discussed, rather we will focus on our model implementation and simulation analysis. The MATLAB implementation code of the four models will be included in a .zip file along with the report. First of all, we will present our motivation for this topic selection, discuss the design of the Petri Net model that has been applied, and provide the analysis of the simulation results.

## 2.1 Problem Definition


The consulting industry is highly sought-after and shows continual growth across various sectors, whether they are small or large. Consulting firms offer professional training, advice, product delivery, and technical support directly to their clients. However, there are several challenges existing[1].

Challenges such as developing new clients, keeping the right resources, gaining project efficiency, innovating new ideas to stay competitive, keeping up with shifting client demands and staying profitable, etc [2]

Given that common sense, it is vital to assess one of the main challenges - project efficiency, as the consultants are valuable yet costly assets for consulting firms[3], A crucial factor lies in the efficient management and evaluation of their work performance, which can significantly contribute to the success of the consulting firms[4]. Therefore, we decided to investigate the consultant's operational efficiency and give insights on how to maximize human resource utilization in the consulting business.

In a consulting firm where the scenario is job tasks are rapidly assigned and there is a substantial workforce on hand, who is the best candidate for this specific task and the most cost-saving selection at the same time? It will not be very straightforward to discover instantly. Therefore, we designed a Petri Net model to investigate the optimized solution for identifying the most desirable and prioritized candidate in the consulting company employee roster. It is very interesting to see who wins the task with the same required qualifications scenario. Who will be prioritized in the employee roster? If there are changes in business demand, which employees should we aim to keep?

This analysis will be shown in our model simulation. The final version of the model offers insights into optimal candidate selection by considering job task prerequisites, employees' skill proficiency, and availability altogether, streamlining the process.

## 2.2 Adaptation

This work is inspired by the showcase specifically from lecture materials.

- Discrete Flow Model-I: Airport
- Project Management and Scheduling
- Introducing Resources (Addendum-2: Using GPenSIM Resources)

Our choice of this topic comes from that we have viewed the pool of previous project reports and we noticed that no groups had explored this intriguing subject. This led us to believe that focusing on a consulting firm could provide valuable insights into project efficiency and even support some challenging decision-making processes.

# 3 Method and Design

## 3.1 Overall Design

The primary objective of this project is to create a system capable of efficiently matching work tasks with suitable employees. We assume that small tasks are continuously generated and added to a task pool while a roster of available employees is at hand. The program's role is to assign tasks based on priority and specific task requirements. In more advanced versions, the system should be capable of optimizing the scheduling of all available tasks.

To visualize the model's flow, we think of it as a consultant company:

- Tasks originate from an external source and are deposited into the company's 'Inbox.'
- Company administrators determine which employee is the best fit for each task.
- Employees await task assignments in a break room.
- The task and the selected employee are sent to a 'Workstation' where the employee works on the task until it's completed.
- Upon completion, the task is placed in the 'Outbox,' and the employee returns to the break room.
- Throughout this process, relevant information regarding the tasks and employees is recorded for further analysis.

Bird view design of the model is shown Figure 1.

It's important to note that while this project focuses on employee-task matching, the system's architecture can be adapted for other scenarios, such as queuing systems for printers or other machines, showcasing its flexibility.

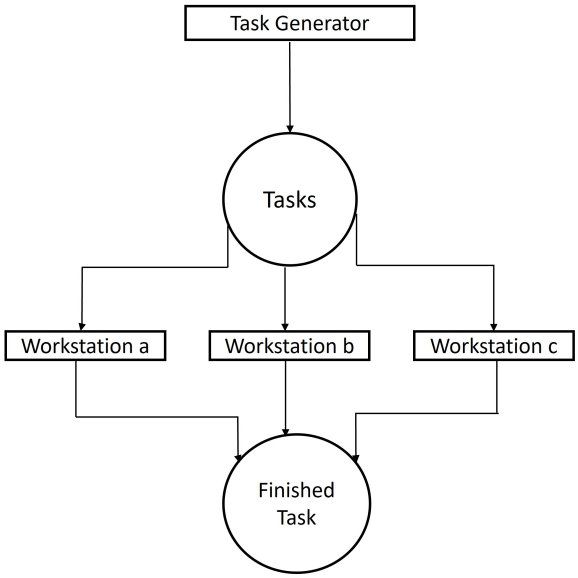


Figure 1: Graphical Model Design

## 3.2 Techniques

Here is a brief description of the key MATLAB and GPenSIM features that have been combined to produce the model:

- Assigning properties to tasks and employees has been done with GPenSIM coloring, more specifically index coloring, where the assigned color is a pointer to a library containing (and recording) information.
- GPenSIM resources is used to track employee utilization and for easily plotting results.
- Classes and function are used to be able to re-run code rather than duplicating it.

The coding approach is quite naive, largely due to the inexperience of the development team. The only advantage to this is that the code should be relatively easy to read, as the authors have commented it harshly for their own understanding.

# 4 Implementation

It is thought that a suitable way to introduce the final product is to present the journey of development from the very beginning. In this way the complexity of the code will gradually crystallize around the initial grain.

## 4.1 Version 1

In this version, a task generator 't1' and a 'front desk' record the time when tasks are received. The Petri net is shown in Figure 2.

1. Tasks are defined and their properties are stored in global memory.
    - Task properties include: Client, Size, and TimeStart.
2. Tasks are represented as tokens with index coloring in 'p2'.
3. Transition 't2' picks up tokens from 'p2' and records the received time.
4. The finished task is deposited in the 'p3'.

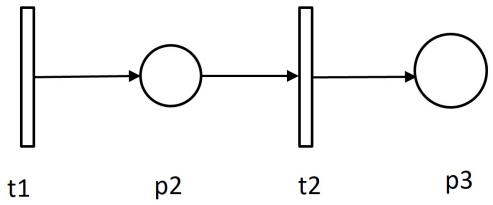


Figure 2: Petri net for Version 1.

## 4.2 Version 2

This version includes a single task and a single employee, aiming to make the employee complete the task. The Petri net is shown in Figure 3.

