Heuristic Algorithms - Assessment Task Protocol

Analysis of Heuristics Methods for Acitivity Dependency Problems

Lukáš Matthew Čejka

July 19, 2023

Contents

1 Introduction

The development of large-scale projects often requires vast resources, however, the resources available are often limited. Therefore, completing projects with limited resources before a deadline requires meticulous planning. Specifically, the resources available at any point in time must be utilized as efficiently as possible. This goal is one of the key aspects of project management and scheduling.

Many methods that aid in planning the activities of a project with limited resources exist; referred to as *activity dependency problems* in this protocol. Among the most well-known methods is the Critical Path Method (CPM) [1] introduced by J. E. Kelley; M. R. Walker. Other methods include heuristics such as those presented in *Řízení projektů* [2]. While these methods are predominantly used in project management, they can also be used to plan jobs of a data-processing engine. For this purpose, the protocol aims to introduce a selection of heuristic methods and compare them on a set of problems.

First, the theory behind the heuristic methods is introduced. Next, the implementation done as part of the assessment project is briefly described. Then, the results of the comparison of the heuristic methods across several problems is presented. Finally, the last section summarizes the contents of this protocol.

2 Theory

To facilitate a better understanding of the heuristic methods presented in this protocol, key concepts must first be introduced.

Problem In the context of this protocol, a *problem* represents a project that is made up of activities. Each activity takes a certain amount of time to complete and a set amount of resources at each point in time. To complete the project, all activities must be finished while adhering to their dependencies and the resources available for the project at each point in time. The challenging aspect of the problem is to complete the project in the shortest time possible while keeping to the dependencies, and not exceeding the resources available. The activities and their relationships within a project can be represented using an event-oriented directed graph, called the *CPM network*.

In a CPM network, the nodes represent milestones that mark the start or completion of activities, while the directed, weighted edges represent the activities and their duration. For this project, it is assumed that there is only one start node initiating the project and one or more end nodes that represent the completion of the project. An example of such a graph is presented in Figure 1.

For clarity, the term *dependency of activities* means that an activity cannot be started before all of its predecessors have been completed. For example, in Figure 1, activity 5-6 can only be started once activities 2-5 and 3-5 have been completed.

The graph presented in Figure 1 is referred to as a CPM network due to its widespread use during the analysis of projects using the Critical Path Method (CPM) [1].

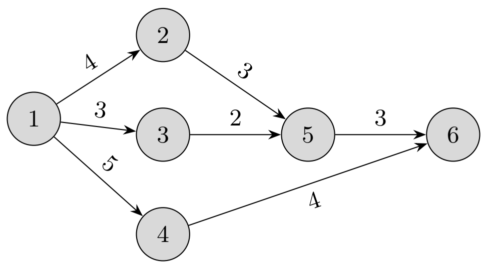

Figure 1: Visualization of a project comprising activities as a CPM network. Each node represents a milestone that either marks the start or the completion of activities. The number inside each node is the ID of the milestone represented by the node. Each weighted edge represents an activity and its duration. The ID of an activity is determined from the node it starts from and the node it ends in, i.e. i-j, where i is the ID of the start node and j is the ID of the end node. For example, the ID of the activity starting in node 1 and ending in node 2 is 1-2 and its duration is 4.

Critical Path Method (CPM) The Critical Path Method is an algorithm used for scheduling activities of a project. It is used to determine the earliest possible end of the project while only adhering to the dependences of activities and their durations, i.e., not their resources. Given its dependencies and duration ($t_{i,j}$), for each activity, the algorithm of CPM determines the following:

- Earliest Start (ES) - Earliest possible time the activity can *start* considering its dependencies.
- Earliest End (EE) - Earliest possible time the activity can *end* considering its dependencies.
- Latest Start (LS) - Latest permissible time the activity can *start* considering its dependencies.
- Latest End (LE) - Latest permissible time the activity can *end* considering its dependencies.
- Time Reserves (TR) - Number of time units that the activity can be delayed before the end of the project must be delayed. It is equal to the difference between LE and EE (or LS and ES).

If the time reserves for an activity are equal to zero, then delaying the activity means delaying the project. Activities with zero time reserves are referred to as *critical activities* and they make up the so-called *critical path*. The critical path is a sequence of dependent activities between the start and end nodes that dictates when the project can be completed earliest.

For example, the output of CPM for the project shown in Figure 1 is presented in Table 1.

The activity timeline produced by CPM can be visualized using a Gantt chart[1], as shown in Figure 2.

The algorithm of CPM is omitted as the output of CPM is the input for the heuristic methods that schedule the activities of a project while adhering to the dependencies, the durations, and the resources.

[1]Gantt chart Wikipedia URL: `https://en.wikipedia.org/wiki/Gantt_chart`

i-j	$t_{i,j}$	$\text{ES}_{i,j}$	$\text{EE}_{i,j}$	$\text{LS}_{i,j}$	$\text{LE}_{i,j}$	$\text{TR}_{i,j}$
1-2	4	0	4	0	4	0
1-3	3	0	3	2	5	2
1-4	5	0	5	1	6	1
2-5	3	4	7	4	7	0
3-5	2	3	5	5	7	2
4-6	4	5	9	6	10	1
5-6	3	7	10	7	10	0

Table 1: Output of CPM for the project presented in Figure 1. The critical path is made up of activities 1-2, 2-5, and 5-6 since their TRs are equal to zero. The last activities, 4-6 and 5-6, are both completed at time unit 10, therefore, the entire project can be completed in 10 time units.

