Resource Constrained Business Process Simulation

Michal Baczun

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2022

Abstract

Business Process Simulation (BPS) is a powerful technique for gaining insight into existing or proposed business processes. In order to maximise the quality and accuracy of the metrics gathered from BPS, we need the simulated workflow and the execution environment to reflect the real world as accurately as possible. The *resource perspective* is one aspect of BPS which is concerned with how resources are defined and used by the workflow, and an area that is often overlooked. Resource constraints have the potential to model complex and intricate properties of real-world resources that could be used to simulate scenarios which were not previously possible, and yet most BPS tools only support basic resource constructs.

Proter is a unique discrete event simulator with prioritisation at its core. Its priority-based approach to scheduling a simulated workflow enables the creation of models that more accurately represent how people schedule tasks in reality. This project implements new resource constraints such as resource capacities to further improve the freedom and flexibility of modelling afforded by Proter. To do this the project also identifies a list of core resource constraints relevant to BPS as part of a review of BPS tools and literature.

One concern is that there are multiple ways to schedule tasks based on the different resource and priority constraints. When we introduce more constraints, the choice of scheduling strategy becomes an important consideration as different strategies can produce drastically different results. This project provides a framework for evaluation built around random workflow generation, which is used to compare the baseline scheduling strategies supported by Proter to show how choice of strategy can impact key performance indicators. By providing the necessary tools and demonstrating the impact of scheduling strategy choice, the project aims to promote comparative evaluation of strategies as an important practice in BPS studies.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Michal Baczun*)

Acknowledgements

I would like to thank my supervisor, Dr Petros Papapanagiotou, for his invaluable guidance and mentorship throughout this project.

I would also like to thank my family for their support and motivation.

Table of Contents

Chapter 1

Introduction

One of the most common applications of computing is simulation, dating back to uses in mathematics and physics since the 1940s [7]. It has become such a prolific and widespread tool because it enables the experimentation, examining, and understanding of complex systems in a manner which is quicker, more flexible, and more easily reproducible than equivalent real-world experiments. In many cases we can use simulation to examine scenarios which are too expensive, dangerous, or outright infeasible to try in reality.

The caveat to a successful simulation is developing a sufficiently detailed model for the phenomenon being studied. If an under-simplified model is used then the observed results may not be accurate to the real world. For instance, if studying the firing of a neuron, a simple voltage and resistance model may suffice (see Leaky integrate-and-fire [1], dating back to 1907!), but such a model would be insufficient for studying ionic current across the neuron (see Hodgkin–Huxley model [11], 1952). Using a model with an insufficient level of detail is one of the most common pitfalls of simulation [27].

This thesis concerns simulation within the discipline of **Business Process Management (BPM)**. BPM includes various activities such as modeling, measurement, optimisation, discovery, and automation of operational business processes in support of enterprise goals [28, 19]. **Business Process Simulation (BPS)** is a powerful tool within BPM, and can used by BPM professionals for analysis and optimisation of business workflows. BPS is versatile, it can be leveraged to gain insight into existing operations by mirroring how a process works in reality to identify bottlenecks and make predictions, or it can be used for guidance, support in decision making, and evaluating possible "What-if" scenarios.

As with other types of simulation, the quality of the model used for BPS has a large impact on the result. It is paramount that BPM professionals can sufficiently express various requirements and constraints involved in business processes to study their impact in detail. Unfortunately many of the existing BPS tools have a simplified approach to modelling various constraints, and lack the ability to express many necessary details of real business processes. The literature, too, tends to focus on specific aspects of simulation such as workflow patterns, and does not place enough consideration into the importance of resource constraints and scheduling choices.

The current project identifies and demonstrates the importance of resource constraints in business processes simulation, and shows the underappreciated impact of scheduling strategy on simulations when introducing additional resource requirements. The project creates a framework used for evaluating scheduling methods, and builds on a powerful simulator, Proter [23], to implement new resource constraints into an existing system and evaluate their impact on schedule strategies.

1.1 Motivation

BPS is a powerful technique that can be used to gain insight into an existing or proposed business processes. By defining a workflow, consisting of some ordering of tasks which require some time and resources to complete, we can extrapolate into the future to study the how the process performs: Studying how the workflow could be executed in practice given limited resources, or the organisation's capacity to deal with such workflows. To be effective, we need for this workflow to reflect the real world as accurately as possible. The level of realism and abstraction available to us will directly influence the accuracy of the model and the quality of our simulation results.

Prioritisation is one aspect which is overlooked by most business process simulators, but it is a core feature of our simulator, Proter [23]. In reality we tend to assign priorities to tasks, even if this is sometimes implicit. For example if a VIP client walks into your store then you might try to serve them as soon as possible, potentially ahead of others. As a surgeon, if you know that a patient in critical condition will arrive on an ambulance in half an hour, you would not start a new operation that will take 3 hours to complete, but instead, to avoid downtime, you might take a few minutes to check on your other patients. This is what Proter does, instead of a greedy approach where as many tasks are started here-and-now, it puts the higher-priority tasks first and ensures these are not delayed in favour of lower-priority ones.

Proter's support for priorities gives users an additional degree of freedom to express how their process works and how it should be simulated. This can be leveraged to create workflows which are closer to the real world, and we hope to make Proter accessible for BPM professionals and researchers so that they can effectively utilise priorities in their simulations.

In a similar vein to priorities, the resource perspective is another commonly overlooked aspect of BPS. The way in which various BPS tools let users define resource requirements tends to be basic and limiting, preventing the creation of workflows which reflect the constraints of their real-world counterparts. For instance, if we consider a surgeon as the resource required to complete a surgery, we may want to express that we have a pool of surgeons available, and we can choose any one of them for the job. What if the surgeons are on a rota, and they are not all available at any given time? Or what if we require two surgeons simultaneously for some surgeries, but not others? Even such basic constraints on the workflow are not supported by the majority of BPS tools, which prevents BPS studies from accurately reflecting business processes.

This project aims to identify resource constraints which are relevant for business process simulation, and begin supporting some of these within Proter, enabling researchers and BPM professionals to create more accurate models. In supporting more resource constraints, the choice of scheduling strategy becomes increasingly less trivial, so researchers also need a way of choosing a scheduling strategy by evaluating relative performance. This is made difficult by the lack of any sizable business process datasets in this field of research (a problem discovered during previous work in part 1 of this project). Many BPS tools support completely different sets of features, and even within a single tool such as Proter, adding new features such as a new resource constraint would make any previous existing datasets insufficient. To resolve this problem, the project designs and deploys a pipeline for extensive evaluation of schedulers in a way which does not rely on large existing datasets.

1.2 Previous work carried out

In the previous part of this honours project I worked to identify key criteria for business process simulators according to existing literature. There is no standard set of such requirements, so in the project I proposed the identified list as a standard against which we could evaluate all BPS tools. This list also revealed a number of essential capabilities missing in Proter, most notable were support for arrival rates and business process modelling notation (BPMN), which were then implemented. With the newly identified criteria I conducted an evaluation of some popular BPS tools alongside Proter, and discussed relevant differences in how these tools performed on a small handmade set of example workflows.

That work revealed many shortcomings related to the “Resource Perspective” of BPS tools and noted the misleading terminology used to talk about resource constraints, inspiring the work in this second part of the honours project. In implementing aspects of the resource perspective I continue to bring Proter closer to the “Gold Standard” proposed by my original list of key criteria, and in so doing I discover and discuss the difficulties associated with supporting a growing number of resource constraints, particularly related the impact of scheduling strategy on the simulation. The various resource constraints identified in this thesis can also be viewed as an extension to the original list of criteria.

1.3 Methodology

First this project collects a set of resource constraints which are important to BPS, based on features supported by other simulators and discussion surrounding the resource perspective in BPS literature. The resource constraint definitions in the literature are sometimes vague and contradictory, so the project aims to provide clear definitions and a coherent motivation for every point.

Some of the identified resource constraints are then implemented in Proter. This has the goal of enabling more intricate simulations in Proter which are better at reflecting real world scenarios.

In implementing resource constraints we also need to update the schedulers in Proter to work with the new features. This introduces the growing concern that more and more autonomy is being placed on the scheduling strategy to make important decisions about how the workflow is simulated. As such, the decision regarding which strategy to use in a simulation becomes more important, but evaluating the impact of various scheduling strategies is tricky because there are no real simulation model datasets in this field - a problem we also encountered last year in part 1 of the project. In this thesis we introduce an alternative framework for carrying out such evaluations using random workflow generation, and this approach is used to evaluate the differences between baseline schedulers available in Proter as well as the impact of introducing more complexity into the model on the simulation results.

1.4 Project Goals

In summary of the above, the goals of this project are to:

1. Identify and motivate a set of resource constraints relevant to BPS (Chapter 3).
2. Extend the Proter simulator by implementing support for additional resource constraints and extend existing scheduling strategies to support these constraints (Chapter 4).
3. Define and implement a scalable framework for evaluating simulation metrics within Proter (Chapter 5).
4. Evaluate the effect of different scheduling strategies before and after implementing additional resource constraints (Chapter 6).

Chapter 2

Background

This chapter serves as a very brief introduction to some of the relevant background for this project, including business process simulation, some information about the Proter simulator, and the wider context of some constraints (particularly resource constraints) found in closely related and widely studied scheduling problems.