1. Employees are introduced as specific resources.
    - Employee properties are initiated and stored similarly to task properties.
    - Employee properties include: Speed and Queue.
2. The employee is assigned to the task.
    - The choice of employee is hard-coded because there is only one to choose from.
3. The time for the employee to complete the task is calculated.
    - This calculation is based on the task size and the employee's speed.
4. The employee is retained for the calculated time.

- This is achieved by manipulating the priorities of transitions 'tw1' and 't3'.

5. The employee is released when the task is finished.
6. The finished task is deposited in the outbox, 'p4'.

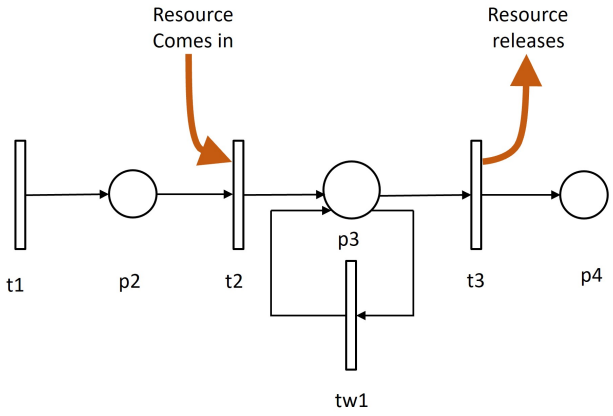


Figure 3: Petri net for Version 2.

### 4.3 Version 3

In this version, the number of both tasks and employees is increased to two. Also, tasks and employees have been given new properties that will be used to select the most appropriate employee to perform a certain task and how long it will take to perform. The Petri net is shown in Figure 4

1. A task may now require a certain certificate from the employee to be allowed to work on it. This means that we need to filter our full roster to a selection of possible candidates for the specific task at hand.
   - Three different certifications are supported.
2. Task completion time is now calculated based on task size, as before, but the employee work speed is calculated based on the combination of their competency profile and the composition of the task.
3. The task will be assigned to the employee that has the correct certification and will complete the task in the shortest amount of time. Here the work queue of the employee is taken into consideration, such that a slower employee will be chosen if the faster employee has already been assigned a huge workload. This also means that the function may choose to keep a slow employee idle if the faster employee can finish its current queue and the new task in less time than the slow employee would spend on only completing the new task.
4. The current model only allows a single employee to work at a time (this will be addressed in Version 4). Feeding two tasks into the same 'workstation'

will cause a mess, so a property is added to ensure that only one task is active in the workstation at a time. This is the job of the 'availability token' that is initialized in 'p6', returning only after the current task has been completed.

5. Due to the nature of color inheritance in GPenSIM [5], the transition 't4' was put in place with the sole purpose of clearing the color off the 'availability token' to prevent the color of the finished task from being added to the next task.

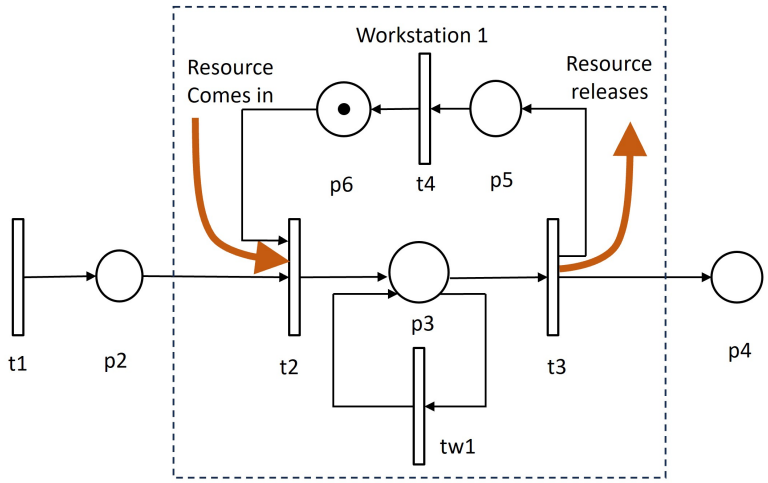


Figure 4: Petri net for Version 3.

## 4.4 Version 4

In this version, there are an equal number of employees and workstations, allowing all employees to work simultaneously. The number of employees is increased to three, and the number of tasks can vary. The inner workings and logic of this version is still the same as before, so we refer to descriptions of earlier versions for that. As this is the final version for the purpose of this project, it will be presented with more detail. The Petri net is shown in Figure 5.

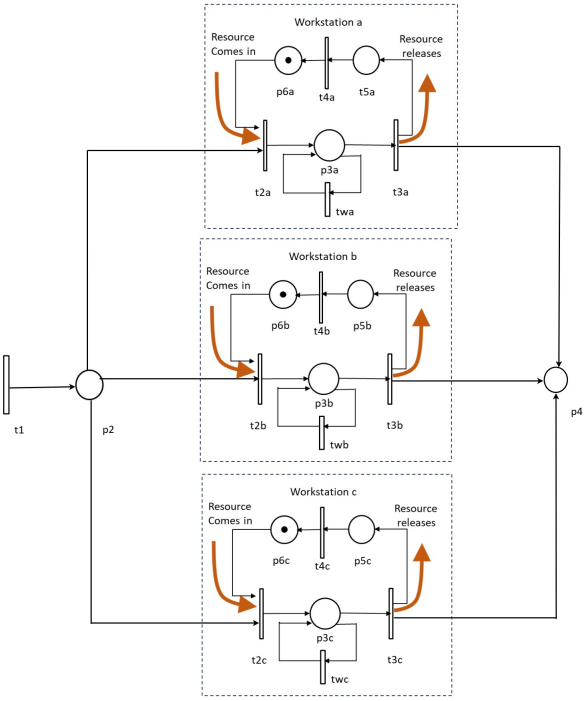


Figure 5: Petri net for Version 4.

### Tasks

A class was created to initialize the tasks, making the code clearer and facilitating the generation of random tasks for larger simulations. Tasks now carry two different types of properties: descriptive and recording.

