香港城市大學
City University of Hong Kong

Department of Computer Science

BSCCS Final Year Project Report
2021-2022

21CS094

Scheduling with calibration

(Volume 1 of 1)

Student Name : **BASIC Andela**

Student No. : **55594403**

Programme Code : **BSCEGU4**

For Official Use Only

Supervisor : **Prof LI, Minming**

1st Reader :

2nd Reader :

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Scheduling with calibration

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name: BASIC Andela Signature: Andela Basic

Student ID: 55594403 Date: March 31, 2022

Scheduling with calibration

Scheduling with many calibration types and probing

Abstract

Scheduling problem with calibration represents a task of scheduling jobs on a single or multiple machines that need occasional regulation, while scheduling with uncertainty refers to a scheduling problem that has one or many components with unknown values. Information about these values can be obtained by probing. Bender et al. (2013) and Angel et al. (2017) studied various settings of problems in the area of scheduling with calibration, while Dürr et al. (2020) studied the problem of scheduling with uncertainty related to job processing times. The contribution described in this paper combines the consideration of the said two problems. We examine a scheduling problem with many calibrations of uncertain lengths. While the set from which the calibration length is drawn is known, the exact value is unknown. Therefore, the concept of probing is introduced to determine the exact calibration length. The problem of finding the optimal schedule that minimizes the total cost is thus reduced to choosing the intervals to probe. We provide a pseudo-polynomial algorithm that approximates the optimal solution. Firstly, we provide a dynamic programming solution when job lengths and calibration lengths take two values. Afterwards, the problem is generalised so that the job lengths and calibration lengths can take many discrete values. Finally, we implement this algorithm in Python and compare its performance to the algorithm that randomly probes intervals. Our algorithm outperforms the random scheduler on all 121 test cases. In addition, we provide an alternative algorithm, that further decreases the cost of the schedule by assuming the second shortest interval instead of the shortest interval in the case of not probing. Further dimensions to explore could be related to describing calibration length as a continuous random variable and allowing jobs to have release times and deadlines.

Acknowledgements

I would like to express my sincere gratitude to my mentor Dr. Minming Li, and to Dr. Christoph Dürr for their support and valuable feedback during the entire project lifetime. Specifically, the weekly analysis of various research papers and suggestions by Dr. Li were crucial for proposing the exact problem definition and algorithm. In addition, the detailed analysis of the initial implementation of an algorithm conducted by Dr. Dürr exposed a fundamental mistake in the first version of the code. Finally, I believe that this project not only finalizes my studies at City University of Hong Kong, but also resembles a valuable insight into professional life in academia. Therefore, I would like to thank City University of Hong Kong for giving me an opportunity to explore this side of computer science via this project.

Contents

1 Introduction

1.1 Background and Motivation

1.1.1 Scheduling with calibration

Scheduling with calibration represents a group of problems that originate from an Integrated Stockpile Evaluation (ISE) program. ISE is a program introduced by Sandia National Laboratories aiming to ensure the better quality of the US nuclear weapons stockpile. Nuclear weapons are periodically tested on machines that need occasional adjustment. The goal is to maximize the number of performed tests for a given amount of money or to minimize the number of calibrations. This problem has been defined in many ways, from basic to more general cases. Bender et al. studied the problem of scheduling jobs with unit processing times and proposed several algorithms(Bender et al. 2013). In addition, Angel et al. propose an optimal polynomial algorithm when jobs have arbitrary length and preemption is allowed (Angel et al. 2017). Also, they showed that in the case of arbitrary calibration interval lengths and arbitrary test lengths, the problem becomes NP-hard, even if we allow the preemption of the tests. Therefore, they focus on exploring the case with many calibrations, unit-time test lengths and non-instantaneous calibrations, for which they provide a polynomial-time dynamic programming solution.

1.1.2 Scheduling with uncertainty

Scheduling with uncertainty has been explored on many occasions and via different approaches. These approaches usually belong to the areas of "online optimization, stochastic optimization, and robust optimization" (Dürr et al. 2020). Uncertainty refers to some components of the problem that are unknown prior to scheduling. In scheduling problems, uncertainty could be related to job execution times, or machine calibration intervals. An uncertain component cannot be precisely determined before establishing a schedule (Verderame et al. 2010). However, many applications of scheduling problems that are encountered in the real-life present a possibility of obtaining additional information about these uncertain variables during execution at additional cost (Dürr et al. 2020). This information is usually related to the probability distribution of an uncertain component, which can be discrete or continuous (Floudas and Lin 2005). Therefore, the purpose of scheduling algorithms is to allocate the limited resources to the processing tasks in an optimal manner based on problem definition and objective (Li and Ierapetritou 2008). In addition, a correlation could be found between scheduling with uncertainty and bin-packing with uncertainty, where the sizes of bins are unknown before the bin-packing has taken place (Basu Roy et al. 2019).

1.1.3 Existing models in the field of scheduling with uncertainty

Most of the research in the field of scheduling with uncertainty focuses on determining an optimal schedule when the execution time of jobs is uncertain prior to scheduling. The apparent motivation behind these types of problems corresponds to applying a test before a task execution which could reduce the execution time of a task. For example, code optimizer, file compressor and job execution in different modes (Dürr et al. 2020). Despite the various applications, it can be observed that addressing only the processing time as an uncertain parameter fails to model some real-life problems. For example, a machine on which the jobs are performed needs to be maintained to function properly. This issue is denoted as scheduling with calibration, where the calibration interval corresponds to the time frame during which the machine is in a good state. The analogy between this issue and the case where the job execution time is uncertain is existent, but not obvious. Therefore, there is a need for extending the current solutions, so that the uncertainty related to calibration intervals is properly addressed.

1.2 Problem definition

1.2.1 Overview

The problem definition may vary based on the objectives and parameters. In this project, we consider the scheduling problem with many types of calibration. The calibration lengths are unknown, therefore, probing is introduced as a way to gain more knowledge at additional cost. The problem is how to schedule jobs such as to minimize total cost. Solving this problem is thus reduced to determining when would it be suitable to do the probing.

Problem 1 (Scheduling with many calibration types with probing) *We are given n jobs, every job j has a given processing time p_j, that is either short or long. A machine is in a calibrated state for some unknown time t, which is uniformly drawn from the set of possible calibrations T, which is known. If a job is completed outside of the calibration, it needs to be repeated. Once the machine is not in a calibrated state, it needs to be re-calibrated. Probing can be done along with the calibration to determine the exact calibration time. Costs of calibration and probing are given by c_c and c_p,respectively. Assuming that calibration and probing take no time, consider the objective of minimizing the total cost for completing all jobs.*

1.2.2 Generalisations of the base case

The main idea of this paper is to first observe the simpler versions of the problem, i.e. one value for processing times and only two types of calibration intervals. Afterwards, we will gradually generalise those cases in the following manner:

- Calibration lengths take many values
- Job processing times take many values

1.3 Objectives

The main objective is to design and implement an algorithm that can provide the schedule with minimal cost, for a problem defined above. This algorithm should successfully determine intervals for probing. A goal is to find a compromise between executing jobs with and without probing (Levi, Magnanti, and Shaposhnik 2018). The analysis will first observe the simple cases. Later, it would proceed to more complicated versions of the problem. Finally, we implement the solution in Python.
We consider following algorithms.

- Deterministic algorithms
- Randomized algorithms

2 Literature review

2.1 Optimal solution with uncertain parameters

Lin et al. (Lin, Janak, and Floudas 2004) explained why a solution to a scheduling problem with initial values is not optimal if those values were uncertain. Their work addresses the importance of alternative solutions for scheduling problems with uncertain components by explaining the following example which was introduced in 1993 by Kondili, Pantelides and Sargent. Two products are produced from three feeds according to certain standards. These standards are dependent on some given. The objective is to maximize the sales of these two products in the production time frame of 12h. It can be observed that slight changes in nominal values of the data yield quite a different optimal schedule. It is explicitly shown that if processing times of tasks were increased by only 0.1% the given schedule would no longer be optimal, as the optimal solution with initial processing times yields an optimal profit of 3639, while the optimal profit when the processing times are increased is 3265 (Kondili, Pantelides, and Sargent 1993). This result stresses the need for alternative approaches when there exists an uncertain component of a problem. Therefore, the results of scheduling problems with calibrations that have given lengths cannot be used directly for solving problems in which lengths become uncertain.

