Finding Robust Schedules in the Stochastic Resource Constrained Project Scheduling Problem using Probabilistic Inference

while using unmodified schedulers

Kasper van Duijne[1]
Supervisors: Sebastijan Dumančić[1]**, Reuben Gardos Reid**[1]**, Issa Hanou**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfillment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Kasper van Duijne
Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Reuben Gardos Reid, Issa Hanou, Merve Gürel

Abstract

Scheduling problems are present in many real-world situations, such as construction projects, manufacturing processes, or train timetabling. One common formalization is the Resource Constrained Project Scheduling Problem (RCPSP), where the goal is to find an optimal schedule given limited resources. Traditional algorithms optimize for project duration under deterministic assumptions, which could lead to poor performance under uncertainty. This thesis explores how to create robust schedules, meaning they can withstand uncertainty, for stochastic task durations. Robust schedules minimize delays, measured as the difference between a task's completion time and its deadline. The method proposed uses probabilistic programming as a tool to accomplish this. A robustness distribution over schedules is inferred using importance sampling, and a robust schedule can be selected from that explored distribution. Importantly, this approach presented in this thesis can be applied on top of existing scheduling and simulation algorithms without requiring any knowledge or changes to themselves. Created schedules can also be abstracted, thus do not need to be analyzed or seen. This makes the proposed method general and easy to adopt in practice.

1 Introduction

Uncertainty is a major challenge in real-world project scheduling in many sectors [12]. Whether in transportation, manufacturing, or construction, the duration of individual tasks often deviates from estimates due to unforeseen delays or variability. For example, a construction project might be delayed by weather conditions, a manufacturing process might be disrupted by equipment failure, or it is uncertain at what time a team needs to complete a certain task. As a result, even optimized schedules can perform poorly in practice when reality diverges from a predetermined schedule. This raises the question: how can we find schedules that remain effective even when task durations are stochastic or uncertain?

The Stochastic Resource Constrained Project Scheduling Problem (stochastic RCPSP) formalizes this challenge. It involves scheduling a set of tasks that require limited resources to be executed. In the stochastic variant, task durations are not fixed but follow probability distributions. This stochasticity can lead to significant variation in the execution times of tasks. Improving the robustness of schedules, such that their robustness remains acceptable under uncertainty, is an important goal.

This work investigates whether the approach of Gardos Reid [3] and the code from van den Houten et al. [10] can be applied to the stochastic RCPSP. The former suggests that inference approaches can be applied to black-box schedulers and simulators to find robust schedules. This paper builds on that insight by asking:

> **Can we use probabilistic inference to obtain robust schedules for the stochastic RCPSP, without modifying the underlying deterministic scheduling algorithm?**

Probabilistic programming is a paradigm where it is easy for users to define statistical models, making it easy to work with uncertainty. These programs also do the work of applying inference steps, so the user does not need to manually provide such code [9].

The earlier stated research question is addressed through the following sub-questions:

1. How can task duration uncertainty in the stochastic RCPSP be effectively modeled using a probabilistic programming framework?

2. Can probabilistic inference using importance sampling generate a meaningful distribution over schedules without modifying the underlying scheduler?

3. How can the resulting distribution be used to select robust schedules, and what metric captures schedule robustness?

This paper applies this novel approach from Gardos Reid to robust planning in the stochastic RCPSP. The underlying instance or base schedule does not need to be analyzed or perturbed to find robust schedules. If successful, this methodology could be applied to a wide range of scheduling domains. Since the scheduler and simulator are treated as black-boxes, the approach is easily transferable and implementable across various scheduling problems and algorithms.

The method is validated on a motivating example, where the inferred robust schedule outperforms the base schedule under uncertainty. Specifically, the robust schedule achieves a 100% deadline success rate, compared to 58% for the base schedule. These results confirm that probabilistic inference over uncertain durations can successfully identify robust schedules, even when using an unmodified deterministic scheduler. The rest of this paper explains the approach, evaluates it, and discusses its limitations and potential extensions.

2 Background

Before proceeding to Chapter 3 (Methodology), it is necessary to establish a clear understanding of the problem instance in Section 2.1, go over related work in Section 2.2, and lastly provide a motivating example of a scheduling instance in Section 2.3.