2.1 Business Process Simulation

Business Process Management (BPM) is a discipline which includes various activities such as modeling, discovery, automation, measurement, and optimisation of operational business processes in support of enterprise goals [28, 19]. There exists a number of BPM tools and software to support such activities, but BPM is fundamentally a set of methodologies for managing and transforming business operations while the software is considered auxiliary [10]. One method which is particularly useful in this field is **Business Process Simulation (BPS)**: BPS can enable BPM professionals to analyse and optimise business workflows by simulating many potential scenarios for "Whatif" analysis, in support of decision making, to identify bottlenecks in a system, or for prediction, making it a very versatile and useful technique.

BPS refers to discrete event simulation of business processes within the field of BPM. In BPS we are interested in simulating workflows which consist of tasks that take some time to complete and potentially require the use of certain resources. The advantage of BPS over alternatives such as Markov Chains is that it tends to be flexible, easy to understand, and capable of answering a wide variety of questions [27].

In BPS we typically have a workflow which we want to simulate. A workflow consists of a number tasks which are linked together to describe the order in which they should be executed. Tasks usually have additional properties such as duration and cost. It is common for BPS tools to use **Business Process Modeling Notation (BPMN)** [17] diagrams to describe the order of tasks. Additionally, the simulation environment usually has a number of resources. Depending on the BPS tool, resources might have capacities, availability schedules, or other features which comprise the "Resource Perspective". Tasks typically require one or more resources before they can be run, and a resource is assigned to a particular task for the duration of that task.

A possible example of this kind of BPS workflow is a surgery: The surgery itself is the task that needs to be carried out, it will require some duration of time to be completed. It also requires a surgeon, which is a resource in this environment. After this surgery is complete, the surgeon becomes free again and can be assigned to another task.

2.2 BPS Tools

This section is partially a review of previous work where we evaluated a number of BPS tools against Proter. The features found in BPS tools are also relevant to this project because they provide an insight into how resources tend to be implemented.

There exists a wide selection of BPS tools, with varying capabilities and target users. A BPS tool is usually based on a modeling notation like Petri nets or BPMN. BPM tools are of particular relevance in this project, but some popular "general purpose" process simulation tools have also been considered here.

In the literature, the term "general purpose simulator" tends to refer specifically to *CPN Tools* [14] and *Arena* [15], a term seemingly used distinguish them as tools made for use in many disciplines outside of BPM, and as such they do not use BPMN. CPN Tools is a general purpose coloured Petri net [13] framework with simulation functionality and Arena [2] is a discrete event simulator which uses its own modelling language. Both are widely acknowledged as being strong simulation tools with good modeling and simulation functionality, but are sometimes critiqued as being hard to model with and "profound knowledge" of their modelling solution [12]. For this reason the more familiar BPMN-based BPS simulators are often preferred for BPM applications [22].

There exist many BPS tools. Some of the most known tools include Adonis [9], BIMP [18], Bizagi [4], Bonita [6], and Visual Paradigm [16]. The tools listed here, like most BPM simulation tools, are BPMN-based. There are many differences in the capabilities of these tools [21, 22], but in general they tend to lack some of the features of the general purpose simulators, such as replications and built-in support for confidence intervals. The biggest advantage that these tools have over the general purpose tools like Arena and CPN is that they are easier to use and do not require "profound knowledge"[12] since they use BPMN. Unfortunately many of these tools are not open-source and do not have free versions, making them inaccessible for research and comparative evaluation. The common pattern is that these tools tend to originate as BPM tools which are extended with simulation capabilities, as opposed being purpose-made simulators [21], and this can result in poorer support for criteria like replications or resource roles.

We used some of these tools in past work to compare against Proter. One noteworthy observation is that all these tools, to our knowledge, only use a greedy first-come first-served strategy for scheduling tasks. This means that tasks always get started as soon as they can given various constraints such as limited resources. Because the approach is greedy, it is sometimes possible for a particular workflow to have a better solution (one that completes quicker) had the tasks been scheduled in a better order.

2.3 Proter

Proter is a powerful tool for discrete event simulation, with novel features that allow a more realistic simulation of business processes. Unlike other simulators, Proter is centered about the importance of priorities in simulation. In the real world activities have varying priorities which may be explicit or implicit. For instance, if a manufacturer receives a large and expensive order, it is likely that this order would be prioritised such that it is handled smoothly and in good time, even though at a surface level the activities might appear to be identical to those related to lower-priority orders. Prioritised scheduling of tasks is at the core of Proter, enabling the modelling and simulation of such prioritised workflows, resulting in simulations which more closely match the real world.

In addition to this, Proter supports opportunistic ad-hoc scheduling which enable workflows to change dynamically during the course of the simulation. This can be leveraged to change the course of a workflow in reaction to events like delays, or to model random rare occurrences such as breakdowns.

The Proter simulator was originally created as part of the **WorkflowFM** platform. WorkflowFM is a logic-based framework for formal process modelling and composition [20], and it consists of many subsystems ranging from the workflow modelling tool to a business process management dashboard.

In past work [3], we identified a set of important BPS criteria, expanded upon Proter to ensure that it supports the most essential of these simulation capabilities such as arrival processes and BPMN support, and evaluated Proter alongside existing tools to show it is equally capable in terms of these standard features, and also to highlight that many of the existing tools support vastly different features for modelling and simulating workflows. It became apparent that the way in which resources are handled varies wildly between BPS tools, and also that there is room for improvement within Proter to allow richer expression of the resource perspective. By improving how resources can be expressed and handled within a simulation it will be possible to simulate more nuanced and intricate workflows, but making such changes while still adhering to Proter's central pillar of prioritised task scheduling makes this a novel and challenging problem.

An important part of Proter is how it schedules tasks. Just because a task is able to start in a workflow does not mean it can actually be started, for instance if two tasks are both waiting to begin but they both need to share the same resource. The scheduler is a component of the Proter architecture which makes decisions about which tasks can start next. Since the scheduler has the power to make decisions about what tasks should start and which ones should be delayed to be handled later, there are actually many possible strategies that can be used.

In Proter by default there are 6 possible scheduling strategies that can be used:

1. The standard one is the "Proter" scheduler, which looks at the priority of a task to decide when it begins. If starting a low-priority task right now would result in delaying a higher-priority task, the Proter scheduler will instead choose to postpone starting this low-priority task until later.

2. The “Lookahead” scheduler works similar to the Proter scheduler, but it additionally tries to look ahead into the future to see what tasks will arrive when the currently ongoing tasks finish, and then recursively checks the tasks after that until it reaches the end of the workflow.
3. The “GreedyFCFS” strategy starts all tasks whose resources are currently idle, in order of arrival in the queue. It ignores priorities.
4. The “StrictFCFS” strategy is similar to the above, but it forces tasks to execute strictly in the order of arrival. This means it will reserve resources for tasks which are at the front of the queue, and tasks at the back might be blocked from starting even if their resources are free.
5. The “GreedyPriority” scheduler starts all tasks whose resources are currently idle, in order of priority.
6. And lastly the “StrictPriority” scheduler, which is again similar to GreedyPriority but it guarantees that tasks will be started in order of priority. A lower-priority task might be blocked from starting even when all its required resources are free, because the resources might be being held for a higher-priority task.

The StrictPriority strategy may seem similar to the Proter scheduler, but the difference is that Proter maintains a timetable for each resource and is capable of identifying if there are windows of time where a low-priority task can run without disturbing any high-priority tasks. StrictPriority will always block lower-priority tasks if it knows a high priority task is coming up, but Proter might start the lower-priority task anyway, knowing that it will finish in time before the high priority task is ready to start.

2.4 Scheduling Problems

The challenge of simulating sequences of tasks with various resource requirements can be likened to a variety of scheduling problems, some of which have been studied for over half a century. Different scheduling problems employ various constraints and assumptions about the permissible sequences of tasks and resource assignments, but they usually have the goal of finding an optimal sequence of tasks given various constraints which minimises the makespan of the sequence.

Below we discuss two of the scheduling problems which are most similar to the scheduling problem faced in BPS.

2.4.1 Job Shop Scheduling

The job shop scheduling problem is one of the oldest scheduling problems that is applicable in business process simulation.

This problem consists of a set of M different machines that perform operations on a job, and a set of N jobs that needs to be processed through the machines. Each job consist of a set of operations which need to be processed on a specific machine and in a specific order, which takes some processing time.

Job shop scheduling has many related variants, such as Flow Shop Scheduling which restricts each operation in a job to being executed in a specific order and on a specific machine, or Open Shop Scheduling in which the order of operations in a job is unconstrained and so operations can be completed in any arbitrary order.

This scheduling problem is closey related to business process simulation. It provides a solid foundation for reasoning about scheduling tasks which require resources, making it a good starting point for BPS, but it is insufficient for expressing certain resource features. For example machines are dedicated, meaning one operation occupies an entire machine all to itself, making it unavailable to be used by other operations, whereas in BPS it is often desirable to express machine capacities, or the available quantity of a resource (for instance if there are 4 identical laser cutters in a workshop and any of the 4 machines can be used for the task). To this end there exist extensions of the problem such as Flexible Job Shop Scheduling, which allows an operation to be processed by any machine from a given set, but this extension still has limiting restrictions, most notably that a task only uses one resource.