2.2 Scheduling with uncertain job execution times

An example of a problem statement and objective for a scheduling problem where the uncertain information is the execution time of a job is provided by Dürr et al. (Dürr et al. 2020) as follows.

Problem 2 (Scheduling with uncertain job processing times) *We are given n jobs to process on a single machine. For each job i a maximum processing time $\hat{p}_i$ is given. A job i can be executed untested, in which case its execution would take time $\hat{p}_i$, or tested (testing takes time 1), which would reduce the execution time to the arbitrary p_i, where $0 \leq p_i \leq \hat{p}_i$. Initially, only $\hat{p}_i$ is given for each job, while p_i is unknown unless the job is tested. At every moment a machine can either execute or test a job. If the completion time of a job i is given by C_i, consider the objective of minimizing the total completion time $\sum C_i$. It is important to note that the goal is* not *to minimize the objective value in the worst case because otherwise, it would be optimal to execute all jobs untested. The goal is to minimize the competitive ratio, which is the objective value normalized by the offline optimum, which executes every job i untested, iff $p_i + 1 > \hat{p}_i$. In other words, the optimal scheduler knows which jobs are worth being tested.*

An example of the application of described problem is a code optimizer that can be applied before running a program. Unless a code optimizer is used, the execution time of the code will be given by the upper limit. Another application is a file compressor for transmitting files, which could potentially reduce the size of the file.

2.3 Scheduling with many calibration types

Scheduling with calibration has been studied on various occasions as to propose a model for optimizing the performance of the ISE program introduced by Sandia National Laboratories. Angel et al. (Angel et al. 2017) introduced several generalisations of an initial model for the ISA program proposed by Bender et al. (Bender et al. 2013). Firstly, they studied the case of arbitrary processing times on a machine with a single calibration type, for which an optimal polynomial algorithm is introduced. Secondly, they studied the case of arbitrary job processing times and many calibration types, which have arbitrary lengths and costs. They showed that the second problem is NP-hard, even when the preemption of the jobs is allowed. Finally, they formed a more specific case of the last problem by considering many calibration types that are given as input, unit job processing times and non-instantaneous calibrations. This means that after calibration, the machine needs time t_a, activation length, to become ready to execute jobs. They provide a polynomial running time dynamic program that optimally solves this variant of a problem.

Definition *Preemptive scheduling is a subset of scheduling problems that allow temporal interruption of processing tasks with an intention of resuming them at a later point in time.*

Angel et al. (Angel et al. 2017) proved that the problem of minimizing the number of calibrations when the job execution times and calibration interval lengths can take many arbitrary values is NP-hard. This holds even in the case job can be preempted. The proof is obtained by reduction from the Unbounded Subset Sum problem, which is NP-hard (Angel et al. 2017).

2.4 Pandora's box problem

2.4.1 Standard formulation of Pandora's box problem

Many optimization problems that consider the uncertain input also consider the concept of probing, i.e. attaining the information about the uncertain input data at an additional monetary or computational cost. Furthermore, information that an algorithm obtains about the input data affects the further manner in which it is executed, and the problem has a decision making aspect. An example of such a problem is Pandora's box problem (Chawla et al. 2020). Weitzman (Weitzman 1979) formulated Pandora's box problem in the following manner.

Problem 3 (Pandora's Box Problem) *We are given n closed boxes, each carrying the potential reward x_i defined by a probability distribution $F_i(x_i)$, which is independent of probability distributions associated with other boxes. Each box is associated with the opening cost c_i. At every step Pandora can choose a box and open it, or she can decide to finish the process. Her net profit is the maximum reward among the opened boxes, minus the total opening cost. The goal is to maximize her expected net profit.*

Weitzman (Weitzman 1979) provided an optimal algorithm for which he firstly computes a so-called Gittin's index of each box based on the reward probability distribution. Afterwards, the boxes are opened in decreasing order of these indices, until a box that is associated with a reward greater than all of the remaining indices is reached. The solution to Pandora's box problem can be applied in the areas of economic search problems and sequential research strategy problems.

2.4.2 Pandora's box problem with correlations

The main limitation of the model introduced by Weitzman is the assumption that the costs of opening the boxes are independent. Indeed, this assumption fails to model many real-world problems such as online shopping for a specific item. Specifically, if we are searching for a specific item across various websites, with an aim to find the cheapest one for the shortest time, it is highly possible that prices across these websites are correlated (Chawla et al. 2020). Chawla et al. (Chawla et al. 2020) addressed this limitation, by assuming the joint distribution of the rewards. Algorithms for this group of problems should not only determine when to stop and return the best result found so far but also an order of probing random variables. It is shown that fully adaptive strategies cannot be approximated. Thus, the focus is shifted to the simplification of the problem concerning partially adaptive strategies that probe random variables in a predetermined order, but decide whether to terminate the search based on the observed results.

2.5 Bibliography on bin-packing with random capacities

2.5.1 Standard bin-packing problem

The problem of bin-packing has been studied in the field of computer science since the 1970s Seiden 2002. The standard formulation of the bin-packing problem is as follows. We are given n items of sizes $s_1, \ldots, s_n$, and need to pack them into a few possible unit capacity bins. This problem is NP-hard, but there are several approximation algorithms for it. As suggested by Gilmore and Gomory in 1961, and later on developed by Vence in 1998, linear programming methods can be used for exactly solving this problem (Kang and Park 2003). The linear programming solution takes too much time in practice. Therefore, various heuristics algorithms have been developed. Seiden (2002) summarized some asymptotic approximation algorithms as follows. The First Fit algorithm is investigated by Ullman in 1971. The Next Fit algorithm, investigated by Johnson in 1974, has a performance ratio of 2. Also, it has been shown that the First Fit algorithm has a performance of 17/10. A Series of papers have been published from 1970 to 2000, aiming at providing better approximations of the standard bin-packing problem (Karmarkar and Karp 1982). In 1990, the Revised First Fit algorithm was presented, with a competitive ratio of 5/3. Also, it has been shown that the performance in terms of a competitive ratio of all online algorithms is not less than 3/2. This lower bound has been improved in work done afterwards (Seiden 2002). Variable sized bin-packing problem is closely related to the classic bin-packing problem, and its formulation can be related to the scheduling with many calibration types.

2.5.2 Variable sized bin-packing problem

Variable sized bin-packing problem is a generalisation of a standard formation, where the bins are described by the set T of m bins. Each bin is associated with capacity (that is equal to the cost), and we have infinite many bins of each type. In this problem, we are given a set L of n items and asked to assign them to bins in a way that would minimize the total cost (Kang and Park 2003). In the general case, the variable-sized bin-packing problem is NP-hard. The problem is solvable in an easier manner if some divisibility constraints are imposed on the sets L and T (Kang and Park 2003). For example, there are proposed solutions for the following cases. Firstly, if divisibility constraint is imposed on the sizes of bins and items. Secondly, if only bins have divisible sizes, and finally, if the bins have non-divisible sizes (Kang and Park 2003). Kang and Park (2003) provide two algorithms, "Iterative First-Fit Decreasing" and "Iterative Best-Fit Decreasing" that are based on well-known algorithms for solving a one-size bin-packing problem - First-Fit Decreasing (FFD) algorithm and Best-Fit Decreasing (BFD) algorithm.

3 Design and analysis of dynamic programming solutions

3.1 Dynamic Program - first version

3.1.1 Overview

We propose a dynamic program as a solution to the problem described in 1.2.1. In addition, we consider the generalizations described in 1.2.2. Following algorithms minimize the expected cost of the schedule, by choosing between probing and not probing the current interval. If we probe the current interval, the cost is calculated as $c_1 = C + c$, and the value of its length is one of k uniformly distributed values. Otherwise, the cost is calculated as $c_2 = C$, and its value is assumed to be the shortest among k uniformly distributed values, $k = 2, 3, ...$ These algorithms are the result of the personal communication and discussion with Dr. Minming Li.