2.1 Scheduling problem instance

Formally, the stochastic RCPSP instance contains the following elements:

Let $\mathcal{T} = \{1, 2, \ldots, n\}$ be the set of tasks, and let $\mathcal{R}$ be the set of resources.
For each task $t \in \mathcal{T}$:

- Let $\hat{d}_t \in \mathbb{Z}_{\geq 0}$ denote the estimated (mean) duration.
- Let D_t be a random variable modeling the uncertain duration of task t
- Let $\delta_t \in \mathbb{Z}_{\geq 0} \cup \{\infty\}$ denote the deadline, where $\delta_t = \infty$ indicates no deadline.
- Let $S_t \subset \mathcal{T}$ be the set of successors of task t, that can only start after task t has completed.
- For each resource $r \in \mathcal{R}$, let $a_{t,r} \in \mathbb{Z}_{\geq 0}$ denote the number of machines of resource r required by task t.

For each resource $r \in \mathcal{R}$:

- Let $c_r \in \mathbb{Z}_{>0}$ denote its capacity, i.e., the number of machines available.

A resource can execute multiple tasks concurrently, provided that the total number of machines in use at any given time does not exceed its capacity.

The RCPSP consists of constructing a schedule that includes all tasks, satisfies all constraints, and minimizes the total project duration, which is the time at which all tasks have been completed, also known as the makespan. It is not needed for the purpose of this research what a schedule looks like under the hood, as a goal of this research is to keep a schedule abstracted.

2.2 Related Work

This paper builds upon the probabilistic inference methodology proposed by Gardos Reid [3]. Their work demonstrated that robust schedules can be inferred from black-box schedulers and simulators by using probabilistic programming. In their case, various inference techniques were used to find robust shunting schedules under delay uncertainty. They hypothesized that the densest region in the posterior robustness distribution over schedules corresponds to more robust schedules. This insight is crucial to this paper's work: adapting Gardos Reid's probabilistic modeling approach to a new domain, the stochastic Resource Constrained Project Scheduling Problem. The method demonstrates that robustness can still be inferred without analyzing the schedule or modifying the simulation or scheduling programs.

Another paper by van den Houten et al. [10] approaches the stochastic RCPSP domain from a learning perspective. They propose a Decision-Focused Learning (DFL) approach that learns to predict schedule-relevant parameters (task durations) using historical data, and repairs schedules to reduce post-hoc regret. In contrast to our method, their approach requires a training set of historical task durations. Our approach does not require training data or a differentiable pipeline. It can be applied directly to any black-box scheduler and simulator, making it usable for settings where domain data is scarce or unavailable.

Further robust scheduling techniques for stochastic RCPSP are explored by Fu et al. [2], who minimize expected makespan by analyzing instance-specific properties. Their method and others like it depend on interpreting the internal structure of the schedule and model. In contrast, this paper proposes a method in which no knowledge of the schedule's internal representation is required. Schedules are sampled, evaluated via simulation, and selected based on empirical robustness.

Finally, Yeganeh and Zegordi [14] presents a scenario-based heuristic using a relaxation of constraints to find robust schedules under duration uncertainty. While their method incorporates scenario data, it requires direct modification of the schedule search process. Again, this contrasts with this paper's black-box belief. The proposed method uses inference to operate over the output of an existing scheduler and simulator, without modifying their internals.

In summary, the novelty of this paper is that it uses a probabilistic inference technique to infer robustness directly from observed outcomes, requiring neither training data nor access to the generated schedules, scheduler, and simulator logic. By evaluating sampled schedules under simulation and weighting them by performance, we construct a posterior over schedules from which robust ones can be selected. This general method is potentially applicable across scheduling domains, wherever uncertainty exists, if a scheduler and simulator are available.

2.3 Motivating Example

Consider a scenario with:

- One resource with two machines.
- Three independent tasks: t_1, t_2, and t_3, with no precedence constraints.
- A deadline of time 8 for t_3.
- An assumed distribution for each task's duration. Here assumed to be normally distributed around each task's expected durations, with a standard deviation $\sigma = 0.8$. This is further motivated in Section 4.1

In Table 1, one can find the expected durations and resource requirements of the tasks.

Task	Expected Duration	Resource Demand
t_1	2	1 machine
t_2	4	1 machine
t_3	4	2 machines

Table 1: Expected durations and resource demands.

Using this instance together with the given expected task durations, the scheduler produces what this paper calls the 'base schedule'. In the base schedule (Figure 1), t_1 and t_2 start at time 0, each on one machine. t_3 starts at time 4, when both machines are free.

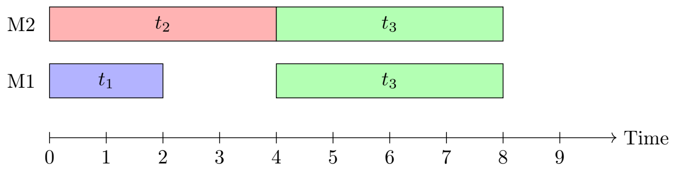


Figure 1: Base schedule. t_3 starts at time 4.