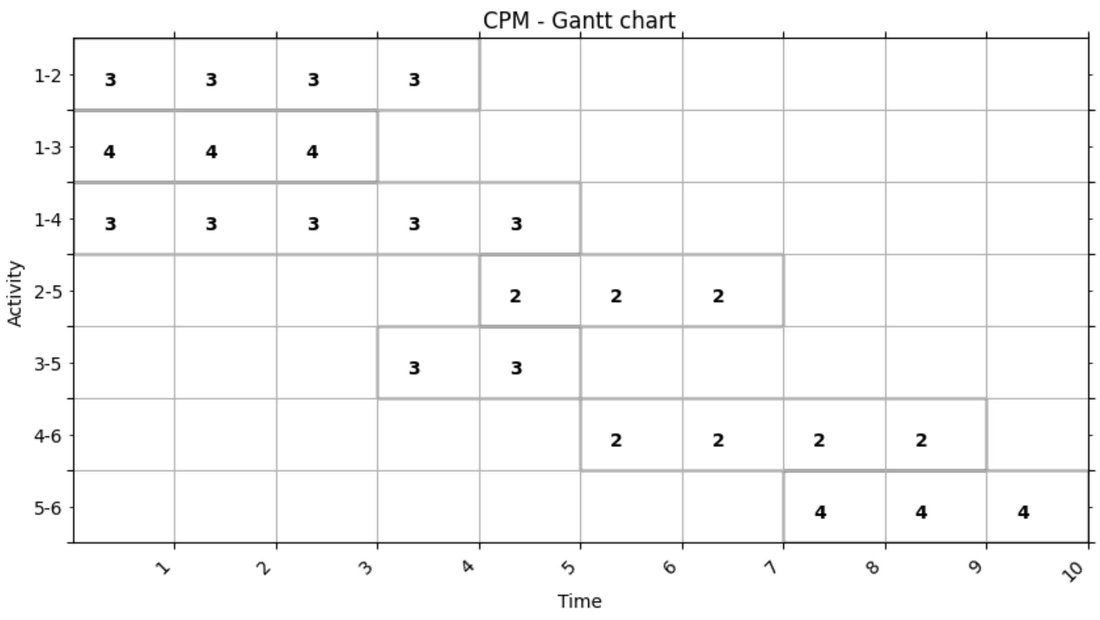

Figure 2: Timeline of the activities presented in Table 1 as scheduled by CPM. The vertical axis represents the activity, while the horizontal axis displays the time units. The numbers in the squares represent the units of resources required by each activity at a point in time. Note that CPM does not take the resources into account, they are only displayed here for completeness. The image was generated using the Matplotlib[2] Python[3] package.

2.1 Heuristic Methods

As mentioned earlier, CPM determines the earliest possible end of a project while taking into account only the dependencies of activities and their durations. On the other hand, the heuristic methods analyzed in this protocol also take into account the resources required by each activity at a point in time.

The heuristic methods selected from from *Řízení projektů* [2] for this assessment project are the

[2]Matplotlib webpage URL: `https://matplotlib.org/stable/index.html`
[3]Python webpage URL: `https://www.python.org`

following:

- Serial Heuristic Method (SHM)
- Parallel Heuristic Method (PHM)
- Parallel Heuristic Method with Dynamic Priorities (PHMDP)

All of the above-mentioned heuristic methods use the output of CPM as their input. Additionally, the heuristic methods take in the $r_{\max}$ parameter which represents the maximum number of resources available at each point in time.

2.2 Serial Heuristic Method (SHM)

Using the output of CPM and the provided $r_{\max}$, the Serial Heuristic Method performs the following steps:

1. Arrange the activities into a sequence $(i_1, j_1), \ldots, (i_m, j_m)$ so that $i_k < i_{k+1}, j_k < j_{k+1}$.
2. Starting with the first activity, schedule each activity while adhering to the dependencies and the maximum resources available at a point in time ($r_{\max}$).

To demonstrate the process, the activity timeline for the example project scheduled using SHM with $r_{\max} = 6$ is shown in Figure 3.

Specifically, the order of activities in the sequence is identical to the order of activities in column (i, j) in Table 1. SHM schedules activity 1-2 from time 0. Then it tries to schedule activity 1-3, however, it can only do so from time 4 when activity 1-2 has finished as the resources required by both activities would exceed the resources available $3 + 4 = 7 > 6$. According to SHM, the project can be completed in 16 time units while adhering to both dependencies and the available resources.

From Figure 3, it can be seen that SHM is noticeably inefficient. For example, from time 0, activities 1-2 and 1-4 can be scheduled while not exceeding the available resources. However, since activity 1-3 is scheduled earlier, activity 1-4 cannot be scheduled until time unit 7. This inefficiency is addressed by the *Parallel Heuristic Method*.

2.3 Parallel Heuristic Method (PHM)

Unlike SHM, the Parallel Heuristic Method (PHM) provides each activity with a priority equal to its Time Reserves (TR) value. Counterintuitively, the lower the priority value of an activity, the higher the priority of the activity. This is due to TR representing the number of time units that an activity can be delayed for. Thus, if $\mathrm{TR}_{\mathrm{i,j}} = 0$, then activity *i*-*j* must be scheduled as soon as possible, otherwise the project may be delayed.