### Descriptive properties

- Client (string): Not used for any computation or analysis, just to add flavor to the code.
- Size (number): Nominal size of the task, no units have been assigned as all simulations will run on mock data.
- Requirement ([array]): Reflection of potential certificates needed by employees to work on the task. Tasks will require either one certificate or none. If a task would require more than one certificate, it is split into two tasks before being put into the task pool.

- Composition ([array]): The fractional composition of the task in predefined categories. The categories can be any field requiring a corresponding skill to solve efficiently.

**Recording properties**

- TimeReceived (number): The time when the task entered the task pool.
- TimeStart (number): The time when work started on the task.
- AssignedEmp (string): The name of the employee that completed the task.
- TimeToFinish (number): The calculated time it will take to complete the task.
- TimeFinish (number): The time when the task was completed.

The property describing the assigned employee is important for the inner workings of the code. All the time signatures are recorded for diagnostic purposes during development and for analysis.

**Employees**

Similar to tasks, a class was written for initializing employees. Employees also have two different types of properties: ones describing the employee and ones that are recording parameters for analysis.

**Descriptive properties**

- Name (string): Name of the employee.
- Certification ([array]): Corresponding to the requirement of the tasks. An employee may have any number of certificates or none. The employee is only allowed to work on a task if they have the correct certificate.
- Competency ([array]): Corresponding to the composition of the tasks. Each element of the composition array functions as the work speed of the employee for the given category.
- Queue (number): Time units until the employee has finished all tasks currently assigned to them.

**Recording properties**

- Totaltime (number): Records the total time the employee has been working for the current simulation, allowing analysis of employee utilization.
- Totalwork (number): Records the cumulative work performed by the employee. As some employees are quicker workers than others, the work performed is not linearly scaled to time spent working across employees.

# 5 Testing, Analysis and Results

## 5.1 Simple User manual

1. To reproduce the results of Section 5.2, Figure 6, run the main simulation file `cc_v4.m` as is.

2. To reproduce the results of Section 5.3, Figure 7, comment out line 17, `TASKS ();` and comment "in" lines and 14, `% N = 20; % TASKS2 ( N );`.

3. Properties of tasks and employees can be altered in their respective initializing functions; `TASKS.m`, and `EMPLOYEES.m`.

Slightly more advanced use would include changing the number of employees and workstation, to tailor the model for a specific case. The code can also easily be expanded for a higher number of different certificates, or competency categories by adding more elements to respective arrays. It is important to update both corresponding properties for tasks and employees, such that the arrays contain the same number of elements. The code will not work if not, as it makes element wise comparisons of the pairs of arrays.

## 5.2 Sample run: 4 custom tasks

Presenting input to simple sample run. Employees are initialized by running function `EMPLOYEES.m`, and tasks are defined by running `TASKS.m`. Both scripts set up a relevant set of global variables for the simulation.

### Employees

```
EMPLOYEES_class(name, certification, competency, queue, totaltime, totalwork)
Andy = EMPLOYEES_class('Andy', [1, 1, 0], [2, 2, 1], 0, 0, 0);
Bria = EMPLOYEES_class('Bria', [0, 1, 1], [1, 1, 4], 0, 0, 0);
Cass = EMPLOYEES_class('Cass', [0, 0, 0], [4, 3, 3], 0, 0, 0);
```

Some key things to note regarding our team:

1. Between all team members, we have full coverage of certifications, so any task should be completable by someone in our team.

2. From the competency arrays, Cass seems to be a very efficient worker, but has no certificates. This means that Cass can only be assigned to tasks that require no certificates, but we can expect these tasks to be completed relatively quickly.

### Tasks

```
TASKS_class(client, size, requirement, composition)
T001 = TASKS_class('A', 10, [0, 1, 0], [0.5, 0.5, 0.0]);
T002 = TASKS_class('A', 30, [1, 0, 0], [0.2, 0.6, 0.2]);
T003 = TASKS_class('B', 60, [0, 0, 0], [0.5, 0.5, 0.0]);
T004 = TASKS_class('B', 90, [0, 0, 1], [0.1, 0.0, 0.9]);
```

Key things to note regarding our 4 custom tasks:

1. Task 2 can only be performed by Andy, and task 4 can only be performed by Bria.
2. Task 1 can be performed by either Andy or Bria, and task 3 can be performed by any team member.

Figure 6 shows the resource scheduling output from GPenSIM. In our case this shows how the generated tasks were allocated and completed by our team. In Table 1 a step-by-step explanation for every relevant time step in this simulation is shown.

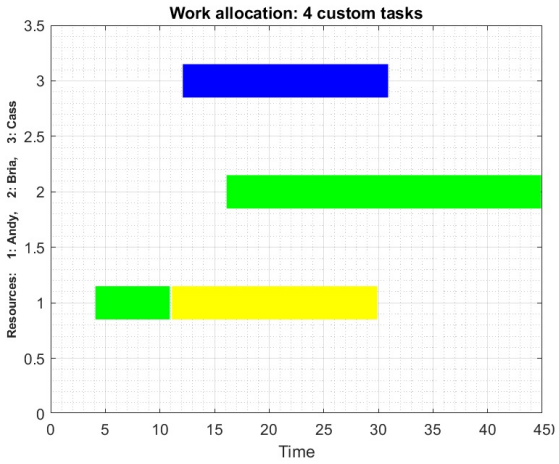


Figure 6: Gant chart: 4 custom tasks

Table 1: 4 Custom Tasks: Step-by-step Explanation

| **Time** | **Description** |
|---|---|
| 0 | Simulation starts |
| 1 | All employees are idle, as no tasks have been received. |
| 4 | **Task 1 is generated.** Both Andy and Bria are certified to complete this task. Andy is chosen due to having more relevant competency for performing the task. |
| 5 | Bria and Cass have not received tasks yet. |
| 8 | **Task 2 is generated.** Only Andy is certified to perform this task, so it cannot be assigned to anyone else. |
| 9 | Bria and Cass are still idle, watching Andy hard at work. |
| 11 | Andy finishes Task 1 and immediately starts working on Task 2. |
| 12 | **Task 3 is generated.** The task requires no certificate. Cass is chosen due to having more relevant competency for performing the task. |
| 16 | **Task 4 is generated.** Bria is the only idle worker and luckily has the correct certificate to perform the task. Work is commenced. |
| ... | Hereafter tasks are completed in respective times. No more tasks are available for our team at this time. |

## 5.3 Sample run: 20 generated tasks

Employees are initialized in the same way as in the first simulation. Tasks will now be randomly generated by the function `TASKS2.m`. A seed was set at the start of the main simulation file, `rng(123)`, to ensure replicability. By setting this seed, and setting number of tasks to `N = 20`, the result should be as shown in Figure 7.