However, when introducing the uncertainty in the task durations, this schedule is vulnerable: delays in durations of t_1, t_2, or t_3, can postpone the finish time of t_3, risking a missed deadline.

An alternative schedule improves robustness by prioritizing t_3 (Figure 2). t_3 starts first, ensuring its deadline of 8 is always met (except if t_3's duration is longer than 8 units itself, which, assuming the current uncertainty, is highly unlikely). This robust schedule improves deadline reliability: t_3 is no longer affected by other delays, only itself, and finishes earlier. The goal of this paper's proposed method is that this robust schedule can be found.

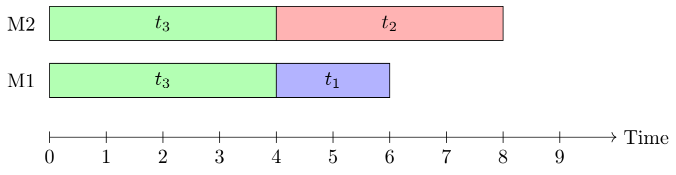


Figure 2: Robust schedule. t_3 starts first.

3 Methodology

With the background in place, the method of this paper can be discussed. The first section will give a high-level overview of the process, and the subsequent sections will explain the individual steps in detail.

3.1 Delay and Robustness Definition

The first step in inferring robust schedules is to define robustness. This paper defines robustness as the empirical success rate of a given schedule under simulated uncertainty in task durations. Let

1. π be an arbitrary schedule.
2. $\mathcal{T}$ be the list of all tasks.
3. $\vec{d} = (d_1, d_2, \ldots, d_{|\mathcal{T}|})$ be a randomly sampled vector of durations for each task.
4. S be a simulator that, given a schedule π and durations $\vec{d}$, returns a finish time f_t for each task $t \in \mathcal{T}$.

Then the *delay vector* of a schedule π under duration vector $\vec{d}$ is defined as:

$$\vec{delay}(\pi, \vec{d}) = (f_t - \delta_t)_{t \in \mathcal{T}}\,,$$

where f_t is the finish time of task t obtained by simulating π with a duration vector $\vec{d}$, and δ_t is the deadline of task t.

We say that a schedule π is *successful* in simulation i if all components of the delay vector are less than or equal to zero:

$$\text{success}_i(\pi, \vec{d}) = \begin{cases} 1 & \text{if } delay_t(\pi, \vec{d}) \leq 0 \quad \text{for all } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

Let N be a number of simulations. The *robustness* of a schedule π is defined as the fraction of successful simulations:

$$\text{robustness}(\pi) = \frac{1}{N} \sum_{i=1}^{N} \text{success}(\pi).$$

It is important to note that the delays in the delay vector can be negative. This is useful for plotting a delay distribution for a given schedule, to analyze its performance under uncertainty.

3.2 Method Overview

The methodology consists of constructing a probabilistic framework that models uncertainty in task durations, calling a scheduler on sampled durations, and evaluating the created schedules using a simulator. By writing this pipeline in a probabilistic programming framework, we apply importance sampling to infer which schedules are most robust under the given uncertainty.

At a high level, each iteration of the inference algorithm, importance sampling in particular, performs the following steps:

1. **Sample** a vector of task durations from the uncertainty distributions.
2. **Call** the scheduler to compute a schedule for those sampled durations.
3. **Simulate** how this schedule performs across multiple uncertain scenarios.
4. **Assign** a weight to the schedule based on the average delay of its delay vectors.
5. **Return** a trace (consisting of a schedule and its corresponding weight).

Importance sampling is repeating this process many times. The resulting weighted traces form an empirical distribution over schedules. This distribution is then cumulatively weighted for each unique schedule, giving a robustness distribution over all schedules. Finally, the highest cumulatively weighted schedule is selected as the inferred robust schedule.

Figure 3 illustrates this pipeline, showing the interaction between the model, importance sampling, and the post-processing step, where the most promising schedule is selected.