Using the output of CPM and the provided $r_{\max}$, PHM performs the following steps:

1. For every time unit:

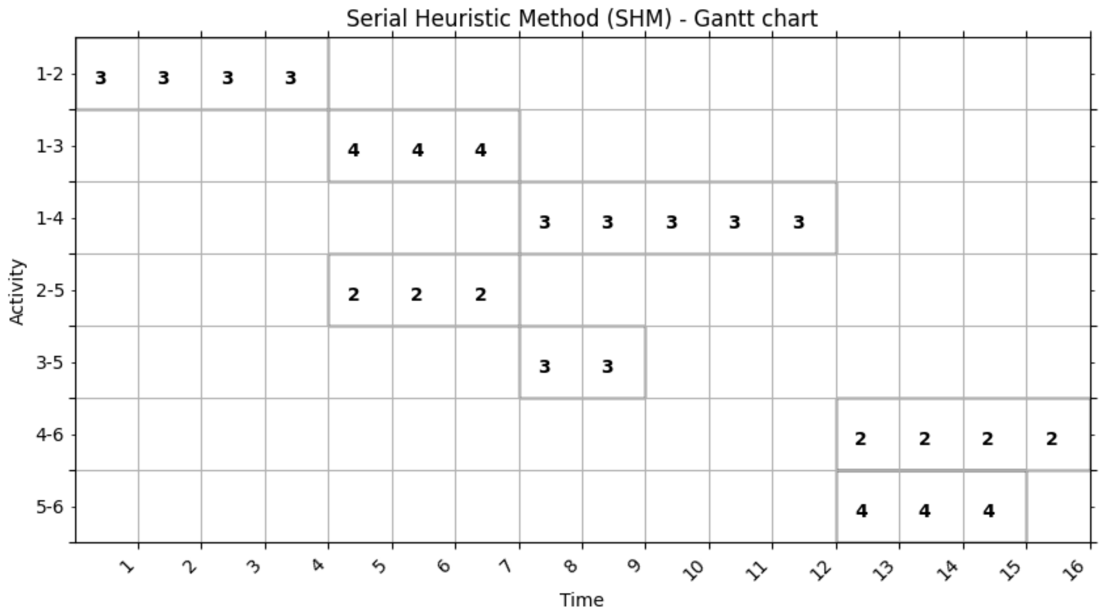

Figure 3: Timeline of the activities presented in Table 1 as scheduled by SHM. The vertical axis represents the activity, while the horizontal axis displays the time units. The numbers in the squares represent the units of resources required by each activity at a point in time.

(a) Create a set of activities whose predecessors are all completed.

(b) Arrange the activities in the set in ascending order according to their TR value. If two activities have the same TR value, then they are ordered using their ID as done in SHM: $(i_1, j_1), \ldots, (i_m, j_m)$ so that $i_k < i_{k+1}, j_k < j_{k+1}$.

(c) Starting with the activity with the lowest TR value, schedule as many activities from the time unit as possible while adhering to the resources available in every time unit.

The activity timeline for the example project scheduled using PHM with $r_{\max} = 6$ is shown in Figure 4.

As can be seen from Figure 4, unlike SHM, PHM scheduled activities 1-2 and 1-4 to run parallel. The prioritization of activities allows PHM to determine that the project can be completed in 15 time units while adhering to both the dependencies and available resources.

However, PHM has a weakness: statically set priorities. The priorities set by PHM do not change with the flow of time. Therefore, the completion of non-critical activities can be delayed, thus causing suboptimal scheduling. For example, since the TR value of activity 1-3 is 2, its scheduling is delayed until time 7. Therefore, activities 3-5 and 5-6 are not even considered until time 10 which delays the entire project. This flaw is addressed using the *Parallel Heuristic Method with Dynamic Priorities.*

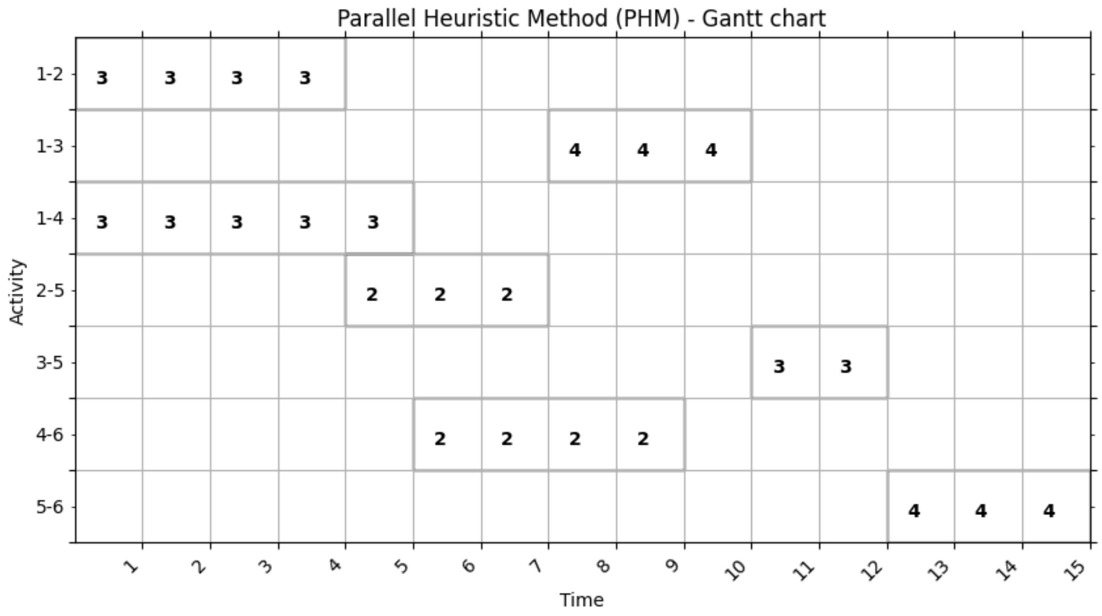

Figure 4: Timeline of the activities presented in Table 1 as scheduled by PHM. The vertical axis represents the activity, while the horizontal axis displays the time units. The numbers in the squares represent the units of resources required by each activity at a point in time.