2.4.2 Resource Constrained Project Scheduling Problem

More recently the **Resource-Constrained Project Scheduling Problem (RCPSP)** has become a popular alternative problem definition to the Job Shop Scheduling family, which alleviates the resource limitations described above. The mathematical model for the RCPSP was first formalized in 1969 by Pritsker et al. [25], but the standard unified notation used today was introduced 30 years later by Bucker et al. [8]. The standard RCPSP belongs to a class of strongly NP-hard problems, as shown by Blazewicz et al. [5].

In its basic form the problem consists of a set of J activities, $1,...,J$, each with a processing time p_j. An activity cannot be interrupted once started, and additionally constraints can be placed on the order in which activities must be carried out using precedence constraints. This is commonly given as a set of immediate predecessors of a task which must be completed before the next task can be started. Alternatively a graph can be used, where single precedence constraints are denoted by as edges. This activity-on-node network is assumed to be acyclic.

There is a set of K renewable resources, each one with an availability R_k per time period. Renewable means that the same specified number of units of a resource are available at every time period before being divided among activities. Each activity may require some number of various resources to be completed, and for the entire duration p_i of an activity i, all the required units of resource k must be allocated to i.

The objective of this problem is to minimise the makespan by finding a sequence of start times of activities that leads to the earliest possible completion of the project which satisfies all of the above constraints.

There exists a wide variety of approaches to finding solutions for the RCPSP, including stochastic methods such as markov chains, numerical methods such as dynamic planning, and most prominently in recent years we see a number of meta-heuristic approaches including artificial immune systems, ant colony optimisation, and genetic algorithms which remain the de-facto approach for finding optimal solutions.

The downside of methods such as genetic algorithms is that they are slow and require many hours of processing time to evaluate hundreds of generations of genomes in search of the optimum answer. This is problematic for BPS use-cases, especially in tools such as Proter in which quick ad-hoc simulation is desirable, and where it is common to make changes to the model and re-simulate for purposes such as live decision support, or in use cases where the simulation is frequently updated to reflect the state of a real system in order to provide a simulated preview of the upcoming minutes or hours of an environment. In such cases we need faster solutions which complete in a matter of seconds, potentially delivering ”good enough” solutions as opposed to optimal results.

A further problem with using RCPSP approaches in simulation is that it assumes absolute knowledge of the entire workflow from the get-go. In reality future tasks are not always certain, and with tools such as Proter the ad-hoc nature of simulated workflows means that everything could change as a result of a delay or malfunction. In a standard RCPSP solution, such unexpected events would require that the entire schedule is re-calculated, which again may take a very long time.

As with the Job Shop Scheduling problem, the RCPSP provides a good baseline for reasoning about the structure of tasks and resources in BPS, but this problem does not entirely match the goals of simulators such as Proter, and does not express all aspects of the resource perspective in which we are interested. There is an improvement in expressive power when compared to Job Shop Scheduling in that it can express multi-resource requirements with renewable resource capacities, but there is still no way to express resource roles or availability of resources. The next chapter delves further into BPS literature to identify a list of relevant resource constraints which are desirable for creating accurate and descriptive workflows.

Chapter 3

The Resource Perspective of BPS

As discussed in the previous, there are a variety of BPS tools available for general use, but these tend to support vastly different capabilities for modeling and simulation. The resource perspective, which encapsulates a number of features relating to how resource constraints can be expressed with the given tools, is an area which is particularly lacking in most tools, and an aspect of simulation which is frequently overlooked [27]. The existing literature surrounding the resource perspective is also sparse, but we can turn to closely related fields such as job shop scheduling for help in defining standard resource constraints. This chapter aims to define and motivate a list of relevant resource constraints which are beneficial to BPS.

3.1 Resources in BPS

As defined in Chapter 2, a workflow consists of a number of tasks, where each task might require one or more resources to complete. A resource may be anything ranging from a tool or machine required to complete the task, to a human actor (for instance a qualified operator of the machine). Unfortunately, the way in which these resources can be expressed within existing tools tends to be limited.

Through existing literature we can evaluate the resource perspective supported in existing BPS tools. There is no consensus within the literature on which resource constraints should be evaluated, and to our knowledge only a handful of reviews exist which even consider resources, but these provide a good starting point.

The first review, by M.H. Jansen-Vullers and M. Netjes [12], is largely focused on evaluating the usability of tools and the way in which they present results, for instance a focus on simulation replays and animations. There is relatively little regarding evaluation of the tools' technical capabilities. Regarding resources, the paper states "the process model should include the resource and data perspective", without going into detail about what this means.

Peters et al. [22] surveys a number of BPS tools as part of their work on a prototype. This work is very useful and has a detailed breakdown of some of the most common resource constraints, namely Capacities, Roles, Schedules, Cost of Usage, and Multiple Roles.

They also define a set of advanced constructs which offer an interesting perspective into the way resources are utilised in simulations, but these seem to be chosen specifically to bolster their prototype, and they remain as the focus for the rest of the paper.

The terminology used for some resource constraints by Peters et al. can be misleading and conflicts with our own definitions which are listed at the end of the chapter. Specifically Roles and Multiple Roles need to be defined in more detail: In this survey Roles refer purely to the fact that distinct classes of resource exist, in other words that the simulation has more than one type of resource and that a task can specify exactly which type of resource it needs. It is strange to have this distinction, because a tool which does not implement roles would essentially not support resources at all, so for this reason we believe it is clearer to simply refer to this construct as "Resource" instead of "Role". In combination with resource capacities, we can view a Role as a bucket, where the capacity described the number of resources in each bucket, and when a task requests a specific Role then a single resource is removed from the bucket for the entire duration of the task, similar to the RCPSP which we defined in Chapter 2. The "Multiple Roles" category refers simply to the fact that a task can request two or more different resources, instead of being limited to just one. For instance, this feature would enable a task to require both a machine resource from the pool of machines, and a worker resource from the pool of workers to operate the machine. For clarity, this thesis will instead refer to these resource constructs as Single-Resource and Multi-Resource assignment, and the term "Roles" is reserved for a different concept.

The final survey we review here, by J.L. Pereira and A.P. Freita [21], also has relevant section on the resource perspective. It includes categories for Capacity, Allocation Plan, Unavailability, and Schedule. Unfortunately these categories hardly explained, and are not cited. Capacity and Schedule categories are fairly common and consistent with other papers, but Allocation Plan and Unavailability are hard to interpret. Unavailability is only described with as "Definition of unavailability periods for resources", but the definition of Schedules in the literature is essentially identical (and the paper defines Schedules as "Definition of work schedules for resources"), so unfortunately I am not able to discern the difference given these definitions. In other literature such as the Survival Guide [27], it is commonplace to define resource schedules as periods of unavailability, for instance to model a worker which takes lunch breaks and goes home in the evenings.

Two of the above reviews lean on the influential paper on resource patterns by Russel et al. [26], which is concerned with identifying resource patterns in workflows and support for these patterns within workflow systems. The identified patterns are grouped into a number of categories, but the focus is generally on defining how tasks are assigned to resources as opposed to how different properties of a resource might be modelled. For instance, the paper defines "push patterns" where the system offers or allocates tasks to be handled by each resource, versus a category such as "pull patterns" where task allocation is resource-initiated. The paper is insightful for thinking about the resource perspective from the angle of how allocation is handled in systems, but the target is workflows systems rather than simulators, and so many of the allocation patterns defined here are not applicable in this project.

3.2 List of Resource Constraints

Compiling the relevant features of resources from the above literature gives a more complete perspective on resources. This set of resource constraints is shown below, with clear definitions of each constraint. The list additionally includes some novel ideas which to our knowledge were not present in the literature nor in any existing BPS tools, yet describe common aspects of resources in real world scenarios. The function and motivation for these novel items is also explained below.

- **Single Resource Assignment** - Tasks should be able to require a resource in order to run, where the resource is assigned to said task for the duration of the task. This is the most basic resource constraint, for example a surgery task which requires a specific surgeon to be assigned to perform the surgery.
- **Multi-Resource Assignment** - Tasks can require multiple resources of different types. This could be a surgery task that requires two resources: A specific surgeon, and a specific operating room.
- **Resource Capacity / Quantity** - Resources have a capacity or quantity, so that they can handle multiple tasks at the same time, for instance a single oven resource which has room for 2 baking tasks. This has advantages over modelling the two shelves of the oven as separate resources: We can say that our baking task just requires any oven shelf and we don't mind which one is used, instead needing to commit to a specific shelf for this task when we create the workflow. With capacities, if shelf 1 is occupied the task can resort to using shelf 2; Without capacities, the task has to wait for shelf 1 to become free, which is wasteful.
- **Multi-Capacity Assignment** - Resources have a capacity or quantity, and tasks can require more than one unit of a resource. This builds on from the previous point, enabling tasks to request multiple capacity slots from a resource, such as a single baking task which is allowed to use both shelves of the oven resource simultaneously.
- **Resource Schedules** - Resources can become unavailable at certain times, at which point they cannot be used to complete tasks. This also comes with the question of what happens to tasks when a resource becomes unavailable - can tasks be interrupted, can they be restarted or are they continued once the resource becomes available? This constraint can model a worker's shifts, or routine downtime of a machine.
- **Resource Efficiency** - Depending on the choice of resource, a different amount of time may be needed to complete the task, for instance one surgeon which is experienced and performs the surgery much faster than another surgeon.
- **Variable Resource Efficiency** - Resources that can become more or less productive based on any number of factors such as time of day, remaining capacity, or uptime, for instance a worker which becomes tired and less productive over the course of the day.
- **Resource Roles** - Instead of requiring a specific resource, tasks require a certain

role to be completed, and multiple resources can fulfill the same role. For example, we could have a surgery which requires a surgeon resource, but there are multiple doctors that can fulfill the surgeon role and the system can choose any one of them. On the surface this might seem similar to having a single "Surgeons" resource which has a capacity of *n* surgeons, but this is different because the surgeons are unique: They can have different efficiency or availability from one another.