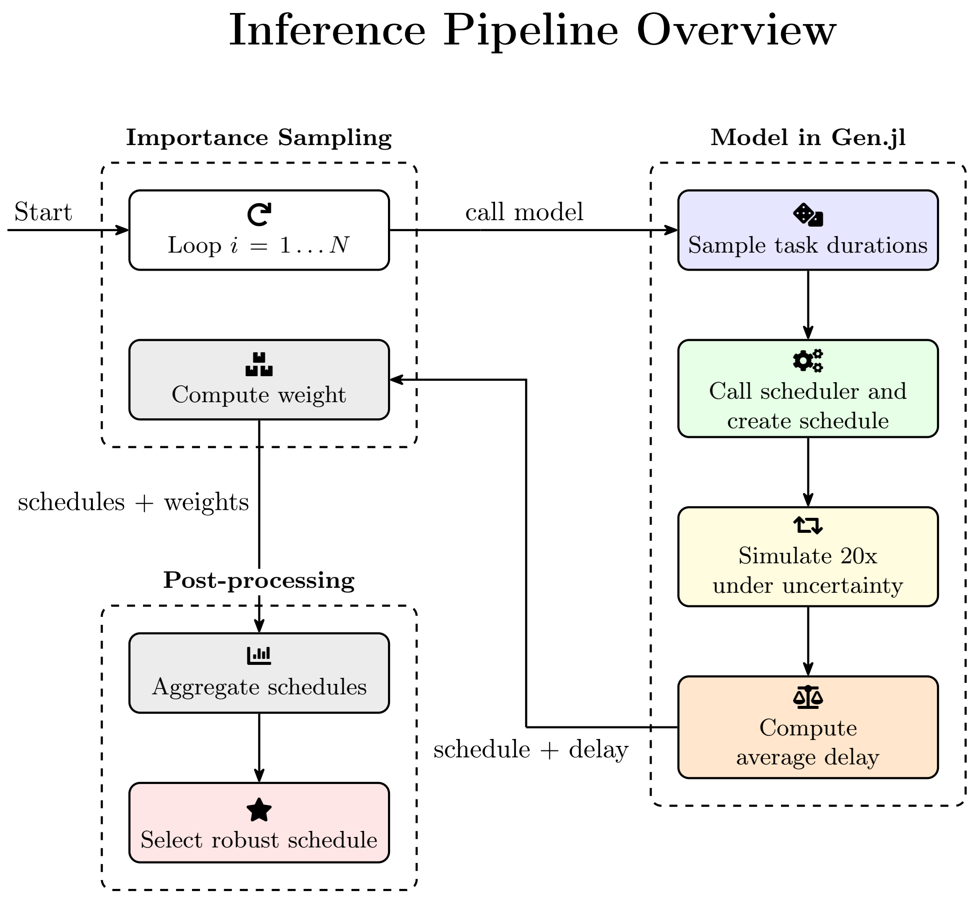


Figure 3: Inference pipeline layout. The program starts with importance sampling. This repeatedly calls the model. The model samples durations, creates a schedule, simulates the schedule, and then assigns a weight to it. After the importance sampling loop, post-processing is done to select the potential robust schedule.

```
@gen function rcpsp_model(expected_durations)
    sampled_durations = []
    for i in 1:n_tasks
        sampled_durations[i] = @trace(distribution(expected_durations[
            i]), (:duration, i))
    end
    schedule = create_schedule(sampled_durations)
    mean_delay = estimate_delay(schedule, n_sim=20)
    @trace(normal(max(0, mean_delay), 0.1), :delay)
    return schedule, mean_delay
end
```

Listing 1: Model for stochastic RCPSP

```
@gen function importance_sampling(num_iters)
    observations = Gen.choicemap()
    observations[:delay] = 0
    traces, log_norm_weights, _ = Gen.importance_sampling(
        rcpsp_model, observations, num_iters)
    return traces, log_norm_weights
end
```

Listing 2: Importance sampling loop.

3.3 Generative Model

To model uncertainty, we use the Gen.jl probabilistic programming framework [1]. The generative model samples task durations from a prior distribution, calls the scheduler, and estimates the schedule's average delay via some simulations. That average delay is later used to assign a weight to that generated schedule.

As seen in Listing 1, pseudocode for this model is shown. The *estimate_ delay* function runs 20 simulations on the schedule and returns the average delay. The *distribution* function can be an arbitrary uncertainty or distribution.

3.4 Inference via Importance Sampling

Importance sampling is used to infer robust schedules [8]. Each trace generated by the model includes a schedule and a corresponding normalized log-likelihood. This log-likelihood is determined by how close the schedule's average delay is to zero. The observation *':delay = 0'* encourages the model to favor traces with a simulated average delay closest to zero. Thus, schedules with a lower simulated average delay receive a higher corresponding weight.

3.5 Post-Processing and Schedule Selection

After importance sampling, we obtain a set of weighted traces, each containing a potential schedule and its normalized log-weight.

To use these traces to find a robust schedule, we proceed as follows:

1. Extract all the distinct schedules from the traces.

2. For each unique schedule, sum the weights of all traces that contain that schedule.
3. Select the schedule with the resulting highest cumulative weight as the most robust schedule.