2.4 Parallel Heuristic Method with Dynamic Priorities (PHMDP)

The Parallel Heuristic Method with Dynamic Priorities (PHMDP) sets the priority value of each activity to $\mathrm{LS}_{i,j} - t$, where t represents the time. Furthermore, for every t, PHMDP updates the priorities, which eliminates the inadequacy of PHM.

Using the output of CPM and the provided $r_{\max}$, PHMDP performs the following steps:

1. For every time unit t:
   - (a) Create a set of activities whose predecessors are all completed.
   - (b) Update the priorities for the activities to $\mathrm{LS}_{i,j} - t$.
   - (c) Arrange the activities in the set in ascending order according to their priority value. If two activities have the same priority value, then they are ordered using their ID as done in SHM: $(i_1, j_1), \ldots, (i_m, j_m)$ so that $i_k < i_{k+1}, j_k < j_{k+1}$.
   - (d) Starting with the activity with the lowest priority value, schedule as many activities from the time unit as possible while adhering to the resources available in every time unit.

The activity timeline for the example project scheduled using PHMDP with $r_{\max} = 6$ is shown in Figure 5.

As shown in Figure 5 the dynamic priorities allow for activities to be scheduled more efficiently. Ultimately, this results in PHMDP determining that the project can be completed in 13 time units while adhering to both the dependencies and available resources.

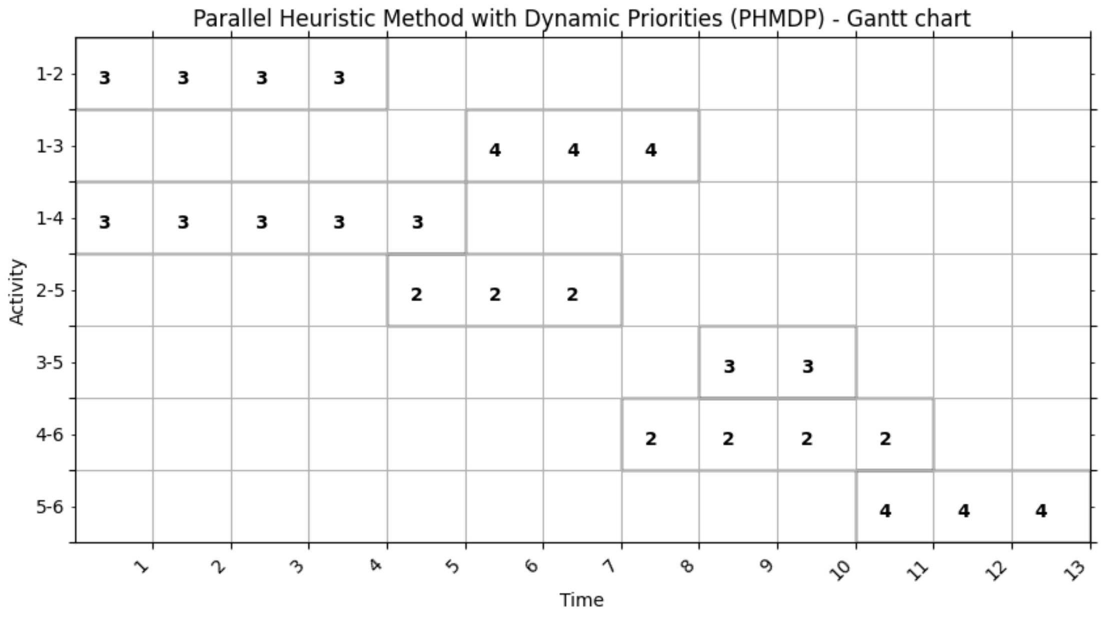

Figure 5: Timeline of the activities presented in Table 1 as scheduled by PHMDP. The vertical axis represents the activity, while the horizontal axis displays the time units. The numbers in the squares represent the units of resources required by each activity at a point in time.

To examine the behavior of the methods further, they were implemented and compared on more examples.

3 Implementation

In this section, the implementation of the project comprising the methods introduced in Section 2 is presented. The source code is available on request or in the project's GitHub repository[4]. The project was implemented in Python (version 3.9.6[5]) as it offers clean data structures and support for visualization tools.

The project uses the following *make*[6] tasks:

- `make init` - Download and install the required Python packages.
- `make tests`, `make tests_coverage`, and `make tests_coverage_report` - Run the unit tests using *nose2*[7] alone, with coverage, and with coverage generated into an HTML report, respectively.

[4]Heuristic Methods for Activity Dependency Problems GitHub repository URL: https://github.com/CejkaLuk/heuristics-task-dependency
[5]Python 3.9.6 available at: https://www.python.org/downloads/release/python-396
[6]GNU Make webpage URL: https://www.gnu.org/software/make
[7]Nose2 testing framework webpage URL: https://docs.nose2.io/en/latest

- `make docs` (executed in `docs/`) - Generate the project documentation using *Sphinx*[8].
- `make clean` (executed in the repository root or in `docs/`) - Clean the generated files.