3.1.2 Algorithms

1. **Two uniformly distributed values for calibration intervals $\{\mathbf{t_1}, \mathbf{t_2}\}$, and $\mathbf{t_1} < \mathbf{t_2}$, $\mathbf{n_1}$ and $\mathbf{n_2}$ jobs that have processing times $\mathbf{p_1}$ and $\mathbf{p_2}$, where $\mathbf{p_1} < \mathbf{p_2}$**
   Algorithm:

$$C(n_1, n_2) = \min\,(c_1 + \sum_{i=1}^{2} \frac{1}{2} \min_{1 \leqslant j \leqslant c_i} C(n_1 - \alpha_{i,j}, n_2 - \beta_{i,j}), c_2 + \min_{1 \leqslant j \leqslant c_1} C(n_1 - \alpha_{1,j}, n_2 - \beta_{1,j})),$$

   where $\min 0 <= j <= c_i$ loops through all maximal coverings of an interval of type i, that are formed from the remaining jobs. Namely, if we added another job of the either length to the covering $(\alpha_{i,j}, \beta_{i,j})$, the total execution time would be greater than the length of an interval i. Also, c_1, correspond to the coverings of the shortest interval t_1.

2. **k uniformly distributed values for calibration intervals $\mathbf{t_1}, \mathbf{t_2}, ...\mathbf{t_k}$, and $\mathbf{t_1} < \mathbf{t_2} < ... < \mathbf{t_k}$, $\mathbf{n_1}$ and $\mathbf{n_2}$ jobs that have processing times $\mathbf{p_1}$ and $\mathbf{p_2}$, and $\mathbf{p_1} < \mathbf{p_2}$**
   Algorithm:

$$C(n_1, n_2) = \min(c_1 + \sum_{i=1}^{k} \frac{1}{k} \min_{1 \leqslant j \leqslant c_i} C(n_1 - \alpha_{i,j}, n_2 - \beta_{i,j}), c_2 + \min_{1 \leqslant j \leqslant c_1} C(n_1 - \alpha_{1,j}, n_2 - \beta_{1,j})),$$

   where $\min 0 <= j <= c_i$ loops through all maximal coverings of an interval of type i, that are formed from the remaining jobs. Namely, if we added another job of the either length to the covering $(\alpha_{i,j}, \beta_{i,j})$, the total execution time would be greater than the length of an interval i. Also, c_1, correspond to the coverings of the shortest interval t_1.

3. **Two uniformly distributed values for calibration intervals $\{\mathbf{t_1}, \mathbf{t_2}\}$, and $\mathbf{t_1} < \mathbf{t_2}$, $\mathbf{n_2}$,...,$\mathbf{n_m}$ jobs that have processing times $\mathbf{p_1}$, $\mathbf{p_2}$,..., $\mathbf{p_m}$, and $\mathbf{p_1} < \mathbf{p_2} < ... < \mathbf{p_m}$**
   Algorithm:

$$C(n_1, n_2, ..., n_m) = \min(c_1 + \sum_{i=1}^{2} \frac{1}{2} \min_{1 \leqslant j \leqslant c_i} C(n_1 - n_{1,i,j}, n_2 - n_{2,i,j}, ..., n_{m,i,j})$$
$$c_2 + \min_{1 \leqslant j \leqslant c_1} C(n_1 - n_{1,1,j}, n_2 - n_{2,1,j}, ..., n_{m,1,j})),$$

   where $\min 0 <= j <= c_i$ loops through all maximal coverings of an interval i, that are formed from the remaining jobs.

Equivalently, if we added another job of the either length to the covering $(n_{1,i,j}, n_{2,i,j}, ..., n_{m,i,j})$, the total execution time would be greater than the length of an interval i. Also, c_1, corresponds to the coverings of the shorter interval t_1.

4. **k uniformly distributed values for calibration intervals $\mathbf{t_1}, \mathbf{t_2}, ...\mathbf{t_k}$, and $\mathbf{t_1} < \mathbf{t_2} < ... < \mathbf{t_k}$, $\mathbf{n_1}$, $\mathbf{n_2}$,...,$\mathbf{n_m}$ jobs that have processing times $\mathbf{p_1}$, $\mathbf{p_2}$,..., $\mathbf{p_m}$, and $\mathbf{p_1} < \mathbf{p_2} < ... < \mathbf{p_m}$**
Algorithm:

$$C(n_1, n_2, ..., n_m) = \min(c_1 + \sum_{i=1}^{k} \min_{1 \leqslant j \leqslant c_i} C(n_1 - n_{1,i,j}, n_2 - n_{2,i,j}, ..., n_{m,i,j})$$
$$c_2 + \min_{1 \leqslant j \leqslant c_1} C(n_1 - n_{1,1,j}, n_2 - n_{2,1,j}, ..., n_{m,1,j})),$$

where $\min 0 <= j <= c_i$ loops through all maximal coverings of an interval i, that are formed from the remaining jobs. Equivalently, if we added another job of the either length to the covering $(n_{1,i,j}, n_{2,i,j}, ..., n_{m,i,j})$, the total execution time would be greater than the length of an interval i. Also, c_1, corresponds to the coverings of the shortest interval t_1.

3.1.3 Performance analysis

In this subsection we will analyse the performance of the algorithms presented in 3.1.2. We will derive the time complexity for each case presented before. We assume that $p_1 < p_2 < p_3 ... < p_m$ and $t_1 < t_2 < t_3 ... < t_k$

1. **Two job lengths and many calibration types**
Firstly, we bound the coverings c_i of calibration intervals t_i. We loop through all non-negative $\beta_{i,j}$ values such that $\beta_{i,j} * p_2 \leqslant t_2$, and fill the remaining space with as many shorter jobs as possible. Therefore, for each calibration interval i, $1 \leqslant i \leqslant k$, computation of $c_i(n_1, n_2)$ takes $O(\frac{t_2}{p_2})$ time. Since there are $O(n_1 \cdot n_2)$ values, we obtain the final complexity $O(n_1 \cdot n_2 \cdot t_2 \cdot \frac{1}{p_2})$

2. **Many job lengths and many calibration types**
For each t_i, we firstly bound the coverings c_i similarly to the previous case. We have the following equation $p_1 \cdot n_{1,i,j} + p_2 \cdot n_{2,i,j} + ... + p_m \cdot n_{m,i,j} \leqslant t_i$, where $2 \leqslant t \leqslant m$, $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant c_i$. Each $n_{t,i,j}$ is bounded with $0 \leqslant n_{t,i,j} \leqslant \frac{t_i}{p_j}$. In other words, each interval t_i can have from 0 to $\frac{t_i}{p_j}$ jobs of type j, for all m jobs. Also, we fill the remaining space with as many jobs of type 1 as possible. Multiplying these values gives the following complexity. For each calibration interval t_i, computation of $c_i(n_1, n_2, ..., n_m)$ takes $O(t_i^{m-1} \cdot (\prod_{j=2}^{m} p_j)^{-1})$ time. Since there are $O(n_1 \cdot n_2 ... \cdot n_m)$ values, we obtain the final complexity $O(p_1 \cdot t_k^{m-1} \cdot \prod_{j=1}^{m} \frac{n_j}{p_j})$, because the term with the longest calibration is dominant.

It is important to notice that this program is pseudo-polynomial. This complexity is good enough, since the problem is NP-hard.

3.2 Dynamic Program - second version

3.2.1 Overview

The alternative version of the dynamic program is proposed by Dr. Christoph Dürr, and it is summarized below. We consider 2 possible calibration lengths $T_1 < T_2$, and n jobs for scheduling in the order $1, \ldots, n$. Every job j has processing time either p or $q > p$, i.e. $p_j \in \{p, q\}$. We assume that $q < T_1$. Calibration cost is C, and probing cost is c. Therefore, this corresponds to the first problem of the previous section.

At every moment we have a short jobs to schedule and b long jobs for execution. Let $A(a, b)$ be the optimal expected cost for scheduling those jobs, such that we need to open calibration intervals for all these jobs. For the base case we have $A(0, 0) = 0$, because there is no job to be scheduled. The cases $A(0, b)$ and $A(a, 0)$ are trivial, i.e, $A(0, b) = A(a, 0) = C$. So assume that $a, b \geq 1$.

Let $B(a, b, k)$ be the minimum expected cost to schedule the jobs, knowing that there is a probed calibration interval of type k, which is already paid for and not yet used. We have $B(a, b, k) = \min A(a', b')$, where the minimum is taken over all $0 \leq a' \leq a$, $0 \leq b' \leq b$ such that $(a - a')p + (b - b')q \leq T_k$. B optimizes the decision on which jobs to execute, by considering all calibration types.

Similarly let $C(a, b)$ be the minimum expected cost to schedule the jobs, knowing that there is an un-probed calibration interval, which is already paid for and not yet used. Here we need to decide on a set of jobs to be scheduled within time T_1, the shortest interval length. We have $C(a, b) = \min A(a', b')$, where the minimum is taken over $0 \leq a' \leq a$, $0 \leq b' \leq b$, such that $(a - a')p + (b - b')q \leq T_1$.