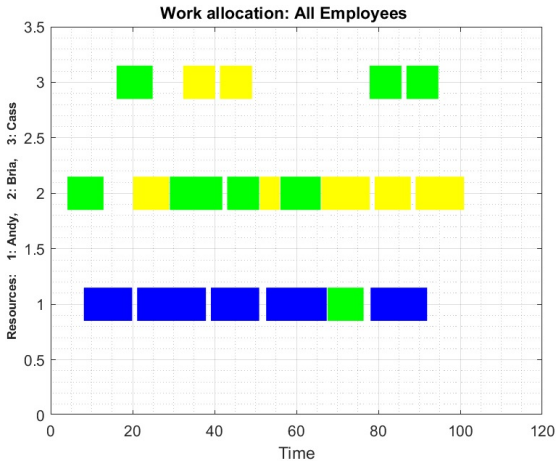


Figure 7: Gant chart: 20 generated tasks

Aside from tasks being (pseudo) randomly generated there is nothing fundamentally different from the 4-task example to this one. We can still notice a trend of Cass being assigned fewer tasks than the other two employees, which stems from the differing certifications.

## 5.4 Who should we fire?

Sadly, due to changes in business demand, we must let one member of our team go. Which is the least valuable employee to release?

To investigate this question the model will be run three more times, it fires one employee on each simulation. As the random seed was set before generating the 20 tasks, they will remain the same for all simulations. After analyzing the results it should be possible to visualize and examine which one is the least valuable employee. Results of the simulation is shown in Figure 8 and Table 2. Figure 8 shows how the 20 tasks would be allocated to the team if only two of them were available.

In this scenario, we make the following assumptions:

- Tasks are abundant, so lacking specific certifications is not a critical issue.
- Value generation is directly proportional to 'Total work,' which represents the work completed by the team.
- Employees are on payroll throughout the simulation, regardless of utilization.

- The primary goal is to maximize value creation in the shortest time, with no concern for sustainability.

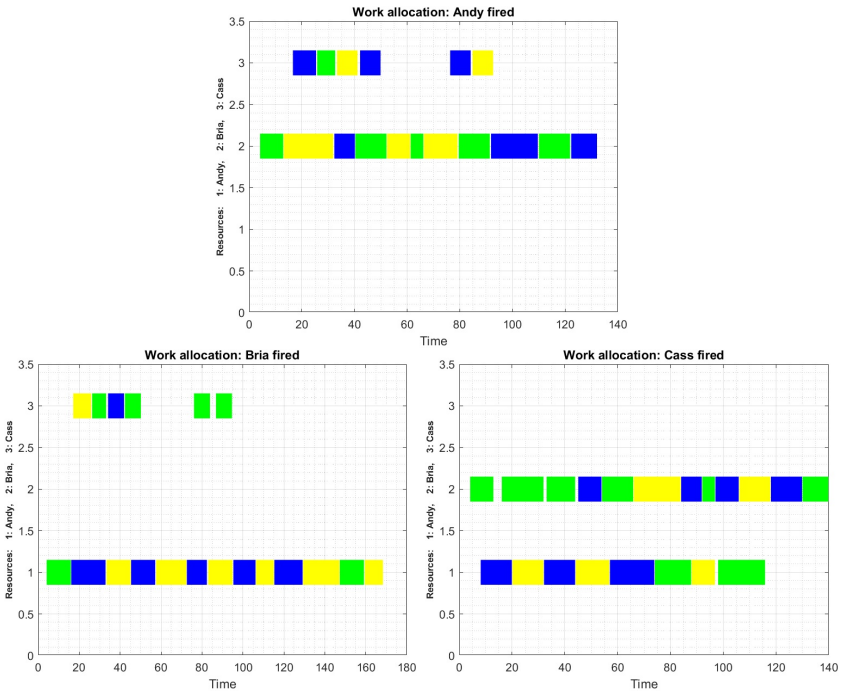


Figure 8: Comparison of task completion depending on which employee was fired

Table 2: Employee Utilization and Team Metrics

| | **Default** | **Fired Andy** | **Fired Bria** | **Fire Cass** |
|---|---|---|---|---|
| **Andy** | | | | |
| Total time | 67 | 0 | 137 | 91 |
| Total work | 120 | 0 | 233 | 159 |
| Utilization | 66% | 0% | 81% | 65% |
| **Bria** | | | | |
| Total time | 69 | 105 | 0 | 107 |
| Total work | 145 | 190 | 0 | 211 |
| Utilization | 68% | 79% | 0% | 76% |
| **Cass** | | | | |
| Total time | 31 | 36 | 36 | 0 |
| Total work | 105 | 121 | 121 | 0 |
| Utilization | 30% | 27% | 21% | 0% |
| **Team** | | | | |
| Time to complete all tasks | 102 | 133 | 169 | 141 |
| Completed tasks | 20 | 17 | 19 | 20 |
| Total work | 370 | 311 | 354 | 370 |
| Work per time | 3.6 | 2.3 | 2.1 | 2.6 |

### Analysis of Simulation Results

The table in Figure 2 was manually generated from recorded simulation data. Of particular note is the last segment, which shows how the team performed as a whole. The interpretation of the data can vary depending on the criteria used for evaluation. When considering termination decisions, it's essential to focus solely on numerical data, disregarding other factors such as seniority or office chocolate contributions.

Our objective is to retain the two team members who can generate the most value per unit of time. With these assumptions in mind, let's analyze the results.