The core functions of SHM, PHM, and PHMDP are presented in Listings 1, 2, and 3, respectively.

```python
def solve(self):
"""Solves the activity dependency problem with resources."""

  self.cpm.solve()

  # Schedule activities
  for act in self.cpm.project.activities:
    self._schedule_activity(act)

  self.cpm.project.actual_end = self._get_project_actual_end()

def _schedule_activity(self, act: Activity):
  """Schedules an activity as soon as possible considering dependencies and ↩
      available resources."""

  # Get the time when all precessors of act have been completed
  time = self._get_predecessors_finished_time(act)

  while not act.is_scheduled():
    tentative_act_end = time + act.duration

    # Get the time when the available resources are exceeded between 'time' and ↩
        'tentative_act_end'
    time_resources_exceed = self._get_time_available_resources_exceeded(act, ↩
        time, tentative_act_end)

    # If the resources are not exceeded in the time frame, then schedule the ↩
        activity from 'time'
    if time_resources_exceed is None:
      self._schedule_activity_from(act, time)
    # Otherwise proceed to the next time when the resources are not exceeded
    else:
      time = time_resources_exceed + 1
```

Listing 1: The core functions of SHM: `solve()` and `_schedule_activity()`. Note that unimportant functions have been omitted for brevity.

```python
def solve(self):
  """Solves the activity dependency problem with resources and time reserves as ↩
      priorities."""

  self.cpm.solve()

  self._init_activity_priorities()

  time = 0
  while self._unfinished_activities_exist(time):
    # For PHM, this function does nothing, it serves as a placeholder so that ↩
        PHMDP can reuse the 'solve' function
    self._update_priorities(time)

    # Get activities that can be schedule from 'time'
```

[8]Sphinx documentation generator webpage URL: https://www.sphinx-doc.org/en/master

```
    viable_activities = self._get_viable_activities(time)

    if len(viable_activities) > 0:
      self._sort_by_priority_and_id(viable_activities)

      # Try to schedule all viable activities if they don't exceed the available↩
          resources
      for act in viable_activities:
        if not self._resources_exceeded(act, time, time + act.duration):
          self._schedule_activity_from(act, time)

    # Jump to the next time when an activity finishes as that is when more ↩
        activities can be scheduled
    time = self._get_time_next_act_finish(time)

  self.cpm.project.actual_end = self._get_project_actual_end()
```

Listing 2: The core function of PHM: `solve()`.

```
def _init_activity_priorities(self):
  """Override the method in PHM to avoid initializing activities without time.↩
      """

def _update_priorities(self, time: int):
  """Override the method in PHM to update the priorities dynamically."""
  for act in self.cpm.project.activities:
    act.priority = act.latest_start - time
```

Listing 3: The core functions of PHMDP. The PHMDP class inherits from PHM, therefore, it only overrides functions that deal with priorities.

4 Comparison

This section presents the results of the comparison of the heuristic methods presented in Section 2. While the heuristics were compared on thousands of problems, only the results of four are presented in this protocol due to trade secrets. Specifically, the methods were compared across four real-life scheduling problems obtained from a data-processing engine. The problems contained between 7 and 12 activities, and the resources corresponded to the number of CPU cores available to the engine. One problem was presented in Section 2, and three problems are presented in this section.

For each problem, the CPM network is presented. In this section, the edges representing jobs in the CPM networks have two values associated with them: computation time in seconds and CPU cores required.

4.1 Problem 1

The visualization of problem 1 as a CPM network is shown in Figure 6. For this problem, $r_{\max}$ was set to 7.

The timelines produced by SHM, PHM, and PHMDP are shown in Figures 7, 8, and 9, respectively. As can be seen from the figure, unexpectedly, PHM produced the best results for this

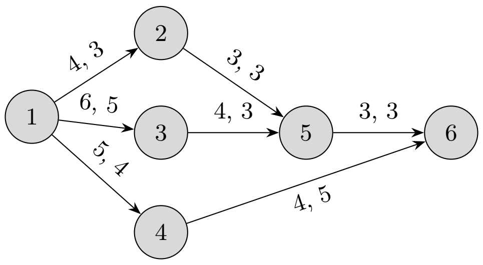

Figure 6: Visualization of problem 1 as a CPM network. Each weighted edge represents a job, its duration (in seconds), and the number of CPU cores required.

problem as it scheduled the jobs to be completed within 20 seconds. However, PHMDP was only one second behind. This is due to the dynamic priorities holding back certain jobs, such as 1-4, even though they need not have been.

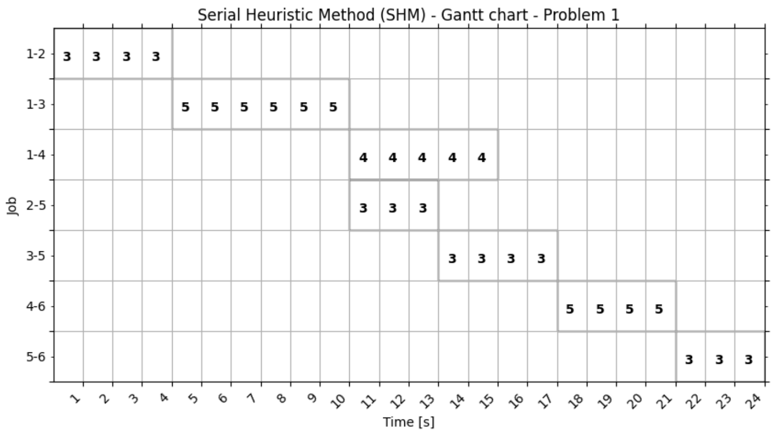

Figure 7: Timeline of the activities in problem 1 as scheduled by SHM. The vertical axis represents the job, while the horizontal axis displays the time in seconds. The numbers in the squares represent the CPU cores required by each job at a point in time.