3.2.2 Algorithm

We combine B and C in order to make the right choice in A, which gives us the following dynamic program. Note that c_1 and c_2 are defined in 3.1.1. So we have

$$A(a, b) = \min \begin{cases} c_1 + \sum_{k=1}^{2} \frac{1}{2} B(a, b, k) \\ \\ c_2 + C(a, b). \end{cases}$$

3.2.3 Performance analysis

We have $O(n^2)$ variables, each is the minimization over $O(n^2)$ options. This gives an overall running time of $O(n^4)$, which makes it practical only for a small number of jobs, e.g. 200.

3.3 Comparing two algorithms

The main difference between the two algorithms is the formulation of the objective function. The second version could be viewed as a more concise way of formulating the first version. It considers the problem of finding the maximal packs of an interval via introducing B and C. In the first version, the objective function directly considers minimization over all maximal packs of all calibrations. Practically, we would need to minimize over all maximal packs of all calibrations in both algorithms.

4 Software project plan

4.1 Resource estimation

The resources required for an implementation of the dynamic program proposed in section 3.1 are minimal. A personal computer is sufficient. However, the most challenging part of this project is designing the scheduling algorithm. The reason for this is that this project combines two areas of research on scheduling problems. Therefore, both areas were examined independently prior to finalising problem definition.

4.2 Risk plan

4.2.1 Performance issues

As stated in the section 3.1.3, the time complexity of the dynamic program is pseudo-polynomial. Thus, the performance of an algorithm is sensitive to the data size. As the schedule is implemented as a recursive function, the lower performance could be related to repetitive calculations Erickson 2019. Ericson (2019) argues that memorization techniques are essential for optimal performance of dynamic programs. This can be achieved by using `functools` module from the Python's standard library (Christoph Dürr, pc). Python 3.2 introduces the decorator `functools.lru_cache()`, that enhances the performance by using the Least Recently Used (LRU) strategy for avoiding the calculations that are expected to be the same (*What's new in python 3.2* n.d.). In that sense, to achieve significantly better performance, it is sufficient to add the following code `@lru_cache(maxsize=None)` in front of the recursive functions and tune parameters such as they are hashable.

4.2.2 Management and technical constraints

There are very little constraints related to the implementation to the dynamic program. However, there are certain constraints related to the design of an algorithm. These constraints are mainly related to the extended research that could be conducted to apply the results of related work to designing the more complex algorithm, that could achieve even better performance. The performance of our algorithm is compared to the algorithm that randomly decides whether to probe intervals, and it shown that it performs better.
Management of the project is conducted on a weekly basis via emails and Zoom calls. Important parts of these discussions were analysis of the related works and the extraction of the important ideas. These ideas are applied to the problem at this stage, and some of them are inspiration for the future work, discussed in section 7.3.

4.2.3 Monitoring the status of a project

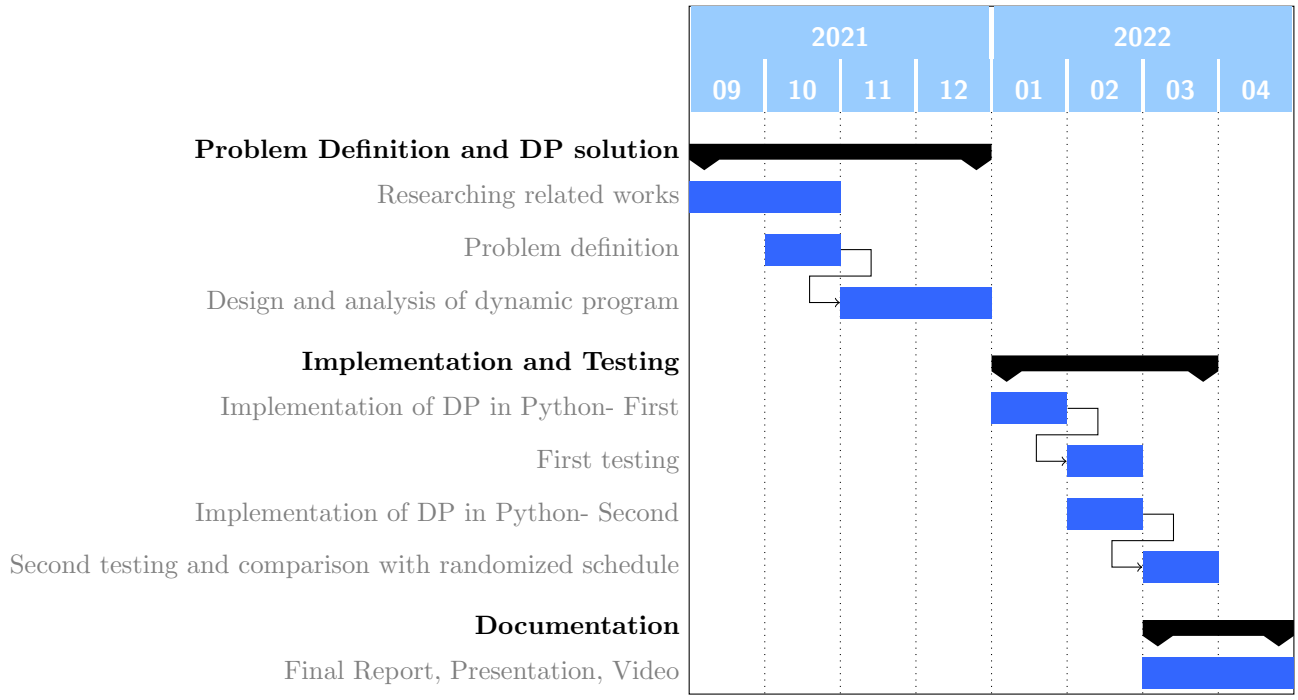
The status of a project is monitored on the weekly meetings. The project can be divided into three main stages: problem definition and theoretical framework of a solution, implementation and testing of an algorithm, and documentation. The lifetime of a project is throughout the 2021/22 academic year. The first stage was mainly related to researching and extracting the existing ideas from the related works. Also, we precisely defined the problem at this stage. Scheduling with probing and calibration is not a unique problem, and there could be many areas of exploration. Therefore, the was the most challenging stage, as it includes some idle time in terms of progress. Still, the ideas that were not used directly were useful for proposing extensions to this project. Overall, there are no major issues related to the project management, as it is conveniently scheduled via online communication.

4.3 Requirements analysis

4.3.1 Work breakdown structure

1. Project Management
   (a) Progress evaluation
      i. Weekly meetings
      ii. Email communication
   (b) Project planning
      i. Define requirements
      ii. Define stages of a project
      iii. Identify resources required at each stage
2. Problem Definition
   (a) Examine different related works
   (b) Select the most relevant ideas
   (c) Problem definition - base case and generalizations
3. Design and analysis of an algorithm
   (a) Dynamic programming solution
      i. Solve for base case
      ii. Solve for generalised cases
   (b) Performance analysis - time complexity
   (c) Alternative notation
4. Implementation and Testing
      i. Implementation
      ii. Implementing maximal coverings of an interval
      iii. Implementing schedule that minimizes expected cost
      iv. Implementing random schedule
      v. Optimizing performance with memorization
      vi. Displaying the schedule
      vii. Analysing the probing rank for the base case
   (a) Testing
      i. Unit testing
      ii. Designing test cases
      iii. Comparing two schedules
5. Documentation
   (a) Interim reports
   (b) Final report
   (c) Presentation
   (d) Video

4.3.2 Gantt chart



4.3.3 Comments

As stated in section 4.2.3 and displayed in the Gantt chart, this project can be divided into three main stages - Problem definition and theoretical framework of a solution, implementation and testing of an algorithm, and documentation. The most time-consuming aspect of this project was research and problem definition. The reason behind this is that the family of scheduling problems that have the uncertainty parameter linked to the calibration type are not studied extensively as some other family of scheduling problems, such as scheduling with uncertain execution times of the jobs (Dürr et al. 2020). Therefore, the important part of this project was examining the correlation between our problem and existing problems. Finally, we defined the base case as in 1.2.1 and studied the generalizations as in 1.2.2. Secondly, the theoretical framework for solving these problems is provided, as a pseudo-polynomial running time dynamic program. Afterwards, the algorithm is implemented and tested in Python. The first implementation contained an important oversight related to maximal coverings of the intervals, that was discovered after testing it. More details about the first version are provided in section 6.4. The second and final version of the implementation is provided and discussed in this report. Its performance is tested by comparing it to the randomized algorithm, and the details are provided in section 6.2. Finally, the last stage requires producing interim reports, a final report, a presentation and a video.