This aggregation process approximates the posterior robustness distribution over schedules. This informed distribution (due to importance sampling) assumes a "robustness" value for each given unique schedule. We hypothesize that the densest region in this posterior corresponds to the most robust schedule.

Finally, we simulate the selected robust schedule and compare its robustness (success rate) to that of the base schedule, which is computed using the expected durations. This determines whether the inferred schedule is an improvement.

4 Results

Having introduced our methodology, we now present the experimental setup and the resulting data. First, we describe how the experiment is structured so it can be reproduced. We then show results on the motivating example problem instance.

4.1 Experimental setup

The problem instance chosen is the one discussed in Section 2.3.

The process is run on a *i5-6400 Intel processor* with 16 *GB of memory.* There are no time limits for the runtime of the algorithm. The scheduler and simulator are used from van den Houten et al.[10], and an IBM CPLEX CP solver version 12.9 is used [5]. Python version 3.11.4 is used (https://www.python.org/), and Julia version 1.11.5 (https://julialang.org/). It is important to note that the scheduler and simulator used work with discrete task duration lengths. Thus, after sampling durations, they are rounded to the nearest integer.

We model task durations using a normal distribution centered around each task's expected duration, with a standard deviation of 0.8. The reason for a normally distributed uncertainty is that given enough samples of task durations from real-world execution, it should follow a normal distribution [6]. Thus, the uncertainty of task durations can be modeled using that distribution. The standard deviation of 0.8 was chosen arbitrarily, large enough to see a wide range of potential unique schedules, but low enough to make sure the expected task duration is most common. With this standard deviation, approximately 50% of the sampled task durations will lie within ± 0.54 of the expected duration. As task durations are rounded to the nearest integer, this ensures that around half of the sampled durations are equal to the expected task duration.

Each schedule is evaluated in the Gen.jl model using 20 simulations to estimate the average delay. Importance sampling runs for 2000 iterations, which is sufficient to explore the schedule space and see recurring schedules appear multiple times. This indicates enough coverage of the high-density regions in the posterior schedule distribution. The number of simulations is a tradeoff between speed and performance, as the total number of times the scheduler is called grows by the number of iterations that importance sampling is doing, multiplied by how often the schedule is simulated in the Gen.jl model. Finally, the base schedule and the robust schedule are compared using 2000 simulations.

4.2 Example Instance

We apply our method to the example from Section 2.3, where a more robust schedule is known to exist. This serves as proof of concept of our method. To confirm our results, the resulting schedule distribution of our experiment is plotted. This is done to verify that the posterior schedule distribution is sufficiently explored. As seen below in Figure 4, that is the case, as we see some schedules occur over 150 times. We assume the robust schedule is in the densest region of the robustness posterior distribution, or the highest cumulative weighted schedule.