4.2 Problem 2

The visualization of problem 2 as a CPM network is shown in Figure 10. For this problem, $r_{\max}$ was set to 8.

The timelines produced by SHM, PHM, and PHMDP are shown in Figures 11, 12, and 13, respectively. In the case of problem 2, PHMDP scheduled the jobs to be completed in the shortest time: 41 seconds. However, the slowest heuristic, SHM, was only three seconds behind.

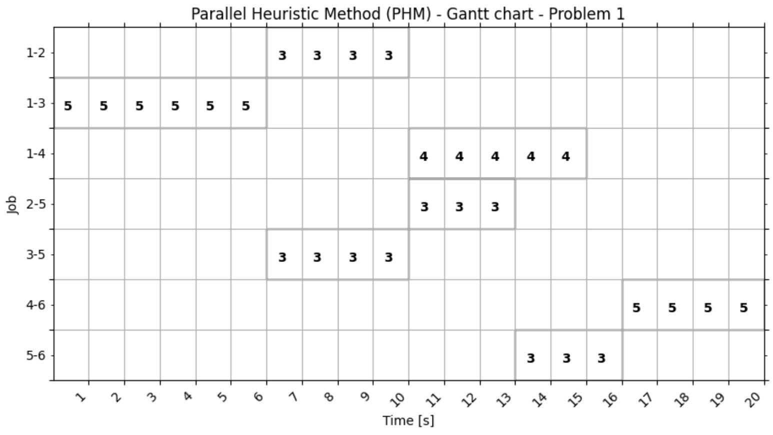

Figure 8: Timeline of the activities in problem 1 as scheduled by PHM.

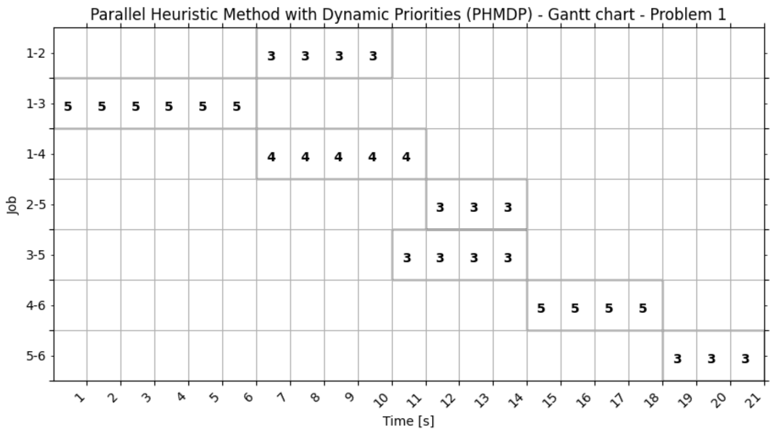

Figure 9: Timeline of the activities in problem 1 as scheduled by PHMDP.

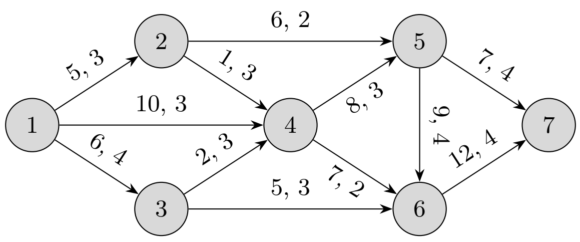

Figure 10: Visualization of problem 2 as a CPM network. Each weighted edge represents a job, its duration (in seconds), and the number of CPU cores required.

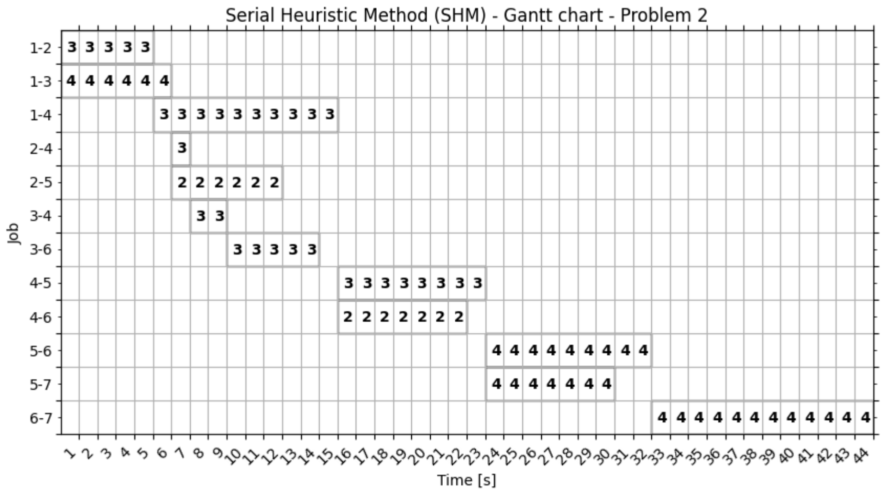
Figure 11: Timeline of the activities in problem 2 as scheduled by SHM. The vertical axis represents the job, while the horizontal axis displays the time in seconds. The numbers in the squares represent the CPU cores required by each job at a point in time.