- **Multiple Roles per Resource** - Resources can implement any combination of the existing roles, for instance a scenario where one medical practitioner can serve as a surgeon or as a GP, while another can only do the GP role. In combination with previous points such as variable efficiency and resource schedules, it could become hard to choose which resource to use for a given task, for instance one strategy could be greedy while another could try to keep the resources that fulfill rarer roles free, like making sure the only person with the surgeon role is always available to perform an emergency surgery.

Note that some of the above points have overlapping criteria. For example, in a system which supports variable resource efficiency, resource efficiency also needs to be supported, because the prior builds on top of the latter feature. If a system supports resource roles but not multiple resource roles, then such a system is equivalent to one which only supports resource capacity: Instead of defining a resource with N capacity, we can define N resources which each fulfill the same role R, such that when role R is required by a task then there are N resources to choose from. This breaks down if we consider roles to be discrete but allow capacities to be continuous, for instance we can have a system which allows a resource to have 2.5 capacity.

Various BPS tools support different mixtures of the above capabilities, but to our knowledge features such as multiple resource roles are novel and are unsupported by any existing solution.

One reason why tools tend to only support a subset of this resource perspective is that scheduling becomes more difficult as more elements are introduced. Scheduling a task is simple when you only support single resource assignment - if the specific resource required by that task is available then you can start the task, if not you need to wait. Resource capacities are only slightly more difficult, just check if enough of that

Criteria	Job Shop	RCPSP	Bonita	BIMP	Proter
Single Resource Assignment	+	+	+	+	+
Multi-Resource Assignment	-	+	+	-	+
Resource Capacity	+	+	+	+	-
Multi-Capacity Assignment	-	+	+	-	-
Resource Schedules	-	-	-	+	-
Resource Efficiency	-	-	-	-	-
Variable Resource Efficiency	-	-	-	-	-
Resource Roles	-	-	-	-	-
Multiple Roles per Resource	-	-	-	-	-

Table 3.1: Resource perspective support across existing problems and tools

resource quantity is available and if so you can start the task. It is harder to schedule tasks optimally if you need to make a choice about which resource to use when they have different efficiencies, and harder still when resources have multiple roles: For instance suppose resource 1 can do roles A and B, and resource 2 can only do role A; If a task arrives which needs role A and you assign it to use resource 1, then another task arriving which requires role B will need to be delayed. Maybe resource 1 is more efficient though, and it is better to use it despite the delay? What if the second task only has a 50% chance of arriving?

Table 3.1 summarizes this resource perspective landscape by indicating what aspects are supported by the previously discussed scheduling problems, as well as showing what is supported by some existing BPS tools. Note that for the scheduling problems there tend to exist various extensions that support some additional resource criteria, but for simplicity only the standard general version is shown.

Chapter 4

Implementation of Resource Constraints

To enrich the simulations that are possible in Proter, this project originally aimed to implement as many of the resource constraints as possible out of the list outlined in Chapter 3. This list is already roughly in order of importance- the topmost features being those that are commonly discussed in literature and widespread among other BPS tools, while the bottom features were completely novel to our knowledge. As explained in Chapter 3, Proter already supported Single- and Multi-Resource assignment before this project, which means that tasks in Proter can express the need to be assigned to one or more resources in order to be executed. The next order of business was to add capacity to resources, and then multi-capacity assignment to tasks, which would enable a single task to not only request multiple resources but also a custom amount of capacity in each resource.

This project didn't implement further resource constraints beyond these two capacity-related points due to various challenges that arose in implementing these features and supporting them in an existing codebase, such as updating the various scheduling strategies in Proter to support the new resource system. A portion of the limited time and bandwidth for this project also had to be allocated to implementing code related to the evaluation framework, which is also discussed in the next chapter.

To implement capacity-related criteria this project introduced "Weighted Schedules" into Proter, and updated the functionality of the various schedulers to use these new weighted schedules. The design and implementation of this solution is detailed in the following sections.

4.1 Schedules

The Scheduler is an object within Proter which is used to run a simulation. This is a core part of the architecture that existed prior to this project, and it is relevant to the new features implemented in the next section. Whenever a simulation starts, or when some task finishes, the Scheduler is used to determine which new tasks are allowed to begin.

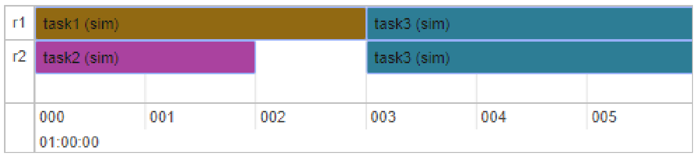


Figure 4.1: Example of a simple schedule in Proter

For instance, consider the example schedule illustrated in Figure 4.1. This schedule is the result of simulating a workflow consisting of three tasks, where tasks 1, 2, and 3 are all allowed to run in parallel. The figure shows how these tasks all require different resources, and task 3 even requires two resources simultaneously, visualised using an are which spans multiple resource lanes in the schedule.

Consider what would happen at time $t = 2$ during the simulation in this example: Task 2 just finished, so the scheduler is consulted about what new tasks are allowed to begin. Task 3 is the only task waiting to start. The scheduler will check if task 3 is allowed to begin by checking if all the resources needed for task 3 are available. It will discover that resource $r1$ is occupied, because $r1$ is still assigned to task 1 at this time, so the scheduler concludes that task 3 cannot begin yet. It returns an empty list of tasks. The scheduler will be queried yet again when task 1 completes at time $t = 3$, at which point it will find that all the resources needed for task3 are finally free, so task 3 will starts.

The system for checking if a resource is free or occupied is called the *schedule* in Proter. Before this project every single resource was assigned a schedule by the Scheduler. The schedule for a resource is simply a list of intervals during which this resource is occupied. Consider the example:

$$\{[0,2],[3,6]\}$$

This schedule represents two busy periods for a resource: This resource is busy from time $t = 0$ to $t = 2$, then it's unoccupied from $t = 2$ to $t = 3$, and then becomes occupied again until $t = 6$. This schedule would correspond to resource $t2$ in Figure 4.1.

This system is important in Proter, because it has advantages over a simple greedy method which does not make a schedule. In the greedy approach, we might be tempted to simply assign a Boolean value to each resource to express if it is busy right now. This does not work when we introduce priorities, a core feature of Proter. If we know that a high priority task is coming up in the future, we don't want to delay it with a lower priority task. For this reason, we need to schedule the high-priority task ahead of time, which is done using schedules.

Schedules are flexible because they still allow low-priority tasks to start, so long as they don't interfere with the high-priority tasks which were scheduled before them. In the ongoing example from Figure 4.1, consider what might happen if we introduce task 4 which has low priority and starts during the small gap in the schedule of $r2$, as soon as task 2 ends.

There are a few options for how things might progress, some of which are shown in Figure 4.2. The greedy approach, without schedules, would give the result on the left,

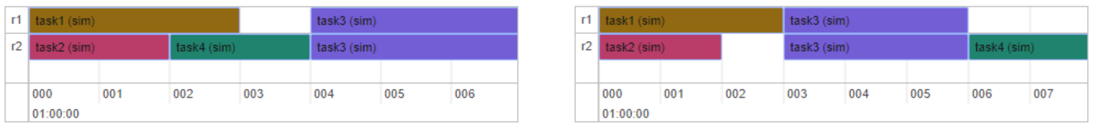


Figure 4.2: Two possible simulation schedules when task 4 is introduced

where task 4 is started as soon as the opportunity arises. This happens because the resource it required was free at the time, but it results in delaying task 3 by one time unit. Now, with schedules, the high-priority task 3 is already scheduled to start, and the gap at time $t = 2$ is not big enough for task 4, which is why it is scheduled at the end instead, at $t = 6$. There is a third option, not illustrated, which would happen if task 4 was even shorter than in the previous example. If it's short enough to fit in the gap at $t = 2$, it would be assigned to that spot at $t = 2$. In this case the scheduler would check the schedule of $r2$ against the expected duration of task 4 to discover that there is enough space remaining to deal with task 4 without offsetting task 3.

This scheduling system is what had to be modified to support capacity-related constraints. The existing schedules are binary; a resource is either completely occupied or completely free. Instead, as part of resource capacities, we need to quantify "how busy" a resource is. Weighted schedules were introduced to solve this problem.

4.2 Weighted Schedules

Now, in this project, we want resources have a capacity. This means that some resource r with capacity n is not just completely busy or completely free. There might also exists periods of time during which only part of the capacity, m, is being used, where $0 < m < n$. This project implements Weighted Schedules, which elegantly solve the problem by building on the ideas of regular ("binary") Proter schedules through the addition of a third value which represents how much of the capacity of a resource happens to be used during each time period.