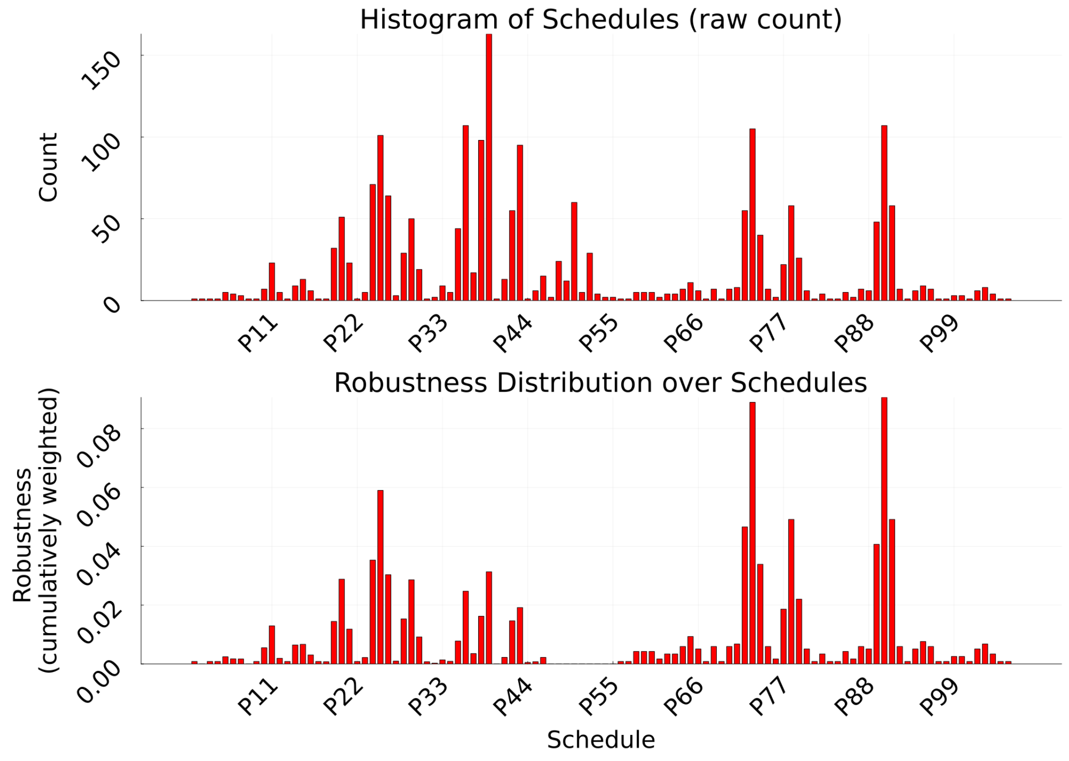


Figure 4: Schedule posterior distributions over 2000 iterations of importance sampling. The empirical distribution (raw schedule counts) is shown above. The robustness distribution over schedules (cumulative weighted distribution) is shown below.

As shown in Figure 5, the selected highest cumulative weighted schedule was indeed the robust schedule as discussed in Section 2.3. This robust schedule schedules t_3 first.

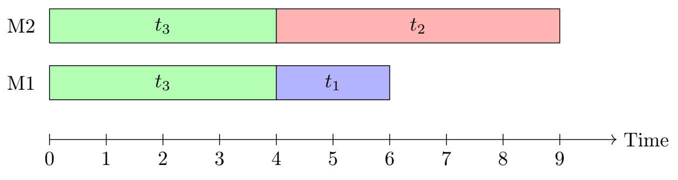
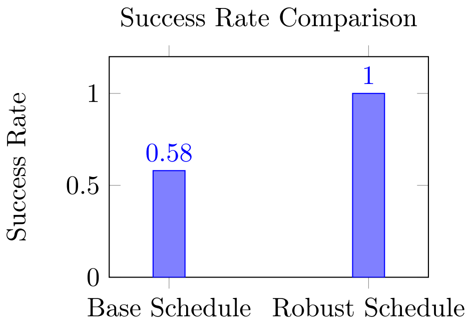
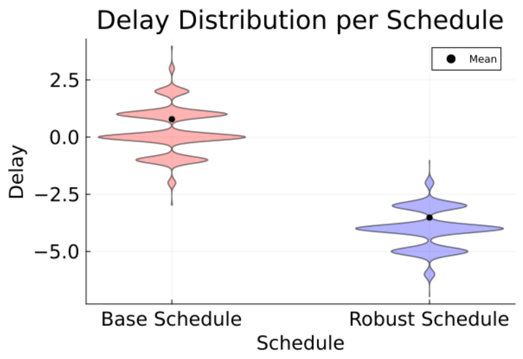


Figure 5: Robust schedule selected from the robustness distribution over schedules. t_3 starts first.

The robust schedule can also be compared with the base schedule. The base schedule is the schedule that is generated deterministically using the estimated task durations. The following result is shown in Figure 6. Just as theorized, the robust schedule has a 100% success rate, while the base schedule fails under certain sampled task durations. This is also seen in the delay distribution on the right-hand side. Here, the robust schedule's delay is always negative (and depends solely on the sampled task duration of t_3). The reason that the delay can only be an integer value is that the scheduler and simulator work with integer task durations.



(a) Comparing robustness of schedules (Success rate for meeting t_3's deadline).



(b) Delay distribution violin plot, using the sum of the delay vector per simulation, as defined in 3.1.

Figure 6: Comparison of base schedule to robust schedule under uncertainty, using 2000 simulations.

5 Discussion

Now that the results are shown, it is possible to analyze them. First, the general results are discussed, after which the limitations of the method are looked at.

5.1 Analysis of Results

The main result is that the proposed method successfully inferred a more robust schedule in the motivating example. First, looking at Figure 4, it is shown that the robustness distribution over schedules is sufficiently explored. In the raw count histogram above, recurring schedules are seen. The distribution over all schedules does not match the robustness distribution, which is good, as this means certain schedules have received higher weights during importance sampling. From this posterior robustness distribution, the densest schedule was selected (or highest cumulatively weighted). Under stochastic task durations, the base schedule failed in a significant fraction of simulations (58% success rate), while the robust schedule found by the inference method achieved a 100% success rate in meeting the deadline of task t_3. A high-risk task seems to be prioritized in this method and scheduled earlier. The reason for this can be explained due to importance sampling. Whenever sampled durations cause the generated schedule to have a makespan larger than 8, task t_3 will be scheduled first. The scheduler prefers to create a schedule where the delay constraint is followed, thus meaning t_3 needs to start first. These schedules also receive a higher weight, as their simulated delay is lowest. This can be verified by Figure 4. In the top graph, it is seen that the most occurring schedule is not the same as the highest cumulative weighted schedule. The reason is that only the schedules that schedule t_3 first receive a high weight from importance sampling. Thus, importance sampling explores the robust schedule distribution effectively, such that a robust schedule is found.