#### Time to Complete All Tasks

The full team completed tasks significantly faster than after firing a team member, which aligns with our expectations.

#### Completed Tasks

The number of tasks resolved is directly influenced by team composition. A team without specific certifications, like Andy or Bria, is expected to resolve fewer tasks.

#### Total Work

This represents the sum of the sizes of all completed tasks. In cases where not all tasks were completed, this number is lower, which is consistent with our assumptions.

#### Work per Time

This metric is a key factor in our decision-making. It quantifies the efficiency of value generation by dividing the total work performed by the time taken to complete tasks. A full team generates 3.6 units of value per time. The team member with the least impact on this metric, should they be fired, is Cass. After letting Cass go, the remaining team can generate 2.6 units of value per time, which is the highest among the alternatives.

#### Conclusion

Based on these assumptions and metrics, retaining Andy and Bria while letting Cass go maximizes value generation per time.

## 5.5 Which certificate should Cass get?

Business is flourishing, and we are considering an investment in our employees to enhance company efficiency. The employee 'Cass' is our most efficient worker but lacks certifications. Out of the three available certificates (Certificate 1, Certificate 2, and Certificate 3), which one should we assign to Cass?

To inform this decision, we will conduct three additional simulations, each representing a scenario where Cass possesses one of the certificates. By employing similar metrics as in our initial analysis, we aim to determine which case will lead to the highest increase in the team's efficiency.

After a thorough analysis in our initial scenario, the subsequent process will be more concise. We will present the results and reach a swift conclusion with a similar underlying thought process as before.

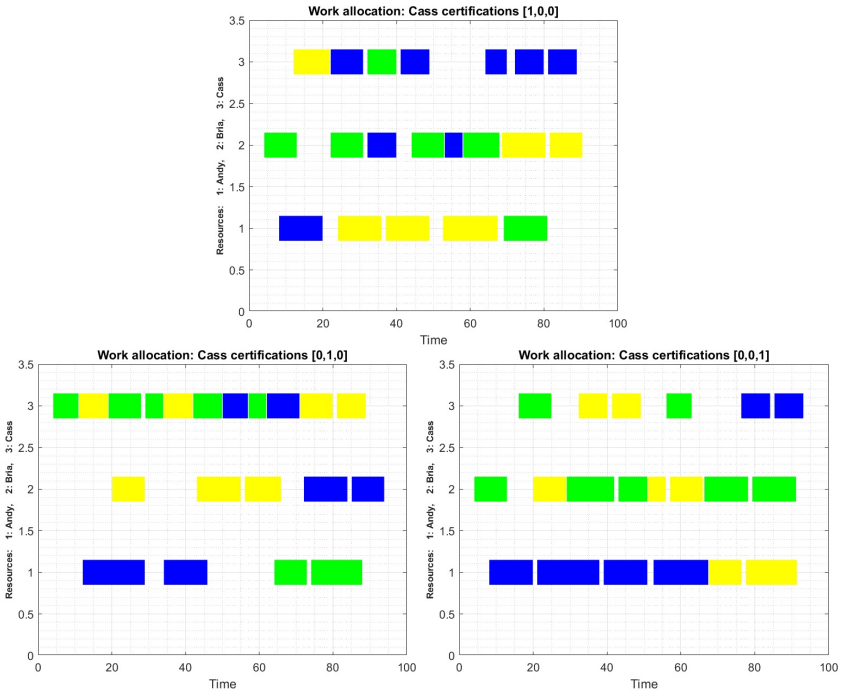


Figure 9: Comparison of task completion depending on which certificate is bestowed upon Cass

Table 3: Employee Performance with Different Certifications

| Metric | No Cert | Cert 1 | Cert 2 | Cert 3 |
|---|---|---|---|---|
| **Andy** | | | | |
| Total time | 67 | 53 | 44 | 67 |
| Total work | 120 | 97 | 77 | 120 |
| Utilization | 66% | 58% | 46% | 71% |
| **Bria** | | | | |
| Total time | 69 | 55 | 42 | 61 |
| Total work | 145 | 127 | 85 | 129 |
| Utilization | 68% | 60% | 44% | 65% |
| **Cass** | | | | |
| Total time | 31 | 43 | 61 | 36 |
| Total work | 105 | 146 | 208 | 121 |
| Utilization | 30% | 47% | 64% | 38% |
| **Team** | | | | |
| Time to complete all tasks | 102 | 91 | 95 | 94 |
| Completed tasks | 20 | 20 | 20 | 20 |
| Total work | 370 | 370 | 370 | 370 |
| Work per time | 3.6 | 4.1 | 3.9 | 3.9 |

**Analysis of Simulation Results**

- We observe increased productivity in all scenarios, regardless of which certificate is assigned to Cass. All three outcomes exhibit relative similarity.
- In all cases, all tasks are completed, which is expected as we have full certification coverage for all scenarios.
- With the amount of work being consistent, the most efficient outcome is the one that completes all tasks in the shortest time.

**Conclusion**

The most significant improvement in work efficiency is seen when Cass is assigned Certificate 1. The default value generation of the team, representing work efficiency, is 3.6. When Cass possesses Certificate 1, this value increases to 4.1, resulting in a 14% boost in team productivity.

The assumptions, results, and conclusions presented in this chapter are open to discussion. However, the goal here is to demonstrate the types of questions such a model can help us address.

# 6 Discussion and Conclusion

## 6.1 Originality and Relevant Works

Using Petri nets for modeling resource allocation is more than common. Other studies have been seen to focus on resource requirement reduction [6], or even combined with machine learning to optimize human resourcing [7]. Even if the idea is not new to the world, it was a new idea to the authors, and such the work can still be viewed as an original contribution.

This model was initially created with a certain idea, but without specific questions in mind. After finishing the model we found it easy to adapt the code to investigate the questions we came up with.

In our adaptation there are an equal amount of employees and workstations. This will not be the case for every case. One way that the relationship between the number of employees and workstations can be leveraged is in a situation where work space is more limited than the number of workers. It doesn't matter how many mechanics you hire if you can only fit one car in your workshop at a time. In such a case a sensitivity analysis towards the number of workstations could be relevant.