Weighted schedules are a set of intervals with "weights": $WS_r = \{(s, e, w)\}$. Each interval is a 3-tuple: s and e are the start and end times of the interval, and w is the weight representing how busy the resource r is during this interval of time. For instance consider the following example weighted schedule which is also visualised in Figure 4.3, where the height of a bar represents how busy the resource is at that time:

$$\{[1,3,2],[3,5,1],[5,8,3]\}$$

Weighted schedules like this can easily be converted into the original binary schedules during task scheduling. To determine if a task is allowed to start we can take the schedule of each resource along with the max capacity of this resource and the capacity required by the task. For instance if the resource shown in the weighted schedule above and in Figure 4.3 has a max capacity of 3, and we have a task which requires 1 capacity from this resource, then this resource has enough space for the task during times where 2 or less capacity is being used. This is shown with the red line: During intervals where

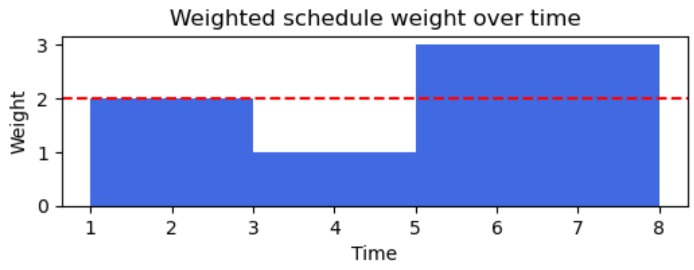


Figure 4.3: Weighted Schedule visualisation

the resource usage is above the line, the resource is too full for our task. Converting this example into a binary schedule will give the following interval, just by checking all the weighted intervals and only keeping the ones that are too full:

$$\{[5,8]\}$$

From here the original method of scheduling can continue by finding a window of space in the binary schedule which is long enough given the expected duration of the task.

There were some technical challenges during this implementation involving merging of multiple intervals. If two intervals are next to each other and they have the same weight, they need to be merged into one. For instance $\{[1,2,1],[2,3,1]\}$ is the same as $\{[1,3,1]\}$. This gets challenging when we need to add a new interval to an existing schedule, because intervals can also overlap in which case we need to add the weights. Consider the expected behaviour for the previous example in Figure 4.3 when we try to add a new interval such as $[3,6,1]$:

$$\{[1,3,2],[3,5,1],[5,8,3]\}+[3,6,1] \Rightarrow \{[1,5,2],[5,6,4],[6,8,3]\}$$

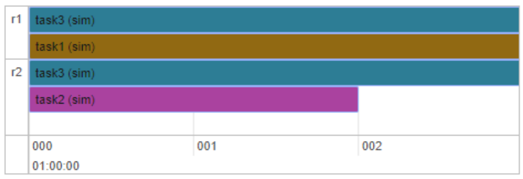
As you can see, all the old intervals were affected by the new addition. The first interval, $[1,3,2]$, didn't overlap with the new interval, but after the addition it was neighbouring another interval which has the same weight, so these needed to be merged. The middle interval, $[3,5,1]$, completely overlaps with the newly added interval, so at first the weights of the two are combined giving $[3,5,2]$, but this is then merged with $[1,3,2]$ as mentioned previously. The final interval, $[5,8,3]$, partially overlaps with the new addition, so it was split into two new intervals.

There exist other minor complications and edge cases along these lines when it comes to adding and merging intervals in weighted schedules. Unit tests were carefully made for all possible examples and the implementation had to be re-visited a number of times before everything worked.

4.3 Tasks, Resources, and Schedulers

Weighted schedules were the core solution for adding capacity constraints to Proter, but all the surrounding elements also had to be updated.

The Proter Scheduler, described in Section 4.1, is not the only scheduling method in Proter. Other prioritised and greedy methods are also supported, which we evaluate in



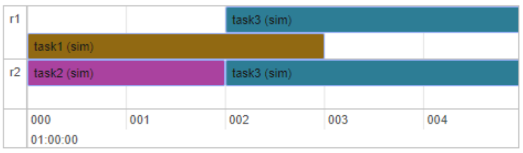


Figure 4.4: Two possible simulation schedules with capacity

more detail later in Chapter 6, and all of which had to be updated to work with weighted schedules and other changes related to the task and resource classes in Proter.

To support the *Resource Capacity* constraint, the resource class was updated. Since resources now have a capacity, multiple tasks can be assigned to a single resource at any one time, so methods related to scheduling and data structures related to storing allocated tasks had to be updated or expanded while remaining backwards-compatible.

To support *Multi-Capacity Assignment*, the task class also had to be updated to express required capacity for every one of the resources requested, while again maintaining backwards-compatibility.

Having implemented all of these changes, the *Resource Capacity* and *Multi-Capacity Assignment* constraints are now fully supported. This lets the user simulate workflows which were not previously possible. Consider the example with 3 parallel tasks introduced in Figure 4.1, but now using capacities we say each resource actually has 2 capacity instead. The timeline on the left of Figure 4.4 shows the result: With this change there's enough capacity for task 3 to start alongside tasks 1 and 2, and we don't need to wait to start task 3 like before. Now let's say that task 3 requires only 1 capacity from resource $r1$ just as before, but also 2 capacity from $r2$ instead of just 1. The right timeline in Figure 4.4 shows what happens: There's not enough capacity in $r2$ while task 2 is running, but as soon as it finishes task 3 can begin. Meanwhile $r1$ still has enough capacity to support tasks 1 and 3 simultaneously.

Implementing capacities was hard because they need to exist alongside other constraints available in Proter, such as priorities and multi-resource assignment. As more and more resource constraints are added to a system, they become harder to implement due to the different ways in which these constraints can interact. Another effect is that scheduling becomes less trivial as new constraints are added. For instance if we didn't have priorities then a greedy approach to scheduling might give the best result, but with priorities the Scheduler has to make a choice: should a low-priority task start now, risking delaying a high-priority task, or should it be delayed until later? New constraints might force the Scheduler to decide between high or low-efficiency resources, or to consider limited availability of a resource.

When choice of scheduling strategy becomes less trivial, it necessitates the ability for BPM professionals to evaluate various strategies in order to pick the best one. The next chapter discusses how this was designed and implemented in Proter.

Chapter 5

Scheduling and Evaluation Framework

As mentioned in the Introduction of this project, there is this problem that optimally scheduling a workflow becomes less trivial when more detail is introduced into the model. In a very simple BPS tool, if tasks can only be assigned to one resource, scheduling is trivial: There is only one way to assign the tasks and complete the simulation. When we add new features, for instance maybe the Scheduler can choose which resource to use given a few options, then there could be many equally compelling ways of scheduling a workflow.

Proter priorities already make scheduling a bit more challenging, because now there is a potential choice about whether we want to respect priorities by delaying low-priority tasks to start high-priority tasks as soon as possible, or if we want to be greedy and start all tasks as soon as possible irrespective of priority. We could even prefer some balance between these two extremes to achieve a good ratio between prioritisation and workflow duration. This is a choice that can be made by the BPM professional based on their requirements: They might be trying to reproduce the planning strategy used by an organisation in the real world, or to find a strategy which maximises some set of key performance indicators.

5.1 The Problem with Datasets

To make an informed decision about scheduling strategy we need a framework for comparison and evaluation of different schedulers. One standard approach might be to collect a large dataset consisting of many workflows which we can feed into the simulator to compare how different scheduling strategies perform. The problem, as discovered in part 1 of this honours project, is that there is no such dataset. For part 1 of the project a small set of handmade examples was put together in order to compare a small number of simulators, but this introduced another issue: Different BPS tools support different features, for instance only Proter supports priorities.

Making a dataset that worked across all the tools that we evaluated was hard, and in the end a number of the handmade examples did not work in some of the simulators because it was impossible to model them. Another issue is that such a dataset is only a description of the workflow, and each workflow still has to manually implemented with each tool.

Clearly using a dataset wouldn't work. We need to be able to simulate and compare performance on very large number of workflows, and even if an extremely large dataset was created it would become obsolete as soon as a new feature is added. We could have a dataset of hundreds of examples in Proter, but every example would need to be updated as soon as this project added any new resource constraint.

Instead, in this Project we propose a different framework for evaluating simulator performance, which involves programmatically generation of random workflows.

5.2 Random Workflow Generation

Instead of relying on a static set of handmade workflows, we can generate them ourselves. This allows for thousands of workflows to be made and simulated in a matter of seconds, solving the problem of limited examples. It also solves the problem of scalability: If the simulator is updated with a new feature, for instance resource capacities, then the workflow generator can also be expanded to enable the generation of workflows which include the new feature.

To our knowledge this approach is not used in any existing BPS tool. There exist some systems for generating random processes, for instance PLG2 [24] which creates randomised BPMN diagrams, but these are not enough for simulation because our models additionally require information about resource requirements, priorities, durations, and so on. In this project a system for random workflow generation was implemented in Proter to support the proposed evaluation framework. The generator enables studying the impact of various workflow properties on the simulation, for instance how different scheduling strategies fare as we implement new resource constraints, which is a topic that this project explores in Chapter 6.