5 Implementation and interpretation of dynamic programs

5.1 Tools

Tools	
Tool	Descrition
Computer	MS Windows 10, 16.0 GB RAM
Jupyter Notebook	Python IDE
Overleaf	Online LaTeX editor
MS PowePoint	Microsoft slide-show program

5.2 Software functions

5.2.1 Overview

The software should provide the schedule that decides on whether to probe an interval or not and display its cost and probing decisions. The implementation of an algorithm is provided by the `schedule()` function, which recursively minimizes the expected cost over maximal coverings of all calibrations. The function `fit_max()` precomputes the maximal coverings of all calibrations. An algorithm minimizes the expected cost by averaging costs among all calibration types and decides to probe by comparing the two alternatives. The first implementation of an algorithm contained a conceptual mistake related to the `fit_max()` function. It was overseen, and therefore, falsely assumed, that the optimal expected cost is monotone with the total job length (Christoph Dürr, pc). This mistake was corrected in the second implementation, which will be discussed in this chapter. Afterwards, its performance is compared to the randomized algorithm, defined by `schedule_rdn()` that randomly selects intervals for probing. Our algorithm provides better performance on all 121 test cases. In addition, it can be seen that the algorithm probes in the beginning, and after some interval, it does not probe anymore. This feature is denoted as the probing rank of the schedule, and it is measured with `algo_choice()` function. Finally, it can be seen that the ratio between calibration cost and probing cost influences the probability of probing an interval, which is demonstrated by `probe_prob()` function.

5.2.2 List of implemented functions

Implemented functions			
Function	Descrition	Parameters	Return
`fit_max(cal_len,job_len)`	computes maximal coverings for each calibration interval	lists of job lengths and calibration lengths	list of lists, where each set of inner lists represents maximal coverings of an interval
`schedule(remaining, cost_cal,cost_prob)`	schedules jobs by an algorithm described in 3.1	a tuple of the remaining number of jobs of each type, calibration and probing costs	cost and representation of a resulting schedule
`schedule_rdn(remaining), cost_cal,cost_prob)`	schedules jobs by an algorithm that randomly selects intervals for probing	a tuple of the remaining number of jobs of each type, calibration and probing costs	cost and representation of a resulting schedule
`schedule_2nd(remaining), cost_cal,cost_prob)`	schedules jobs by an algorithm that randomly selects intervals for probing	a tuple of the remaining number of jobs of each type, calibration and probing costs	cost and representation of a resulting schedule
`prob_rank(solution):`	determines whether probing rank depends on the number of remaining jobs	a schedule returned by a scheduling function	index of the first unprobed interval, number of remaining jobs of each type
`algo_choice(solution)`	determines whether probing is done after probing rank	a schedule returned by a scheduling function	-1 if the probing is done after probing rank,1 if it is not done, 0 if the schedule is inconclusive
`algo1_algo2(sch1, sch2)`	comparing costs of the two schedules	two schedules for comparing	1 if cost of sch1 is bigger than cost of sch2,-1 if it is smaller, 0 if schedules have the same costs
`prob_prob(job_nb)`	calculate the probability of probing an interval for different pairs of calibration and probing costs	a tuple of the number of jobs of each type	list of probabilities associated with calibration and probing costs ratio

5.3 User interface

5.3.1 Variables

The user interface is not a priority in this project. The aim of the software is to calculate the cost of a schedule and to demonstrate which intervals are probed, and which are not. In order to achieve this, the input is randomly generated and we do not require any user input. In case the user wants to change the ranges for job lengths, calibration lengths, calibration cost and probing cost, he or she is free to do so directly in the initialization part of the code. The initial ranges for the values of the variables are listed in the table below.

Initialization		
Variable	Description	Range
`job_types`	Number of different job lengths	2-6
`cal_types`	Number of different calibration lengths	2-6
`job_nb`	Number of jobs of each type	10-38
`cost_cal`	Calibration cost	100-200
`prob_cal`	Probing cost	30-50

5.3.2 Representing a schedule

Scheduling functions return the cost and the representation of a schedule. The first argument refers to the cost of the schedule, and the second argument is the schedule itself. The schedule is presented as a list of lists. Each inner list represents an interval. The first argument of each inner list is an interval type, and the others refer to the number of jobs of each type that are used in that interval. To indicate whether an interval is probed, the following logic is used. If the first argument in an interval is non-negative, then the type of the used interval is simply the value of the first argument, and it is probed. However, if the first argument is -1, this indicates that an interval is not probed, and its type is 0, i.e. it is assumed to be the shortest interval. Figure 1 is an example of the schedules returned by these two scheduling functions. The cost of the schedule returned by the random algorithm is 1782.55. Also, the first interval in this schedule is not probed. Therefore it is assumed to have the length of 33, which is the length of a shorter calibration.

Figure 1: Example of a schedule representation

```
instance job_nb=(12, 15) job_len=[8, 17] cal_len=(33, 55) cost_cal=153 cost_prob=41

Probing based on minimizing expected cost
(1641.7737426757812,
 [[1, [0, 3]],
  [1, [0, 3]],
  [0, [2, 1]],
  [1, [0, 3]],
  [-1, [2, 1]],
  [-1, [2, 1]],
  [-1, [2, 1]],
  [-1, [2, 1]],
  [-1, [2, 1]]])

Randomly probing
(1782.55859375,
 [[-1, [2, 1]],
  [-1, [2, 1]],
  [-1, [2, 1]],
  [1, [0, 3]],
  [1, [0, 3]],
  [-1, [2, 1]],
  [0, [2, 1]],
  [1, [0, 3]],
  [1, [2, 1]]])
```

5.4 Analysis of different schedules

5.4.1 Impact of calibration and probing costs ratio

The ratio between calibration cost and probing cost impacts how likely the algorithm is to perform probing. The function `prob_prob(job_nb)` analyses the probability of probing an interval for different calibration costs versus probing cost ratios. We let the calibration cost vary from 100 to 200, inclusive, and the probing cost vary from 20 to 50, inclusive. Therefore, the ratios are in the range of 2 to 10. We fix the other variables, i.e. job lengths, calibration lengths, and the number of jobs of each length, and run the scheduler. It is observed that the probability of probing an interval is higher if the ratio is bigger. In other words, if the ratio is high, the calibration cost is significantly higher than the probing cost, i.e. probing is relatively inexpensive compared to calibration. Therefore, the difference in the price of probing and not probing is not significant, and eventually, the expected cost of probed schedule will be smaller than that of the probed schedule. Also, if we think about the examples that are applicable to the problems in the real world, the smaller ratios are less relevant. This is because calibrations are, in general, quite costly compared to probing. As stated in section 1.1.1, calibration is associated with restoring the functionality of a machine, while probing refers to tests that are done at additional cost to determine its duration. One example of these probabilities associated with ratios that are generated by `prob_prob(job_nb)` is presented in Figure 2.

5.4.2 Impact of number of remaining jobs on probing

It is natural to assume that there is some correlation between the number of remaining jobs and probing. Here, observed whether the algorithm probes in the beginning or at the end of the schedule. Intuitively, it makes sense that it probes in the beginning, but stops probing in the end. However, after examining many schedules, it appears that this cannot be claimed as a general rule. We assumed that for every schedule that does not have all intervals un-probed, there exists an interval index, that can be understood as a critical point of a function, in a sense that before this index algorithm probes, and after this index it does not probe. Afterwards, we measured the correlation between this index and the number of the remaining jobs of each type. Intuitively, smaller indexes refer to little probing, which could be related to a large number of remaining jobs.

Therefore, there could be a linear relationship between the index and the number of remaining jobs. However, this was not the case for every schedule. Thus, we could not observe it as a general rule. This is probably due to the different job numbers in different observations. In fact, the moment after which the algorithm stops probing is probably related to the number of remaining jobs. However, it can not be claimed that it appears relatively at the end of the schedule because it depends on the initial number of jobs. However, this leads to observing another interesting pattern that appears in many schedules which do not have all intervals un-probed. This feature is called probing rank and it will be discussed in the next section. The absence of correlation between the probing rank and the remaining jobs of each type is presented in Figure 3.