## 6.2 Limitations and Future Work

Here are some thoughts to current limitations that may be addressed in future versions.

1. The introduction of a 'priority' property for tasks, allowing high-priority tasks to be preferred even if they arrived after lower-priority tasks.

2. Implementation of a mechanism that enables employees with the correct certification to work alongside uncertified employees, thus speeding up task completion. This can be compared to bringing along a trainee or apprentice.

3. The incorporation of functionality for cooperative tasks that require more than one person to complete. These tasks can specify a range of acceptable numbers of employees needed for the task. For example, a range of 1-3 means that any number of employees between one and three can simultaneously work on the task. If the range is 2-2, then exactly two employees are required for task completion.

4. Add function to prioritize tasks based on their entry time. GPenSIM offers a built-in function to prioritize early arrived tokens[5]. However, applying this function will cause a workstation to wait for its preferred employee rather than looking for a task that another idle employee can work on.

    - A possible way to solve this problem is to increase the number of workstations beyond the number of employees. This way the likelihood of having an idle employee is reduced, and tasks will be prioritized based on arrival time. This method is crude, however, and there is bound to be more elegant solutions as well.

5. Improve naming conventions for scripts, places and transitions to make code more intuitive.

6. Modularize the work station concept to make the code more easily adaptable to new situations.

After mentioning so many possible additions to the code it is worth mentioning that adding complexity usually comes at a cost. It is not advisable to add more complexity than is necessary to solve the task at hand. Unless the code is very well written the interaction of different added features may produce unpredictable results.

## 6.3 Reflections

### Jing

This project proves to be both experimental and rewarding. We came up with an interesting topic that can be tested out and provide decision-making in job task allocation real-life scenarios which was inspired by the analysis of Petri nets from the lecture materials. My teammate has given significant support during the project. The interpretation of the simulation results match with our expectation about human resource utilization in consulting firms and the interesting findings actually encourage us to experiment other models in real-life scenarios. However, there are some frustrations with the GenPenSim in Matlab. Another wish would be the possible alternative in the future to implement the Petri Net model and simulation in Python instead of Matlab.

### Daniel

As a novice in both MATLAB, Petri net theory and GPenSIM, I'm actually quite pleased with how this project turned out. Armed only with an idea, we were able to achieve 'more or less' exactly what we set to do. With 'more or less' I mean that we did not add as many features to model as was discussed in the project proposal. After having a working Version 4, and producing some interesting results, it felt natural to close off the coding and move onto analysis.

Through the project I have learned a lot and had fun while doing so. Solving problems of this sort in this way feels like a game of logic, which I found myself enjoying (for the most part). The widespread applicability of Petri net modelling combined with the open nature of the project left space for creativity, which in turn led to a great feeling of ownership to the project.

As a group, I feel that Jing and I cooperated very well. Communication is key, and a common sense of responsibility is also helpful. I believe this product is greater than what either of us would have achieved by themselves. You could say that this project is a Task best suited for 2 Employees with a combined balance of competency.

# References


[1] R. Sandberg and A. Werr, "The three challenges of corporate consulting," *MITSloan Management Review*, 2003.

[2] K. Blog, "7 common consulting firm project management challenges," 2023.

[3] M. Ejenas and A. Werr, "Managing internal consulting units: Challenges and practices," *S.A.M. Advanced Management Journal*, 2011.

[4] P. P. Phillips and J. J. Phillips, "Building a successful consulting practice opportunities and challenges," *ASTD Press*, 2004.

[5] R. Davidrajuh, *Modeling Discrete-Event Systems with GPenSIM An Introduction*. Springer, 2018.

[6] G. Mejía and C. Montoya, "Applications of resource assignment and scheduling with petri nets and heuristic search," *Annals of Operations Research*, 2010.

[7] W. Dai, Y. Hu, Z. Zhu, and X. Liao, "Human resource petri net allocation model based on artificial intelligence and neural network," *Mobile Information Systems*, vol. 2021, 08 2021.

# A Code Listings - Version 4

## A.1 Main simulation file, cc_v4.m

```
%  Consultant Company, Version 4
% main simulation file (MSF) to run simulation

clear all; clc;
global global_info

global_info.STOP_AT = 300;

% Set the random seed for replicability
rng(123);

% Load tasks from task generator:
% N = 20;
% TASKS2(N);

% Load tasks from task list
TASKS();

% To investigate which employee is the least valuable
    we can simulate
% firing either employee, {'Andy', 'Bria', 'Cass'}
global_info.fire_employee = '';

% Load employees
EMPLOYEES();

global_info.task_index = 0;

pns = pnstruct('cc_v4_pdf');

dyn.m0 = {'p6a',1, 'p6b',1, 'p6c',1};
dyn.ft = {'t1',4 , ...
          't2a',1 , 'twa',1 , 't3a',1, 't4a',1, ...
          't2b',1 , 'twb',1 , 't3b',1, 't4b',1, ...
          't2c',1 , 'twc',1 , 't3c',1, 't4c',1  ...
         };
dyn.re = {'Andy',1,inf, 'Bria',1,inf, 'Cass',1,inf};

pni = initialdynamics(pns, dyn);

Sim_Results = gpensim(pni);
prnss(Sim_Results);
% plotp(Sim_Results, {'p2','p3a','p4'});
% plotp(Sim_Results, {'p2'});
plotGC(Sim_Results)

```

```
disp('Andy')
disp([{'Totaltime:', 'Totalwork:'};
    num2cell([global_info.Andy.Totaltime,
    global_info.Andy.Totalwork])])
disp('Bria')
disp([{'Totaltime:', 'Totalwork:'};
    num2cell([global_info.Bria.Totaltime,
    global_info.Bria.Totalwork])])
disp('Cass')
disp([{'Totaltime:', 'Totalwork:'};
    num2cell([global_info.Cass.Totaltime,
    global_info.Cass.Totalwork])])
```