The method of generating a random workflow uses Proter *Flows*. Flows provide an easy way to define a workflow by connecting tasks together in a binary tree structure. A typical flow consists of tasks connected by "And" (parallelism) or "Then" (sequence) operators, which are represented by $+$ and $>$ symbols respectively. As an example, Figure 5.1 shows a simple flow consisting of three tasks and the binary tree representation of this flow. In this example, tasks A and C are allowed begin right away in parallel, but task B can only start once A has finished.

$$(A > B) + C$$

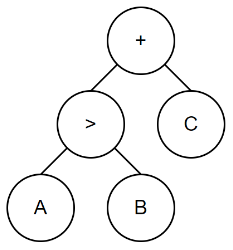


Figure 5.1: Example flow and corresponding tree

This project added a “Random Flow Factory” object which generates these flows given some configuration parameters. The object takes a number of probability distributions which describe properties such as the number of tasks, how many random resources each task uses, the duration, priority, and so on. By default Proter has support for constant, uniform, and exponential distributions, but others can easily be defined. The random binary tree is generated depth-first, recursively:

1. Initialise the root of the tree by specifying n, the number of tasks that this tree needs to include. n will have been drawn from a random probability distribution provided by the user.
2. If $n = 1$, this is a leaf node. Insert a random task at this point of the tree by sampling properties of the task such as duration and priority from the variety of provided probability distributions.
3. Otherwise, if $n > 1$:
   (a) This is a non-leaf node, so randomly choose an operator to place here ($+$ or $>$). The ratio between $+$ and $>$ operators is defined by the user, so we can roughly control how much branching vs parallelism is present in the generated flows.
   (b) n represents how many tasks still need to be placed. Split this number randomly between the left and right children of this tree node, such that each child receives $n \geq 1$. Repeat Step 2 and 3 for each child node.
4. The random flow is complete once all n random tasks have been created and placed in the tree.

5.3 Evaluation Framework

Random workflow generation is the core part of this proposed BPS evaluation framework. Using the system outlined above researchers can generate thousands of workflows while controlling various fine details of the workflows to target the specific research question being evaluated. A large number statistics can be gathered and exported for every single workflow to be analysed separately.

The general elements which should be supported by BPS tools to enable this kind of analysis need to involve:

- A method for generating workflows given a number of parameters, such as the Proter implementation described above.
- An API or other interface method enabling researchers to configure their experiments. Some BPS tools are completely closed-off systems which were only designed for manual human interaction. Proter allows programmatic access to all the relevant features making it possible to define the experiment in code, although the barrier to entry is high due to some required knowledge about the Proter system; Improving this interface is a possible future improvement.

- A system for outputting raw data. Many BPS tools only generate PDF reports or web-based summaries of the simulation, while the raw data is inaccessible.

In regards to the last point, Proter already had a “Metrics Handler” system which collected all the possible simulation metrics. For this project, a new type of metrics output handler was created which collects the metrics from many simulation runs into a handful of CSV files. This was necessary because previous output handlers would overwrite or make new CSV files for every simulation run, and the new handler lets all the results from thousands of random workflow simulations be collected together.

These newly implemented Proter systems finally enable us to test one of the core hypotheses of this project, which states that the implementation of additional resource constraints makes scheduling more difficult and thus careful choice of scheduling strategy becomes more important to the simulation performance. Using this framework, the next Chapter goes on to evaluate how different schedulers perform before and after adding the resource constraints implemented in this project.

Chapter 6

Evaluation of Scheduling Strategies

Throughout this thesis it is mentioned how the choice of strategy used for scheduling becomes more impactful on the simulation outcome when we add more complexity. In a very simple model, there might only be one way of possibly scheduling all the tasks, for instance if there is only one resource and we don't have priorities, then the single viable approach is to simply start a task as soon as the resource is free.

This project aimed to provide a more rich and expressive variety of possible resource constraints in Proter to allow for better simulations, but in doing so we make scheduling more difficult. Suddenly the scheduler has more choices to make, more options to weigh. Should tasks be delayed or started right away? Should this resource be kept available for emergencies (such as extra high priority tasks, like a surgeon on standby for an emergency surgery)? Which resource to choose? BPM researchers and professionals need to be able to make an informed decision about what strategy to use, whether it be an optimal method which maximises some key performance indicators, or a strategy which reflects the real world organisation as closely as possible.

The evaluation framework developed in Chapter 5 equips us with the tools to tackle these questions. Of course, the specific key performance indicators being evaluated will vary from one study to another. In this project we aim to convince you, the reader, that the choice of scheduling strategy has a tangible impact on simulations, especially as we add new features to the simulator, and that the study and comparison of various strategies requires careful consideration and is deserving of more attention in BPM studies and literature.

To be specific, this scheduler evaluation addresses two core questions:

1. How do different scheduling strategies perform over a large variety of workflows? This is a baseline evaluation which aims to definitively and quantitatively show that different strategies are better for different applications. We expect that prioritised strategies will handle priorities better than greedy methods, but that the greedy approach will give shorter workflow durations. Prioritised methods should be slower because they work by deliberately delaying low priority tasks.
2. How does adding new resource constraints impact schedulers? We hypothesise

that more resource features result in more freedom of choice by the scheduler, so the difference in relative scheduler performance across select key performance indicators should increase.

6.1 Baseline Scheduler Differences

Proter currently supports 6 schedulers, which were introduced in Chapter 2. To compare these schedulers, a few sets of results were collected by simulating 10,000 workflows and executing each workflow with every scheduler. Some of the core metrics we are interested in involve the makespan and “priority error” scores.

The makespan of a simulation is the length of total simulated time that it takes to complete the workflow. It should roughly scale as the sum of task durations increase, since it makes sense for the entire workflow to be longer if the tasks are longer. The ratio of makespan to the sum of task durations also depends on the amount of branching and parallelism in the workflow, because we can save time if many tasks can be executed in parallel while respecting resource constraints, reducing makespan.

The priority error is a metric designed to measure how good a particular solution is at respecting priorities. In a perfect simulation result, where every single task starts as soon as it arrives and there are no delays, the priority error will be zero. Such a perfect scenario is rarely possible, it is more likely that there are some conflicts in the workflow resulting from limited resources, so some tasks are inevitably delayed. The priority error is a metric which penalises delaying high-priority tasks more strongly than low-priority tasks. The error is calculated using the formula:

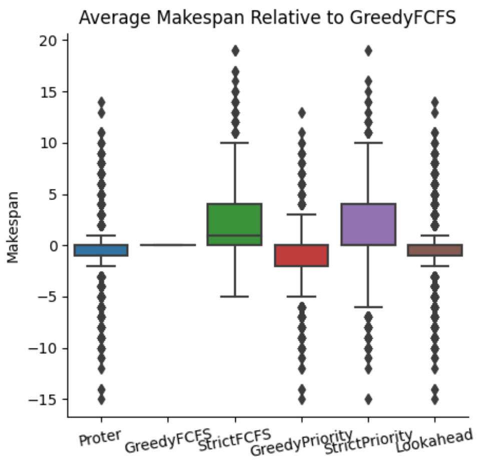
$$PE = \sum_t P_{(t)} D_{(t)}$$

This is a sum over all tasks, t, where $P_{(t)} \geq 0$ is the priority of t and $D_{(t)} \geq 0$ is the amount of time that task t was delayed. A higher-priority task t_{high} has a greater priority than a lower priority task, t_{low}: $P_{(t_{high})} > P_{(t_{low})}$. In our experiments we even allow for tasks to have a priority of 0, meaning we are fine with delaying such a task indefinitely.

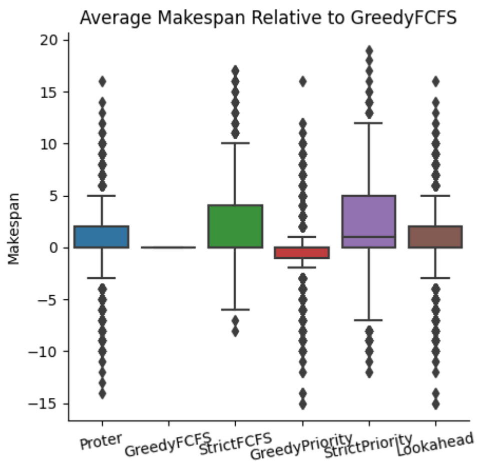
It is expected that scheduling strategies which consider priorities should have a larger makespan but a lower priority error. This is because, in general, in order to start a higher-priority task sooner the scheduler needs to deliberately delay starting a lower priority task. As a result, the total makespan tends to be higher because of the added delays, but the higher-priority tasks are less delayed than when a greedy first-come first-served method is used.

Four sets of workflows are compared: There is a small and large set, and each has a version with random priorities and a version where the priorities of all tasks are equal. 10,000 examples were generated for each set.

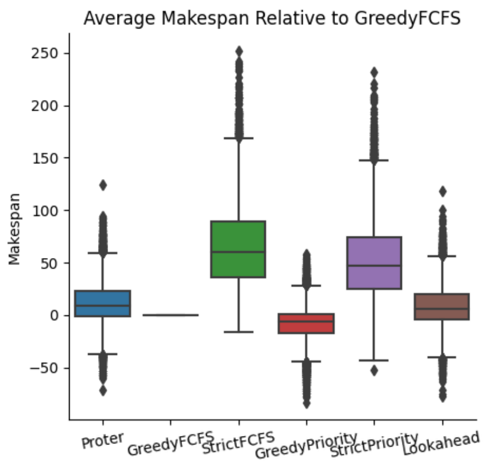
- The small sets consist of workflows which contain anywhere between 5 and 10 tasks and 3 resources. Tasks can use anywhere between 1 and 3 resources, and have a duration between 1 and 10.