Figure 2: Correlation between probing and calibration cost/probing cost ratio

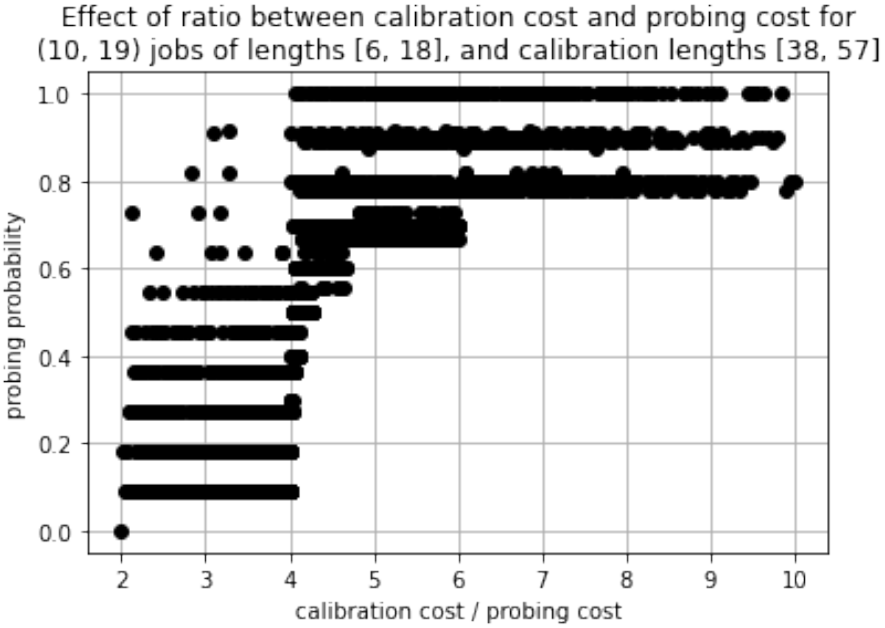
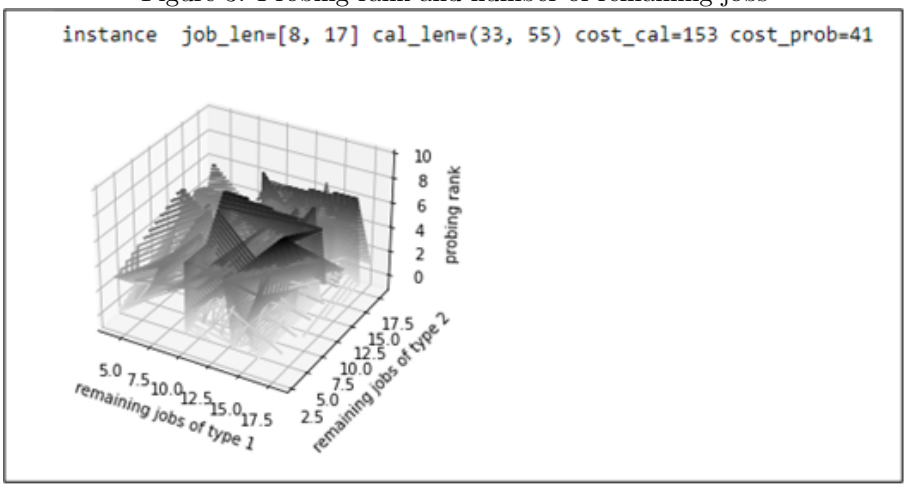


Figure 3: Probing rank and number of remaining jobs



5.4.3 Probing rank

Probing rank refers to the assumption that there exists an interval after which we stop probing. This is defined as a probing rank of a schedule. It has been observed that if the algorithm decides to probe, it usually probes consecutive intervals at the beginning of a schedule, and then stops probing after some index. The function `algo_choice(job_nb)` extracts the probing rank for different parameter settings, and observes how the schedule behaves relative to the rank. If the algorithm still decides to probe, it labels an indicator variable as false, i.e. the pattern is not present in the current solution. However, if the algorithm does not probe after the index, then it labels an indicator variable as true, indicating that the probing does happen before the index, and afterwards, it does not happen. The value -1 indicates that the algorithm does not probe any interval in the resulting schedule. Figure 4 shows the presence of this feature in 121 schedules. For this example, we observe schedules that have job lengths 10 and 15, and calibration lengths 34 and 52. They differ in calibration and probing costs, which are randomly generated, and vary from 100 to 200, and from 20 to 50. Also, the schedules differ in the number of jobs of each type, each varies from 10 to 20, inclusive.

Figure 4: Probing after probing rank in 121 schedules

```
instance job_len=[10, 15] cal_len=(34, 52)
1 indicates that algorithm does not probe after some interval (pattern present),
-1 indicates that it does (pattern not present),
0 indicates that it does not probe at all in this schedule(inconclusive)

array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
Number of 0s: 17
Number of 1s: 104
Number of -1s: 0
```

It can be noted that out of 121 observed schedules, only 17 do not probe at all, which is less than 15%. For all the other schedules, the indicator is 1. This implies that the pattern is present in more than 85% of the all observed schedules, and in all schedules that are not inconclusive.

5.5 Alternative approach

An alternative approach, which increases the performance of our algorithm is if we did not assume the shortest interval in the case of not probing. Moreover, if we uniformly select the length of the calibration, it is reasonable to incorporate some degree of risk in case we do not probe. One way to do this is to assume the second shortest calibration interval instead of the shortest interval. This approach is not applicable to the case with 2 calibration types, because the second smallest is the largest interval, and the schedule would not have much benefit from probing if it always assumes the longer interval in case of not probing. Therefore, in order to analyse this approach, we consider the case of k calibrations and 2 job types. If we let the scheduler assume the second shortest interval in case of not probing, we obtain interesting results. The scheduling function `schedule_2nd()` returns the cost and schedule of this alternative. Firstly, if the interval is actually the smallest, assuming the second smallest would result in executing some jobs while the machine is in the non-calibrated state. In order to address this, we enforce the penalty of one calibration cost in this case. It is important to note that this penalty refers to the way in which the risk is handled. Therefore, the choice of penalty is arbitrary, and the performance is dependent on its value. However, risk could be handled in different ways. For example, by changing the objective to the minimum completion time.

The performance comparison of the two algorithms is observed on 121 schedules that have job lengths 11 and 17, and calibration lengths 38, 46, 60, 76, 95, 110. Schedules differ in calibration and probing costs, which are randomly generated, and vary from 100 to 200, and from 20 to 50. Also, the schedules differ in the number of jobs of each type, each varies from 10 to 20. The results are shown in Figure 5. For 144 out of 121 observed schedules, an algorithm that assumes the second smallest calibration performs better than the initial algorithm. However, it is important to note that this alternative could not be applied without the correct estimation of the value of a penalty.

Figure 5: Example of an alternative approach to not probing

```
instance job_len=[15, 18] cal_len=(36, 50, 61)
0 indicates that algorithms have the same cost,
1 indicates that 2nd worst alg performs worse,
-1 indicates that 2nd worst alg performs better)

array([-1, -1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1, -1, -1, -1,  1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1,  1])
Number of 0s: 0
Number of 1s: 7
Number of -1s: 114
```

5.6 Major problems

There are two problems related to the implementation of an algorithm in 3.1.
Firstly, after the initial implementation of recursive scheduling functions, it was observed that they perform slowly. As explained in section 4.3.1, this is due to frequent repetitive calculations that commonly occur in recursive calls. The Python decorator `functools.lru_cache()` enhances the performance by using the Least Recently Used (LRU) memorization strategy for avoiding the calculations that are expected to be the same*What's new in python 3.2* n.d. The decorator is added in front of each recursive function. However, the decorator requires function parameters to be hashable. Prior to adding the decorator, lists were used for storing the job lengths, calibration lengths and job numbers. To utilize `functools.lru_cache()`, we had to convert them to tuples, which are immutable and hashable (Christoph Dürr, pc).
Secondly, generalizations of the base case aimed at exploring the case when the calibration types are not discrete, but given by some probability distributions. However, the current implementation depends on the total number of calibrations. In `fit_max(cal_len,job_len)` function, the total number of calibrations is used to precompute the maximal coverings of each calibration interval. This result is afterwards used in the scheduling function. Therefore, we were unable to directly implement the case when the calibrations are given as continuous random variables, but it is discussed in section 7.3.