## A.2 cc_v4_pdf.m

```
%  Consultant Company, Version 4

function [pns] = cc_v4_pdf()

pns.PN_name = 'Consultant Company Version 4';
pns.set_of_Ps = {'p2', ...
                 'p3a', 'p5a', 'p6a', ...
                 'p3b', 'p5b', 'p6b', ...
                 'p3c', 'p5c', 'p6c', ...
                  'p4' ...
                  };
pns.set_of_Ts = {'t1', ...
                 't2a', 't3a', 'twa','t4a', ...
                 't2b', 't3b', 'twb','t4b', ...
                 't2c', 't3c', 'twc','t4c'  ...
                };
pns.set_of_As = {'t1','p2',1 , ...
                 'p2','t2a',1, 't2a','p3a',1,
                    'p3a','t3a',1, 't3a','p4',1 ,...
                 'p3a','twa',1 ,'twa','p3a',1 , ...
                 't3a','p5a',1,'p5a','t4a',1,
                    't4a','p6a',1 , 'p6a','t2a',1, ...
                 'p2','t2b',1, 't2b','p3b',1,
                    'p3b','t3b',1, 't3b','p4',1 ,...
                 'p3b','twb',1 ,'twb','p3b',1 , ...
                 't3b','p5b',1,'p5b','t4b',1,
                    't4b','p6b',1 , 'p6b','t2b',1, ...
                 'p2','t2c',1, 't2c','p3c',1,
                    'p3c','t3c',1, 't3c','p4',1 ,...
                 'p3c','twc',1 ,'twc','p3c',1 , ...
                 't3c','p5c',1,'p5c','t4c',1,
                    't4c','p6c',1 , 'p6c','t2c',1  ...
                };
```

## A.3 COMMON_PRE.m

```
% Consultant Company, Version 4
function [fire, transition] = COMMON_PRE(transition)

global global_info

switch transition.name

    case 't1'
        if eq(global_info.task_index,
            global_info.Taskpool_Size)
            granted = 0;
        else
            global_info.task_index =
                global_info.task_index + 1;
            color =
                global_info.tasks(global_info.task_index);
            transition.new_color = color;

            % Document time when task is put into 'p2'
            firing_time = get_trans('t1').firing_time;
            global_info.(color{1}).TimeReceived =
                current_time() + firing_time;
            granted = 1;
        end
    case {'t2a', 't2b', 't2c'}
        % Grab a random token from 'p2'
        % Step made to avoid clogging up a
            workstation with a task
        % that an idle employee cannot solve
        N = ntokens('p2');
        tokIDis = tokenAny('p2', N);
        tokIDi = tokIDis(randi(N));

        color = get_color('p2', tokIDi);
        task = color{1};

        % Find possible employees
        possible_employees =
            find_Possible_Employees(task);

        % If no elible employees, do nothing
        if  isempty(possible_employees)
            granted = 0;
        else
            % Find preferred employees and task
                completion time
            [preferred_employee, task_time] =
                find_Preferred_Employee(task,
```

```
        possible_employees);

    % Request preferred employee as specific resource
    reserved = requestSR({preferred_employee,1});

    % If preferred employee is available update variables and fire
    if eq(reserved, 1)
        transition.selected_tokens = tokIDi;

        % Document time started
        global_info.(task).TimeStart = current_time();

        % Document employee assigned to task
        global_info.(task).AssignedEmp = preferred_employee;

        % Document expected time to finish task
        global_info.(task).TimeToFinish = task_time;

        % Update work queue for selected employee
        global_info.(preferred_employee).Queue =
            global_info.(preferred_employee).Queue
            + task_time;

        % Update total time spent working by selected employee
        global_info.(preferred_employee).Totaltime =
            global_info.(preferred_employee).Totaltime
            + task_time;

        % Update total amount of work performed by selected employee
        task_size = global_info.(task).Size;
        global_info.(preferred_employee).Totalwork =
            global_info.(preferred_employee).Totalwork
            + task_size;

        % Update pritority of 'tw', the work station to simulate work
        lastChar = transition.name(end);
```

```
70                  tw = ['tw', lastChar];
71                  priorset(tw, task_time)
72                  granted = 1;
73              else
74                  granted = 0;
75              end
76          end
77      % Work station retains token as long as it has
            priority
78      case {'twa', 'twb', 'twc'}
79          lastChar = transition.name(end);
80          tw = ['tw', lastChar];
81          t3 = ['t3', lastChar];
82          p3 = ['p3', lastChar];
83
84          if lt(priorcomp(tw, t3),1)
85              granted = 0;
86          else
87              % Find employee assigned to current task
88              tokIDi = tokenAny(p3, 1);
89              color = get_color(p3, tokIDi);
90              name = global_info.(color{1}).AssignedEmp;
91
92              % Find firing time for current transition
93              firing_time = get_trans(tw).firing_time;
94
95              % Subtract firing time from employee queue
96              global_info.(name).Queue =
                    global_info.(name).Queue - firing_time;
97
98              % Decrease priority of work station
99              priordec(tw)
100             granted = 1;
101         end
102
103     case {'t3a', 't3b', 't3c'}
104         lastChar = transition.name(end);
105         tw = ['tw', lastChar];
106         t3 = ['t3', lastChar];
107         p3 = ['p3', lastChar];
108
109         % Disable transition while 'tw1', the work
                station, has higher
110         % priority
111         if gt(priorcomp(tw, t3),0)
112             granted = 0;
113         else
114             tokIDi = tokenAny(p3, 1);
115             color = get_color(p3, tokIDi);
116             transition.selected_tokens = tokIDi;
```

```
117
118             % Document task finishing time
119             global_info.(color{1}).TimeFinish =
                    current_time();
120             granted = 1;
121         end
122     case {'t4a', 't4b', 't4c'}
123         transition.override = 1;
124         granted = 1;
125 end
126 fire = granted;
```

## A.4 COMMON_POST.m

```
1  % Consultant Company, Version 4
2  function [fire, transition] = COMMON_POST(transition)
3
4  switch transition.name
5      case {'t1'}
6          % Do nothing.
7      case {'t3a', 't3b', 't3c'}
8          lastChar = transition.name(end);
9          t2 = ['t2', lastChar];
10         release(t2)
11 end
```