5.2 Limitations

First, going over the validity of the proposed method [13]. The external validity is difficult to verify precisely, as the method has not been tested on larger or multiple instances. This is further proposed as an area of further research in Section 7.3.

The biggest theorized limitation of this method would be that under larger schedules, the resulting posterior distribution would be extremely sparse. This could mean the same schedule is never found twice, thus rendering our robustness distribution over schedules unexplored. A solution, however, would be to group schedules together that are closely related, allowing for a potential posterior distribution that is sufficiently explored. This could be done by eliminating symmetry (schedules that are effectively equal, but differ only by the assignment of tasks to identical machines). A drawback is that an algorithm needs to be developed that compares the schedules with each other. This loses two major benefits of this research. The first being that this method could be applied to existing software, as a new algorithm needs to be developed. The second being that the generated schedules can no longer be treated as a black-box, as they need to be looked at.

The second theorized major limitation is the performance of the method. The relatively small example instance took two minutes to run. Depending on the time complexity of the scheduler and simulator, the method proposed can become infeasible on real-world schedules as the size grows. Larger schedules also mean importance sampling needs more iterations to sufficiently explore the posterior distribution, potentially increasing the run time even more.

6 Responsible Research

6.1 Ethical Concerns

This research was done following the Netherlands Code of Conduct for Research Integrity [7]. The following principles were adhered to: honesty, scrupulousness, transparency, independence, and responsibility.

This research presents minimal ethical concerns. The proposed method aims to improve schedule robustness using existing scheduling and simulation algorithms. In principle, a malicious actor could exploit the method to identify and avoid robust schedules for sabotage purposes. However, such an actor would already require the control to select or influence schedules, in which case they could simply choose non-robust schedules anyway, without relying on this method. Therefore, any ethical concerns or risks introduced by this research are negligible.

6.2 Reproducibility

The reproducibility of this method is high. The probabilistic programming framework used, Gen, is an open-source project[1]. The IBM CPLEX solver is not open source, but is free for students and people working in academics[4]. All algorithmic parameters are described in Subsection 4.1. The instance used is a self-constructed example and described in detail in Section 2.3, and thus can be recreated. The scheduling algorithm and simulator used are from [10], which is also open-source. The code containing the proposed method is shared and open source as well [11].

7 Conclusions and Future Work

This chapter summarizes the findings of this thesis by answering the research questions, discussing the main conclusions, and proposing directions for future research.

7.1 Answering the Research Questions

1. How can task duration uncertainty in the stochastic RCPSP be effectively modeled using a probabilistic programming framework? Task duration uncertainty can be represented using prior probability distributions over task durations. This thesis used normal distributions centered on the expected task durations. These distributions were embedded into a probabilistic model written in Gen.jl. Importance sampling then proceeds to use this model.

2. Can probabilistic inference using importance sampling generate a meaningful distribution over schedules without modifying the underlying scheduler? Yes. Importance sampling using the probabilistic model yielded a weighted collection of schedules, where higher weights were assigned to schedules with lower average delays under simulation. The method effectively built a robustness distribution over schedules, despite using an unmodified and black-box deterministic scheduler and simulator.

3. How can the resulting distribution be used to select robust schedules, and what metric captures schedule robustness? The highest weighted schedule from the robust schedule distribution is selected. This corresponds to the densest region of this distribution. Robustness of this schedule was compared to the base schedule, generated from expected task durations. Robustness was measured as the empirical success rate: the fraction of simulations in which all task deadlines were met. This robustness metric effectively captured schedule quality under uncertainty and guided the selection of a robust schedule, outperforming the base schedule in success rate.

7.2 Main conclusions

This thesis aimed to answer the following main research question:

> *How can we obtain robust schedules for the stochastic Resource-Constrained Project Scheduling Problem (RCPSP) using probabilistic inference, without modifying the underlying scheduling algorithm?*

This thesis demonstrates that robust schedules can be inferred for stochastic RCPSP instances with black-box schedulers and simulators using probabilistic programming. The key contributions are:

- A method that treats both the scheduler and simulator as black-box components.
- No knowledge or analysis of the created schedules is needed.