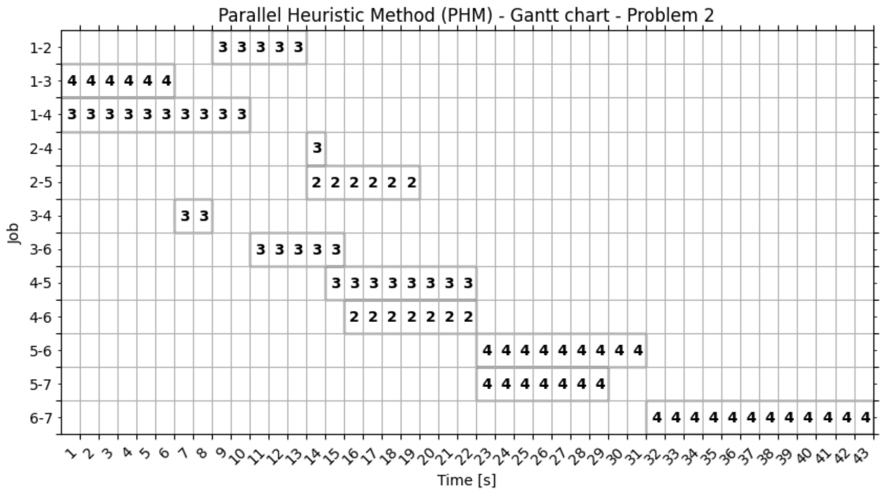
Figure 12: Timeline of the activities in problem 2 as scheduled by PHM.

4.3 Problem 3

The visualization of problem 3 as a CPM network is shown in Figure 14. For this problem, $r_{\max}$ was set to 6.

The timelines produced by SHM, PHM, and PHMDP are shown in Figures 15, 16, and 17, respectively. In the case of problem 3, PHM and PHDMP tied in scheduling the jobs to complete in the shortest time: 17 seconds. However, SHM was only two seconds behind.

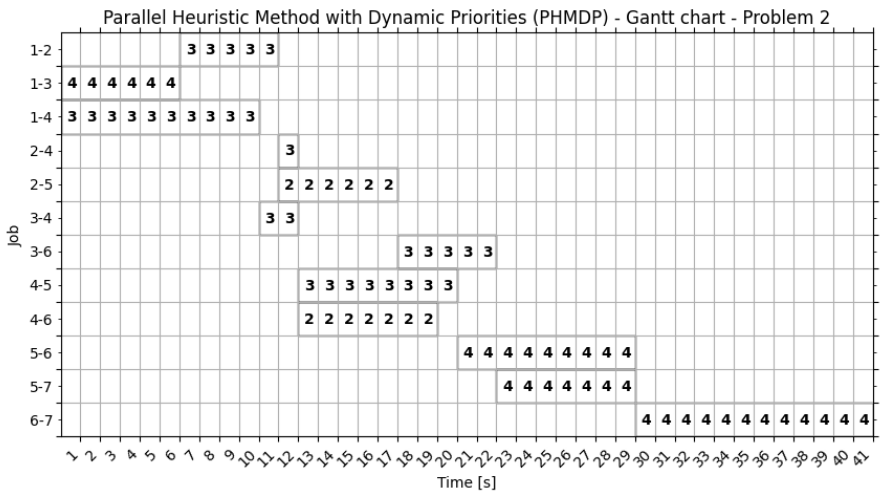

Figure 13: Timeline of the activities in problem 2 as scheduled by PHMDP.

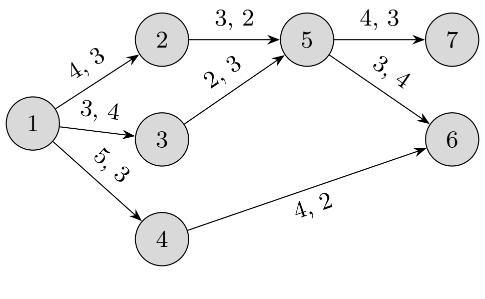

Figure 14: Visualization of problem 3 as a CPM network. Each weighted edge represents a job, its duration (in seconds), and the number of CPU cores required.

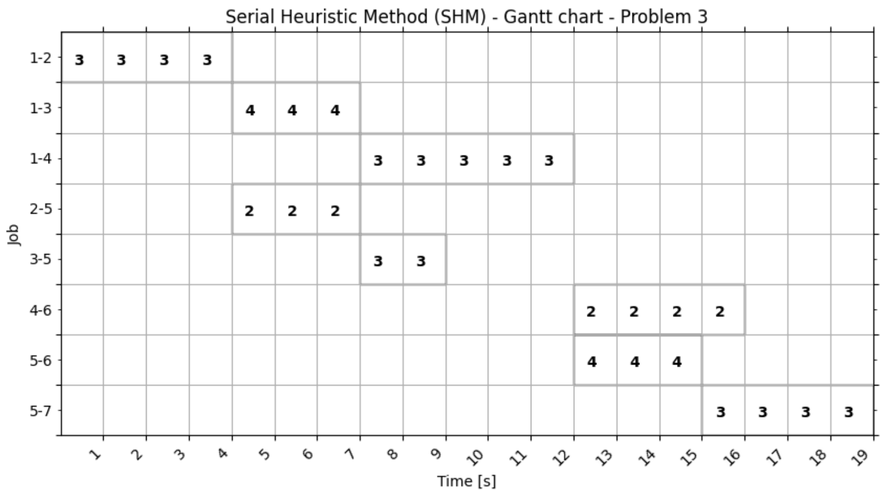

Figure 15: Timeline of the activities in problem 3 as scheduled by SHM. The vertical axis represents the job, while the horizontal axis displays the time in seconds. The numbers in the squares represent the CPU cores required by each job at a point in time.