## A.5 TASKS.m

```
1  % Consultant Company, Version 4
2  function [] = TASKS()
3
4  global global_info
5
6  global_info.tasks = {'T001', 'T002', 'T003', 'T004'};
7
8  T001 = TASKS_class('A', 10, [0, 1, 0], [0.5, 0.5,
       0.0]);
9  T002 = TASKS_class('A', 30, [1, 0, 0], [0.2, 0.6,
       0.2]);
10 T003 = TASKS_class('B', 60, [0, 0, 0], [0.5, 0.5,
       0.0]);
11 T004 = TASKS_class('B', 90, [0, 0, 1], [0.1, 0.0,
       0.9]);
12
13 global_info.T001 = T001;
14 global_info.T002 = T002;
15 global_info.T003 = T003;
```

```
global_info.T004 = T004;
global_info.Taskpool_Size = 4;
```

## A.6 TASKS2.m

```
% Consultant Company, Version 4
function [] = TASKS2(N)
% Generates N random tasks based on input given below

global global_info

% Initialize the global_info structure
global_info.Taskpool_Size = N;
global_info.tasks = cell(1, N);

% Populate the global_info structure
for taskNum = 1:N
    % Generate random properties
    clientOptions = ['A', 'B', 'C'];
    client =
       clientOptions(randi(length(clientOptions)));
    size = randi([10, 30]);
    requirement = [0, 0, 0];
    if rand() < 0.6
        requirement(randi(3)) = 1;
    end
    composition = [rand, rand, rand];
    composition = composition / sum(composition);

    % Create the task and assign it to the
       global_info structure
    taskName = ['T', num2str(taskNum, '%03d')];
    task = TASKS_class(client, size, requirement,
       composition);
    global_info.(taskName) = task;
    global_info.tasks{taskNum} = taskName;
end
```

## A.7 TASKS_class.m

```
% Consultant Company, Version 4
% Employee class

classdef TASKS_class
    properties
        Client
        Size
```

```
        Requirement
        Composition
        TimeReceived
        TimeStart
        AssignedEmp
        TimeToFinish
        TimeFinish
    end

    methods
        function obj = TASKS_class(client, size,
            requirement, composition)
            obj.Client = client;
            obj.Size = size;
            obj.Requirement = requirement;
            obj.Composition = composition;
            obj.TimeReceived = '';
            obj.TimeStart = '';
            obj.AssignedEmp = '';
            obj.TimeToFinish = '';
            obj.TimeFinish = '';
        end
    end
end
```

## A.8 EMPLOYEES.m

```
% Consultant Company, Version 4
% "index coloring" for specific resources / employees

function [] = EMPLOYEES()

global global_info

%global_info.employees = {'Andy', 'Bria', 'Cass'};

% EMPLOYEES_class(name, certification, competency,
    queue, totaltime, totalwork)
Andy = EMPLOYEES_class('Andy', [1, 1, 0], [2, 2, 1],
    0, 0, 0);
Bria = EMPLOYEES_class('Bria', [0, 1, 1], [1, 1, 4],
    0, 0, 0);
Cass = EMPLOYEES_class('Cass', [0, 0, 0], [4, 3, 3],
    0, 0, 0);

global_info.Andy = Andy;
global_info.Bria = Bria;
global_info.Cass = Cass;

```

```
% update list of employees if any of them have been
    fired

if isfield(global_info, 'fire_employee') &&
    ~isempty(global_info.fire_employee)
    switch global_info.fire_employee
        case 'Andy'
            global_info.employees = {'Bria', 'Cass'};
        case 'Bria'
            global_info.employees = {'Andy', 'Cass'};
        case 'Cass'
            global_info.employees = {'Andy', 'Bria'};
    end
else
    global_info.employees = {'Andy', 'Bria', 'Cass'};
end
```

## A.9 EMPLOYEES_class.m

```
% Consultant Company, Version 4
% Employee class

classdef EMPLOYEES_class
    properties
        Name
        Certification
        Competency
        Queue
        Totaltime
        Totalwork
    end
    methods
        function obj = EMPLOYEES_class(name,
            certification, competency, queue,
            totaltime, totalwork)
            obj.Name = name;
            obj.Certification = certification;
            obj.Competency = competency;
            obj.Queue = queue;
            obj.Totaltime = totaltime;
            obj.Totalwork = totalwork;
        end
    end
end
```

## A.10 find_Possible_Employees.m

```
% Consultant Company, Version 4
% Function returns all employees who have the
    certification needed to work
% on the given task
function candidates = find_Possible_Employees(task)

    global global_info

    % Retrieve requirements for given task
    task_requirement = global_info.(task).Requirement;

    % Initiate  return variable
    candidates = {};

    for employee = global_info.employees

        % Access employee data using the global
            structure
        certification =
            global_info.(employee{1}).Certification;

        if ~any(task_requirement - certification > 0)
            candidates{end+1} = employee{1};
        end
    end
end
```

### A.11 find_Preferred_Employee.m

```
% Consultant Company, Version 4
% Function returns the employee who will finish the
    task first based on the
% composition of the task and the competency of the
    employee

function [preferred_employee, task_time_return] =
    find_Preferred_Employee(task, possible_employees)

    global global_info

    % Retrieve task size from global memory
    task_size = global_info.(task).Size;
    task_composition = global_info.(task).Composition;

    record_time = 10000;
    % Initialize return variable
    preferred_employee = [];

```

```
17      % Iterate through possible employees
18      for employee = possible_employees
19
20          % Work speed is calculated from the
                relaationship between employee
21          % competency and composition of given task
22          speed =
                sum(times(global_info.(employee{1}).Competency,
                task_composition));
23          queue = global_info.(employee{1}).Queue;
24
25          % Calculate time taken for employee to finish
                current queue and new
26          % task
27          task_time = task_size / speed;
28          total_time = queue + task_time;
29
30          %Fastest employee is chosen
31          if total_time < record_time
32              preferred_employee = employee{1};
33              task_time_return = round(task_time);
34              record_time = total_time;
35          end
36      end
37  end
```