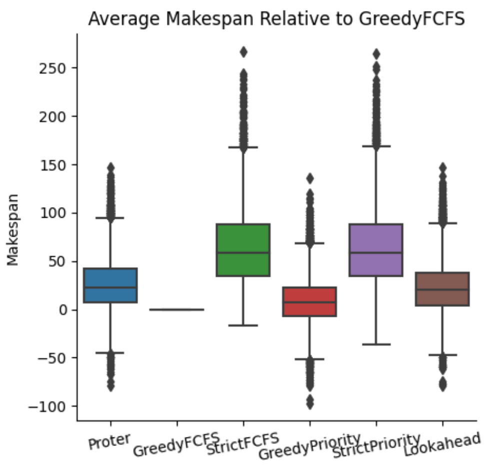
(a) Small workflows with no priorities



(b) Small workflows with priorities



(c) Large workflows with no priorities



(d) Large workflows with priorities

Figure 6.1: Makespan achieved by schedulers, relative to the GreedyFCFS scheduler

- The large sets consist of workflows with 50 to 100 tasks and 10 resources. each task has a duration up to 25, and can request any possible combination of resources.

6.1.1 Makespan comparison

Figure 6.1 shows the makespan achieved by each scheduler relative to the Greedy first-come first-served scheduler. The plots in 6.1a and 6.1b, which correspond to the two sets with small workflow sizes, had to be filtered to make the box plots visible, because by default the 25th and 75th percentile was 0. The filtered results remove the workflow data points where all the schedulers achieved the exact same makespan, the idea being that we only keep the non-trivial workflows where at least one scheduler performs differently. For the small sets, about 70% of the data points were removed with this method.

This filtering method was not necessary for the large workflows, since for both large workflow sets had sufficient complexity to where nearly every single example was gave a different result in at least one scheduler. When filtering was used, less than 1% of the points were filtered out, so these results use all 10,000 data points.

The makespans need to be analysed relative to one another. This is because some workflows are much larger than others (for example if the sum of task durations is bigger), so the only way to evaluate how "good" a makespan is would be to compare it to a baseline. In these experiments, the makespan achieved by the Greedy first-come first-served scheduler is used as the baseline, because this is the method of scheduling which is most commonly used other BPS tools. As a result, the relative makespan of GreedyFCFS is always zero, and any point below the $y = 0$ line in Figure 6.1 represents an experiment where the scheduler achieved a better (smaller) makespan than the greedy strategy.

We can see that the makespans of large workflows were about 10 times bigger than the makespans of small workflows, across all schedulers. This makes sense given that the large workflows had 10 times as many tasks.

Over all 4 sets, the GreedyFCFS scheduler achieves the shortest makespan on average, which matches our expectations. The result is most pronounced on the large, prioritised workflows in Figure 6.1d. In prioritised experiments, occasionally the prioritised methods such as the Proter and GreedyPriority schedulers achieve a shorter makespan, but this is rare and can be attributed to lucky situations where delaying a task accidentally resulted in a shorter overall duration.

It is interesting that prioritised strategies frequently outperform the GreedyFCFS strategy when we set all the priorities to be equal, essentially removing priorities from the experiments. This is especially clear in Figure 6.1a. This is because the Proter prioritisation system has to uses a priority queue to order tasks, so even if they have the same priority we use some other property of the task to determine the order. In particular, tasks which require a larger number of resources are prioritised more, so they get scheduled first. Surprisingly this method often ends up working better than GreedyFCFS of the time.

As a small detail we can check how the average Makespan changes as the workflow becomes larger. The size of a workflow is estimated by taking the sum of the durations of all the tasks in that workflow. Again we look at all the makespans relative to the GreedyFCFS baseline. Figure 6.2 shows the result on the set consisting of large, prioritised workflows (the same set used in 6.1d), but this result was consistent across all the sets: As the workflows become larger and more complicated, the difference in performance between scheduling strategies becomes larger. The greedy methods remain the best on average, while prioritised and strict methods produce progressively worse makespans.

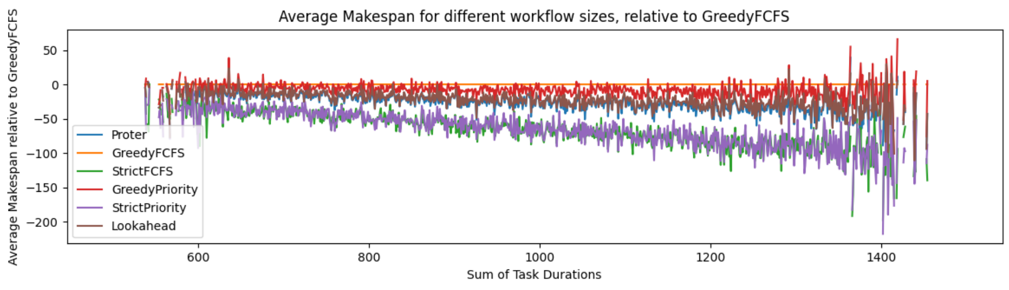
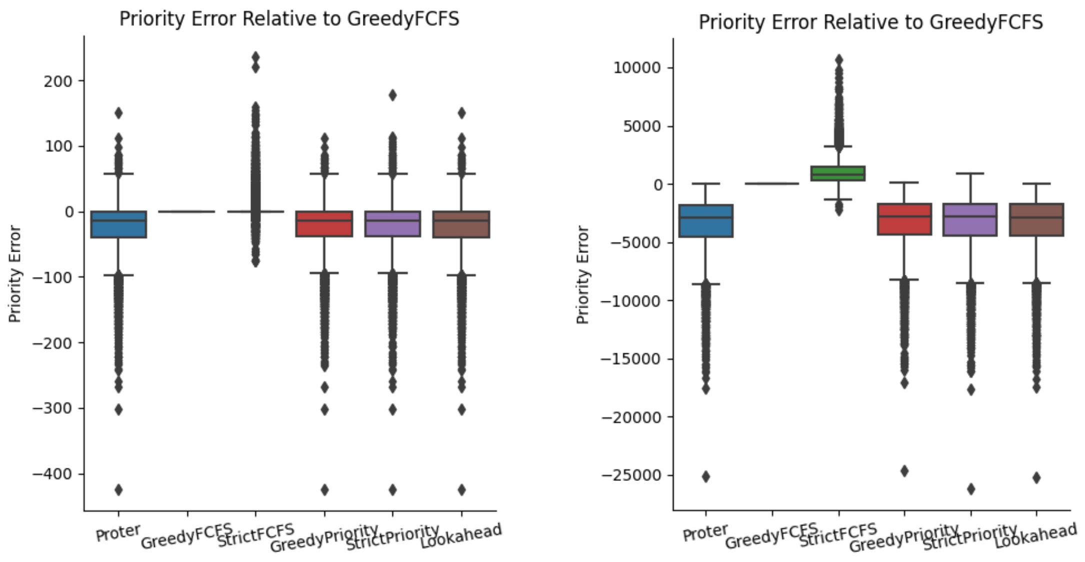


Figure 6.2: Average makespan relative to GreedyFCFS, as a function of workflow size



(a) Small workflow set

(b) Large workflow set

Figure 6.3: Priority error relative to GreedyFCFS

6.1.2 Priority Error Comparison

We compare priority error using the same method as above. Figure 6.3 summarises the results for the prioritised small and large workflows. These figures serve to demonstrate how prioritised strategies do in fact achieve a better error (lower is better), and that the spread of results is very large. Since some of the box plots are very close to one another, the mean priority error achieved by each scheduler is additionally summarised in Table 6.1, again keep in mind that these values are all relative to the GreedyFCFS scheduler, so:

$$\text{Mean Relative Error}_s = \frac{\sum_w PE_{s(w)} - PE_{\text{GreedyFCFS}(w)}}{|w|},$$

where s is the scheduler, w is the w'th workflow, $|w|$ is the total number of workflows (10,000 in these experiments), and $PE_{\text{GreedyFCFS}(w)}$ and $PE_{s(w)}$ are the priority errors as defined previously.

As shown by these results, the various priority methods perform very similarly using our priority error metric. The Proter scheduler's average narrowly beats out the basic

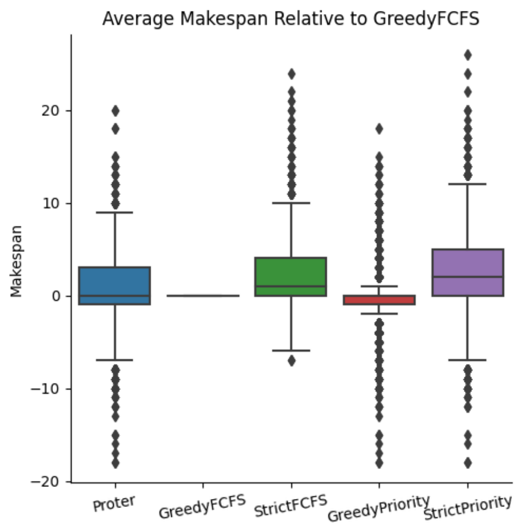
	Proter	StrictFCFS	GreedyPri.	StrictPri.	Lookahead
Small Flows	-24.6938	4.1098	-24.4321	-23.6937	-24.6254
Large Flows	-3442.298	1072.6789	-3329.1951	-3377.9733	-3412.3749

Table 6.1: Mean Relative Priority Error for small and large workflows

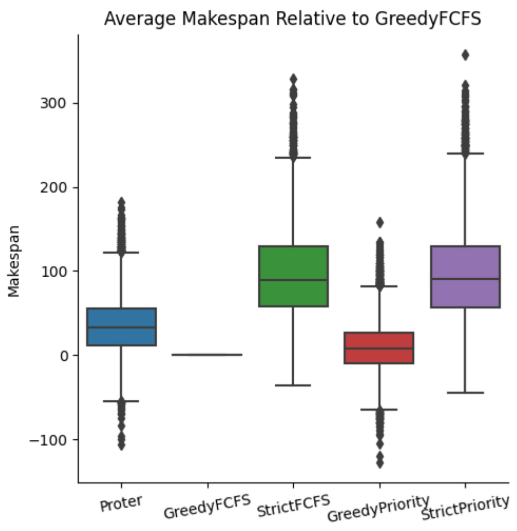
prioritised strategies such as the GreedyPriority scheduler. There exist individual examples where other strategies end up getting a better priority error than the Proter scheduler, but these are uncommon. All the methods get a better score than the basic GreedyFCFS strategy, except for StrictFCFS which gets noticeably worse results, especially on the large workflows.