- A generative model and inference setup in Gen.jl that integrates scheduling, uncertainty modeling, and evaluation.
- Validation of the method on a small example instance where the known robust schedule was successfully recovered.
- The possibility to apply the method to any scheduling domain, provided the needed scheduling and simulator programs exist, to find robust schedules.
- The possibility to define a custom robustness measure, given that the simulator can be used to calculate the given robustness.

7.3 Future work

There are many trivial potential avenues for future research. First, robustness was defined here as the success rate of a schedule completing all tasks before their deadlines, under uncertain task durations. Depending on the user and use case, a robust schedule might hold different properties. For example, these could entail a lowest expected makespan or a lowest worst-case delay. This could be useful in high-risk projects that are expensive if they are delayed. This can be researched further to see what robustness measures are possible to optimize.

Second, uncertainty could be introduced not only in task durations but also in other aspects of the RCPSP, like resource availability. For instance, the number of machines (or employees) available might be uncertain. A robust schedule in such a context may aim to minimize peak resource usage to remain feasible under more variable conditions.

Third, future work should be done on testing this method on larger and real-world RCPSP instances. This paper successfully applied the method to a small example, but larger or real-world schedules remain untested. The performance of the proposed method can be explored on those larger problems to see if robust schedules can be found.

References

[1] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 221–236, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314642. URL `http://doi.acm.org/10.1145/3314221.3314642`.

[2] Fang Fu, Qi Liu, and Guodong Yu. Robustifying the resource-constrained project scheduling against uncertain durations. *Expert Systems with Applications*, 238:122002, 2024. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2023.122002. URL `https://www.sciencedirect.com/science/article/pii/S0957417423025046`.

[3] Reuben Gardos Reid. Inferring Robust Plans with a Rail Network Simulator. Master's thesis, TU Delft, Delft, July 2023.

[4] IBM. Ibm academic initiative, 2025. URL `https://www.ibm.com/academic/`. Accessed: 2025-06-10.

[5] *IBM ILOG CPLEX Optimization Studio Userâs Manual.* IBM Corporation, Incline Village, NV, 2017. Used version 12.9.

[6] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144, 2017.

[7] The Netherlands Organisation for Scientific Research (NWO). Netherlands code of conduct for research integrity. `https://www.nwo.nl/en/netherlands-code-conduct-research-integrity`, 2018. Accessed: 2025-06-20.

[8] Surya T. Tokdar and Robert E. Kass. Importance sampling: a review. *WIREs Computational Statistics*, 2(1):54–60, 2010. doi: https://doi.org/10.1002/wics.56. URL `https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.56`.

[9] Jan-Willem Van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

[10] Kim van den Houten, David M. J. Tax, Esteban Freydell, and Mathijs de Weerdt. Learning from Scenarios for Repairable Stochastic Scheduling. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 234–242, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-60599-4. doi: 10.1007/978-3-031-60599-4_15.

[11] Kasper van Duijne. Finding robust schedules in the stochastic resource constrained project scheduling problem using probabilistic inference, June 2025. URL `https://doi.org/10.5281/zenodo.15716001`.

[12] Peter M. Verderame, Josephine A. Elia, Jie Li, and Christodoulos A. Floudas. Planning and scheduling under uncertainty: A review across multiple sectors. *Industrial & Engineering Chemistry Research*, 49(9):3993–4017, 2010. doi: 10.1021/ie902009k. URL `https://doi.org/10.1021/ie902009k`.

[13] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. *Experimentation in software engineering*, volume 236. Springer, 2012.

[14] Farnaz Torabi Yeganeh and Seyed Hessameddin Zegordi. A multi-objective optimization approach to project scheduling with resiliency criteria under uncertain activity duration. *Annals of Operations Research*, 285(1):161–196, February 2020. doi: 10.1007/s10479-019-03375-z. URL `https://ideas.repec.org/a/spr/annopr/v285y2020i1d10.1007_s10479-019-03375-z.html`.

A Tools

A.1 Spellcheck

During the writing of this paper, Overleaf's built-in spell checker was used to correct any misspelled words.

The tool Grammarly (https://app.grammarly.com/) was also used as a browser plugin. This was also used as a spell checker.

A.2 Generative AI

The following prompts were used on ChaptGPT's 4o model. (https://chatgpt.com/):

1. On June 10th 2025, the current version of the paper was attached with the following prompt:

   > "Give me a list of spelling and grammar mistakes in the following paper."

   After which, the list of mistakes was manually looked at for spelling/grammar. Mistakes were fixed where necessary.

2. On June 22nd 2025, the current version of the paper was attached with the following prompt:

   > "Give me a list of spelling and grammar mistakes in the following paper."

   After which, the list of mistakes was manually looked at for spelling/grammar. Mistakes were fixed where necessary.