6 Testing of dynamic programs

6.1 Testing approach

Testing of the dynamic program is performed in the following ways. Firstly, to test the performance of the schedule presented in 3.1, we compare its performance to the algorithm which randomly chooses whether to probe or not. Secondly, we perform unit testing to individually test the functions `prob_rank(solution)` and `algo_choice(solution)`.

6.2 Comparison with a random schedule

To determine how well our algorithm performs, we compare the cost of its schedule to the cost generated by the random schedule. The difference is how the choice between probing and not probing is made. Our schedule minimizes the expected cost over all maximal coverings of all calibrations. Afterwards, it compares the two alternatives and selects the one which is less expensive. This choice determines whether we probe or not. The current interval is probed if the probing cost added to the expected cost of the remaining schedule is smaller than the expected cost of the remaining schedule assuming that the current interval is the shortest. Random scheduler does not select the alternative that has the smaller expected cost, but randomly.

The performance comparison of the two algorithms is observed on 121 schedules that have job lengths 8 and 17, and calibration lengths 33 and 55. They differ in calibration and probing costs, which are randomly generated, and vary from 100 to 200, and from 20 to 50. Also, the schedules differ in the number of jobs of each type, each varies from 10 to 20. The results are shown in Figure 6. Results show that for all observed schedules, the proposed algorithm performs better than the algorithm that randomly selects intervals for probing.

Figure 6: Example of comparison with random schedule

```
instance job_len=[8, 17] cal_len=(33, 55)
0 indicates that algorithms have the same cost,
1 indicates that random alg performs worse,
-1 indicates that random alg performs better)

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
Number of 0s: 0
Number of 1s: 121
Number of -1s: 0
```

6.3 Unit testing

6.3.1 Overview

The validity of individual components is determined by performing unit testing. In this project, there are two functions that require unit testing, `prob_rank(solution)` and `algo_choice(solution)`. The first function returns the probing rank and the number of jobs of each type when the algorithm stops probing. The second function returns 1 probing takes place before the probing rank, -1 if some interval is probed after the probing rank, and 0 otherwise. These functions are tested with the `assertEquals(a, b)` function, that is implemented for this purpose. Function `assertEquals(a, b)` compares two inputs, a and b, and prints "Test passed. Values are equal" if the function returns the expected value. Otherwise, it prints "Test failed, because a is not b".

6.3.2 Test cases and results

To test these two functions, four test cases are created for each.
For testing `prob_rank(solution)`, we test the following four cases.

1. Probe some intervals
2. No probing
3. Probe every interval
4. Probe every interval, but not last

The reason for selecting these test cases is that they all represent different types of schedules that are usually generated by an algorithm. The function extracts the probing rank and the number of the remaining jobs for all the four test cases. The tests and results are displayed in Figure 7.

Figure 7: Testing probing rank function

```
In [38]: #test1, prob_rank() --> probing happens early
         res = (1663.875,[[0, [0, 2]],
           [0, [0, 2]],
           [0, [0, 2]],
           [0, [0, 2]],
           [-1, [0, 2]],
           [-1, [0, 2]],
           [-1, [0, 2]],
           [-1, [0, 2]],
           [-1, [3, 0]],])
         assertEquals(prob_rank(res), (4, 3, 8))

         Test passed. Values are equal.

In [37]: #test2, prob_rank() --> probing every
         res = (1480.009765625,
          [[-1, [0, 1]],
           [-1, [0, 3]],
           [-1, [0, 1]],
           [-1, [2, 1]],
           [-1, [2, 1]],
           ])
         assertEquals(prob_rank(res), (0, 4, 7))

         Test passed. Values are equal.

In [34]: #test3, probe_rank() --> no probing
         res = (1199,
          [[-1, [0, 2]],
           [-1, [0, 2]],
           [-1, [0, 2]],
           [-1, [2, 2]],
           [-1, [0, 2]],
           [-1, [2, 2]],
           [-1, [0, 2]],
           [-1, [2, 1]]])
         assertEquals(prob_rank(res), (0, 6, 15))

         Test passed. Values are equal.

In [40]: #test4, prob_rank() --> probe every but last
         res = (2062.0009765625,
          [[0, [0, 1]],
           [1, [0, 3]],
           [0, [3, 0]],
           [1, [6, 0]],
           [0, [3, 0]],
           [-1, [3, 0]]])
         assertEquals(prob_rank(res), (5, 3, 0))

         Test passed. Values are equal.
```

For testing `algo_choice(solution)`, we test the following four cases.

1. Feature does not exist
2. Inconclusive
3. Feature exists
4. Feature exists, but only last interval is un-probed

The reasons for selecting these test cases is similar as for the `prob_rank(solution)` function. For example, when the schedule is inconclusive, every interval is un-probed. If the schedule is not inconclusive, some interval is probed. Finally, if every interval is probed except for last, feature does exist. The detailed description of the function is discussed in section 5.4.3. Function returns 1 if the feature is present, -1 if the feature is not present, and 0 if the schedule is inconclusive. Figure 8 shows the test cases and the testing results.

Figure 8: Testing algo choice function

```
In [28]: #test1, algo_choice --> feature exists
         res= (1659.2890625,
          [[0, [1, 1]],
           [0, [1, 1]],
           [1, [1, 2]],
           [0, [1, 1]],
           [1, [1, 2]],
           [0, [1, 1]],
           [-1, [1, 2]],
           [-1, [3, 0]]])
         assertEquals(algo_choice(res), 1)

         Test passed. Values are equal.

In [46]: #test2, algo_choice --> inconclusive
         res= (1659.2890625,
          [[0, [1, 1]],
           [0, [1, 1]],
           [1, [1, 2]],
           [1, [3, 0]]])
         assertEquals(algo_choice(res), 0)

         Test passed. Values are equal.

In [45]: #test3, algo_choice --> feature does not eist
         res= (1659.2890625,
          [[0, [1, 1]],
           [-1, [1, 2]],
           [0, [1, 1]],
           [-1, [1, 2]],
           [1, [3, 0]]])
         assertEquals(algo_choice(res), -1)

         Test passed. Values are equal.

In [41]: #test4, algo_choice --> feature exists, but last
         res= (1659.2890625,
          [[1, [1, 1]],
           [1, [1, 1]],
           [0, [1, 1]],
           [-1, [1, 2]]],
           )
         assertEquals(algo_choice(res), 1)

         Test passed. Values are equal.
```

7 Conclusion

7.1 Summary of achievements

To conclude, we will first review the main achievements of this project. Firstly, this project shows initiative for further research in the field of scheduling problems with many uncertain calibrations. Namely, scheduling problems have been well studied in the field of computer science, and there are many variations including many job lengths, release times and deadlines, scheduling on many machines, scheduling with uncertain job lengths, and others. One variation is scheduling with calibrations, which originates from a program introduced by Sandia National Laboratories. The aim was to ensure the better quality of the US nuclear weapons stockpile. Jobs are scheduled on a machine that needs to be calibrated in order to successfully execute jobs. However, the machine stays in the calibrated state for a limited time, which can be either known or unknown prior to the job execution. In the second case, the concept of probing is introduced, as a mechanism to attain additional information about the calibration lengths at an increased cost. This project defines the problem in the field of scheduling with many calibrations and probing, and provides a pseudo-polynomial running time approximation algorithm for solving it. Firstly, we consider two possibilities for job lengths and two possibilities for calibration lengths. Afterwards, we generalise the problem by analysing the cases with many calibrations and many job lengths. The technique used for deciding whether to probe an interval or not is based on minimising the total expected cost of a schedule over all maximal coverings of all calibrations. We propose four dynamic programs for solving different generalisations of a problem. Finally, we provide an implementation of the solution in Python and compare its performance with a random schedule. An alternative scheduler is mentioned, as a way to increase performance by assuming the second smallest interval instead of the smallest interval, in the case of not probing. In addition, we discuss some properties of the schedule, such as the correlation between probing and calibration cost / probing cost ratio, and the tendency to probe first and then eventually stop probing. Finally, we mention ideas for the extensions to this project and future work.

7.2 Critical review

7.2.1 Research

The problem definition was not finalised at the beginning of the project. Therefore, extensive research was conducted to analyse the problems that were studied in the area of scheduling problems. This resulted in some idle time in terms of progress, as some papers required more time to understand and discuss. The initial phase of the project allowed exploring different settings of the problem, for the purpose of finding the one we would finally work on. For example, we considered adding release time and deadlines of the jobs, activation time to the calibrations (non-instantaneous calibrations), and allowing preemption of the jobs (pausing jobs when the calibration ends). Researching various problem settings was valuable, as it not only helped us finalize the problem definition but also provided the base for extending this project in the future.