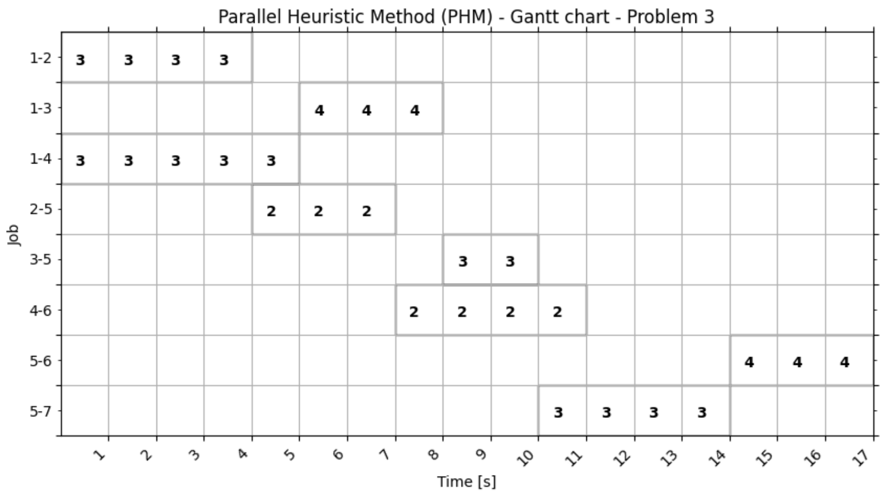

Figure 16: Timeline of the activities in problem 3 as scheduled by PHM.

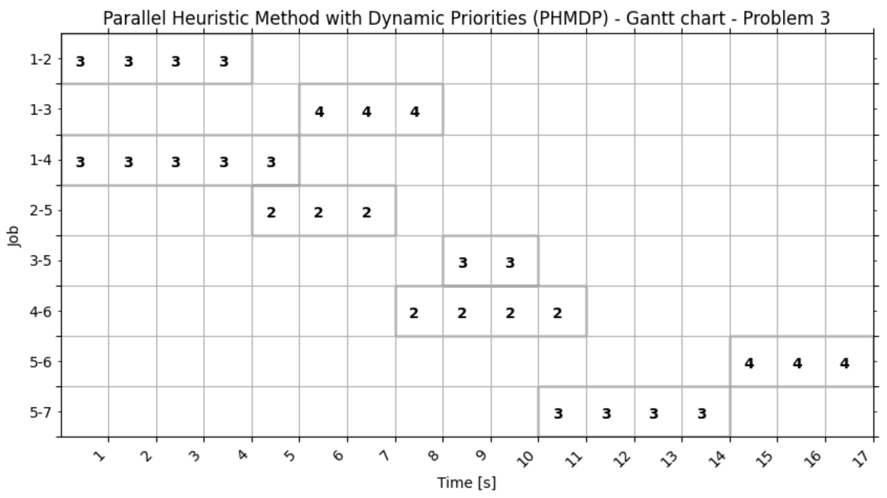

Figure 17: Timeline of the activities in problem 3 as scheduled by PHMDP.

5 Conclusion

In summary, this protocol presents the theory behind a selection of heuristic methods that can be used to determine the earliest possible completion of a set of jobs while adhering to certain limitations. The limitations include dependencies between the jobs and the resources they require. The heuristic methods were implemented in a project using Python and subsequently compared on a small exploratory sample of real-life problems observed on a data-processing engine.

Taking into consideration the small sample size, the results presented cannot be used to definitively claim one heuristic outperforms another. However, it can be stated that in terms of the entire benchmark comprising thousands of problems, PHM and PHMDP outperformed SHM in the majority of cases. Unfortunately, the specific data cannot be presented due to trade secrets. It is noteworthy that, theoretically, while SHM produces suboptimal timelines compared to PHM and PHMDP, it may be the preferred heuristic to use as its implementation is faster than the remaining heuristics. For example, if multiple jobs were to be submitted every second, the timeline may have to be recomputed frequently. Therefore, it is possible that the time spent computing the timeline may be greater than the time gained by using PHM or PHMDP over SHM. Furthermore, theoretically, the performance of the implementations of PHM and PHMDP could further decrease with the presence of thousands of jobs.

In terms of future work, the heuristics ought to be compared on a different set of problems. Furthermore, their implementations ought to be optimized and perhaps implemented using a more performance-oriented language, such as C++.

References

[1] KELLEY, J. E.; WALKER, M. R. Critical-path planning and scheduling. In: *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference on - IRE-AIEE-ACM '59 (Eastern)* [online]. New York, New York, USA: ACM Press, 1959, pp. 160–173 [visited on 2023-07-19]. Available from DOI: 10.1145/1460299.1460318.

[2] FIALA, P. *Řízení projektů*. 2nd ed. Praha: Oeconomica, 2008. ISBN 9788024514130.