7.2.2 Implementation and testing

Before the final implementation of an algorithm, the first version was implemented, as stated in section 4.3.3. The first version of the code contained a fundamental mistake in the way it calculated the maximal coverings of intervals. The problem was that the function did not return all the maximal coverings of an interval, but the maximum covering of an interval, based on the number of the remaining jobs (Christoph Dürr, pc). The problem with this implementation is that it assumes that the total expected cost is monotone with the number of the remaining jobs. However, we cannot assume this property because different numbers of remaining jobs generate different values for expected cost for different maximal packs of an interval.

After the first version was implemented, it was tested against the random schedule, a schedule that probes every interval, and a schedule that does not probe at all. The costs of these schedules are shown in Figure 9.

Figure 9: First version of the code

```
In [18]: result1= schedule1(n_1, n_2)
         result2 = schedule2(n_1, n_2)
         result3 = schedule3(n_1, n_2)
         result4= schedule4(n_1, n_2)


         print("Schedule with algorithm costs :" +str(result1))
         #print("Schedule with algorithm is :" +str(schedule2))
         print("Schedule with random probing costs :" +str(result3))
         print("Schedule with no probing costs :" +str(result2))
         #print("Schedule with no probing is :" +str(schedule1))
         print("Schedule with probing every costs  :" +str(result4))
         #print("Schedule with probing every is :" +str(schedule4))

         Schedule with algorithm costs :3542.062415122986
         Schedule with random probing costs :4282.0
         Schedule with no probing costs :4750
         Schedule with probing every costs  :3557.395707786083
```

From these results, it was clear that the implementation of an algorithm does not perform very well against the schedule that probes every interval. Alternatively, the first implementation of an algorithm chose the probe for almost every interval. This result was crucial for discovering the oversight related to `fit_max()` function, that impacted the correctness and performance of an algorithm. This issue was handled properly in the second version of the code, by precomputing the maximal coverings of the interval and storing their values in `max_fit()` variable. When the scheduling function is called, the algorithm loops through all the maximal coverings of all intervals, instead of calling `fit_max()` function for every interval and the number of remaining jobs.

7.3 Suggestions for extensions to the project

This project can be further extended in two apparent ways. The first area of exploration is related to allowing jobs to have release times and deadlines. Angel et al. (2017) and Bender et al. (2013) studied the problem of minimizing the total cost for the scheduling problem with many calibrations when jobs have release times and deadlines. It is shown that the problem becomes NP-hard when there are many calibration types, and jobs have arbitrary processing times when the preemption is allowed. Therefore, they considered variations of the problem such as adding activation length to the calibrations (non-instantaneous) and assuming unit-time processing times of the jobs Angel et al. 2017. These variations could be studied in our problem as well. Secondly, our project studies the case when the calibration length is described by a discrete random variable. As suggested by Dr. Li during our weekly meetings, the calibration length could be described by a continuous random variable. We could use some of the well-known probability distributions, such as normal distribution, or exponential distribution (Christoph Dürr, pc). For example, if the calibration length is drawn from a normal distribution $N(\mu, \sigma^2)$, then we know that the expected value for the calibration length is μ, and we could utilize the confidence intervals, i.e, 99.7% of the values for calibration intervals would lie within three standard deviations. These properties could be used by algorithms for deciding whether to probe or not.

8 Monthly logs

8.1 September

We began working on this project more seriously in the mid September. Weekly meetings have been fixed for Thursday 4pm Hong Kong time, via Zoom. The participants are Dr. Li Minming, Dr. Dürr Christoph and Andela Basic. The September was mostly planned for discussions and research.

8.2 October

The October was planned for exploring different paths a project might take. As mentioned in the sections 1.2.1. and 1.2.2., the problem definition may vary. Therefore, it was important to determine the base case and agree upon the first approach that we will use to solve it. We defined the base case as in section 1.2.1. and decided that we will first analyse the dynamic programming approach.

8.3 November

In November we started working on building a DP algorithm for solving a scheduling problem with uncertain calibration interval length. As described in section 1.2.1. and 1.2.2. we started from base case and gradually generalised it. The initial DP algorithm is presented in section 3.1. The main limitation of this algorithm is that it assumes the shortest interval in case when probing is not done. We are planning to address this limitation in the future work, by including the risk component to the current solution.

8.4 December

In December we had several meetings to discuss the research papers that we have been interested in. In addition,we developed an alternative algorithm in section 3.2. We decided to take a break because of the holidays.

8.5 January

In January, we focused on correcting the mistakes from the Report I and exploring the preemptive schedules and work done by Angel et al. and Chen et. al.

8.6 February

In February, we focused on the first implementation and testing of the code. After first testing, fundamental mistake was found. This lead to the second implementation and second testing of the code.

8.7 March

March was reserved for final discussions, representing the schedule and documentation.

References

Angel, Eric et al. (2017). "On the complexity of minimizing the total calibration cost". In: *Frontiers in Algorithmics*, pp. 1–12. DOI: 10.1007/978-3-319-59605-1_1.

Basu Roy, Aniket et al. (2019). "Approximating robust bin packing with budgeted uncertainty". In: *Lecture Notes in Computer Science*, pp. 71–84. DOI: 10.1007/978-3-030-24766-9_6.

Bender, Michael A. et al. (2013). "Efficient scheduling to minimize calibrations". In: *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. DOI: 10.1145/2486159.2486193.

Chawla, Shuchi et al. (2020). "Pandora's box with correlations: Learning and approximation". In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. DOI: 10.1109/focs46700.2020.00116.

Dürr, Christoph et al. (2020). "An adversarial model for scheduling with testing". In: *Algorithmica* 82.12, pp. 3630–3675. DOI: 10.1007/s00453-020-00742-2.

Erickson, Jeff (2019). "Dynamic Programming". In: *Algorithms*. Jeff Erickson, pp. 99–100.

Floudas, Christodoulos A. and Xiaoxia Lin (2005). "Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications". In: *Annals of Operations Research* 139.1, pp. 131–162. DOI: 10.1007/s10479-005-3446-x.

Kang, Jangha and Sungsoo Park (2003). "Algorithms for the variable sized bin packing problem". In: *European Journal of Operational Research* 147.2, pp. 365–372. DOI: 10.1016/s0377-2217(02)00247-3.

Karmarkar, Narendra and Richard M. Karp (1982). "An efficient approximation scheme for the one-dimensional bin-packing problem". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. DOI: 10.1109/sfcs.1982.61.

Kondili, E., C.C. Pantelides, and R.W.H. Sargent (1993). "A general algorithm for short-term scheduling of batch operations—I. Milp formulation". In: *Computers amp; Chemical Engineering* 17.2, pp. 211–227. DOI: 10.1016/0098-1354(93)80015-f.

Levi, Retsef, Thomas Magnanti, and Yaron Shaposhnik (May 2018). *Scheduling with testing*. URL: https://pubsonline.informs.org/doi/10.1287/mnsc.2017.2973.

Li, Zukui and Marianthi Ierapetritou (2008). "Process scheduling under uncertainty: Review and Challenges". In: *Computers amp; Chemical Engineering* 32.4-5, pp. 715–727. DOI: 10.1016/j.compchemeng.2007.03.001.

Lin, Xiaoxia, Stacy L. Janak, and Christodoulos A. Floudas (2004). "A new robust optimization approach for scheduling under uncertainty:" in: *Computers amp; Chemical Engineering* 28.6-7, pp. 1069–1085. DOI: 10.1016/j.compchemeng.2003.09.020.

Seiden, Steven S. (2002). "On the online bin packing problem". In: *Journal of the ACM* 49.5, pp. 640–671. DOI: 10.1145/585265.585269.

Verderame, Peter M. et al. (2010). "Planning and scheduling under uncertainty: A review across multiple sectors". In: *Industrial amp; Engineering Chemistry Research* 49.9, pp. 3993–4017. DOI: 10.1021/ie902009k.

Weitzman, Martin L. (1979). "Optimal search for the best alternative". In: *Econometrica* 47.3, p. 641. DOI: 10.2307/1910412.

What's new in python 3.2 (n.d.). URL: https://icsa.cs.up.ac.za/docs/python3/whatsnew/3.2.html.