The above experiments confirm our baseline beliefs, showing that prioritised schedulers such as the Proter scheduler are in fact better at respecting workflow priorities, at the cost of makespan. The experiments also demonstrate that these various basic strategies result in very different simulation behaviour, supporting our claim that choice of scheduling strategy is important, and that being able to compare strategies using this framework against an organisation's personalised metrics has value in revealing which strategy should be used. In the last section we also want to study how the new resource constraints added in this project affect these scheduling strategies.

6.2 New Constraints' Effect on Schedulers



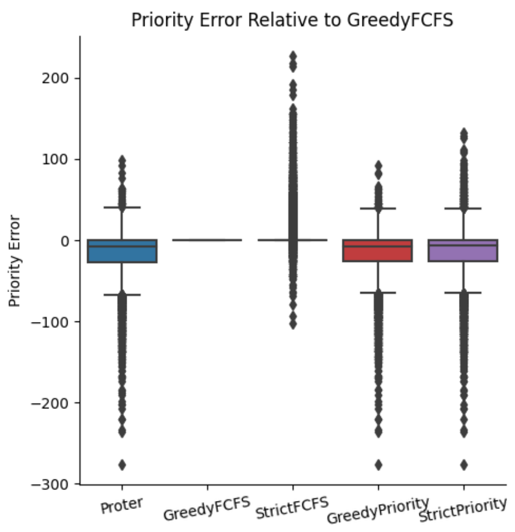
(a) Small workflow set



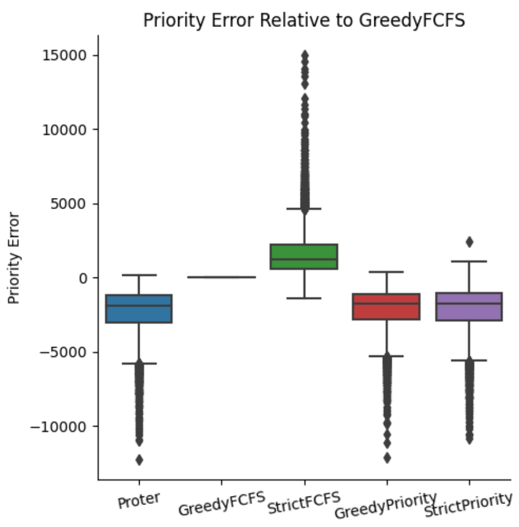
(b) Large workflow set

Figure 6.4: Makespan relative to GreedyFCFS

Building on from the previous section, we can repeat similar experiments on new workflow simulations with the new resource constraints introduced in this project. We use two new sets where resource capacities are enabled, and where tasks can request a random number of resources and also a random capacity from each of these resources, all drawn from uniform probability distributions. As before, we use a set of small



(a) Small workflow set



(b) Large workflow set

Figure 6.5: Priority error relative to GreedyFCFS

workflows which consist of 5 to 10 tasks, and large workflows consisting of 50 to 100 tasks. The small workflows can choose between 3 resources each with a max capacity of 3, and there are 10 resources used for the large workflows, each with a capacity of 30. These parameters are chosen so that there should still be some amount of conflict between task start times, resulting in delays. If we make the capacities of resources too generous, then all tasks could easily run in parallel without the constraint of a resource's limited space.

These results no longer include the Lookahead scheduler because it did not fully support capacities at the end of this project. It is quite difficult to add capacities with the way lookahead scheduling is implemented in Proter, and it would require a number of reworks and a lot of refactoring which was not possible due to time constraints. The remaining schedulers are still tested in a similar manner to the previous section.

Makespan and priority error for the small and large sets are reported in Figures 6.4 and 6.5, relative to GreedyFCFS metrics just as before. Comparing these results to the previous figures will show that the trends between the different scheduling strategies are almost exactly the same.

We expected that adding more complexity via enabling capacities would result in a larger spread of results, so the standard deviations of observation differences should be greater than before. For makespan, the standard deviation of results on the small workflow set is essentially identical, but the large dataset shows a subtle increase. The standard deviation of the relative makespan across all schedulers with large workflows was originally 40.110, and after adding capacities it is now 57.006. This is noteworthy because the average size of the workflows, approximated using the sum of all task durations, is still the same since we use the same parameter settings for number of tasks and their durations. The mean makespan is smaller after adding capacities: 883.436 compared to 775.242 originally. This makes sense because the resources have more capacity than before (originally they essentially only had 1 capacity), and tasks don't always use the entire capacity of a resource, meaning that there are typically more opportunities for parallelism which reduces makespan.

Unfortunately, the results regarding priority error contradict this. The results for small workflows are essentially identical before and after enabling capacities. For the large workflows, the original results without capacities showed a mean relative error of -1815.357, and a standard deviation of 2661.894, and now after enabling capacities we observe an increased mean of -977.326 and a standard deviation of 2046.004. So, just as with makespan, the mean is reduced as a result of more parallelism thanks to capacities, but the standard deviation of results goes up.

The results showing an increased standard deviation in makespans are evidence in favour of the initial hypothesis that adding more resource features results in more freedom of choice by the scheduler, leading to a larger spread of performance by different schedulers. The results for priority score seem to contradict this. One interpretation of this is that not all key performance indicators are impacted the same way by increased complexity. The added capacities seem to make scheduling with priorities more difficult (as evidenced by the increased mean error), but also slightly more consistent across scheduling strategies. In contrast, makespans are shorter but results are more spread out after adding capacities, showing that there is generally more ways for schedulers to run the simulation despite the shorter makespans. The fact that these results are hardly noticeable on small workflows and only manifest in larger workflows is also evidence that added complexity leads to a larger spread of results.

The core takeaway remains, that the choice between scheduling strategies - even very basic ones which only have minuscule differences - can drastically impact the results of simulations. Most BPS tools take scheduling for granted, and simply resort to only using a basic first-come first-served approach like GreedyFCFS in Proter. If researchers and BPM professionals want to model and simulate complex and realistic workflows which match real business operations, they need to be equipped with the tools to examine how different scheduling strategies impact their systems. In this project, while trying to provide users with more flexibility by implementing features such as resource capacities and multi-capacity assignment of resources, we inadvertently impact how well these schedulers perform on various metrics. We discovered here that some key performance indicators like makespan may become more dependent on choice of scheduling strategy, while other metrics such as our priority error might (surprisingly) become more consistent between schedulers. Since different metrics seem to be affected in different ways when we increase model complexity, this is even further motivation as to why researchers need to explore different scheduling strategies to see how they impact their own domain-specific key performance indicators.

Chapter 7

Conclusion

Business Process Simulation is a widespread and very powerful technique for studying business processes. In many cases, it is beneficial to simulate models which closely reflect the real world, such that the results of the simulation accurately predict how the proposed scenario will perform if it was implemented in reality. By making giving researchers and BPM professionals more tools we want to empower them to make more detailed models.

One area which we found to be lacking in terms of expressive power was the "Resource Perspective", which is concerned with how resources are modelled in BPS. We found that many BPS tools only provide basic resource features, and that even in the literature there is little consensus on what resource constraints are relevant to the field of BPM. Our simulator, Proter, already had some resource constraints which are not always found in other tools, such as Multi-Resource assignment, and it has additional unique advantages over other tools such as the prioritised ad-hoc scheduling mechanism which is at its core. We decided to implement some of the resource constraints which we identified into Proter, enabling its users to create more accurate models.

A concern with adding extra resource constraints was that there is more responsibility placed on the scheduler to make decisions about how the workflow is scheduled and executed. We believed that the choice of scheduling strategy will become more important to the outcome of the simulation as we increase model complexity, so researchers also need to be given the tools to effectively compare and evaluate workflows and schedulers in order to make an informed decision for their own systems. Traditionally a greedy scheduling approach was used by default, but there is a high chance that this is not sufficient for more complex systems. We found that using large datasets to evaluate performance is unfortunately not a viable option. As such, this project also proposes an evaluation framework based around random generation of workflows, and the relevant tools for this type of evaluation were implemented in Proter. We used these tools to evaluate the basic schedulers available in Proter, showing how much of an impact they can have on the outcome of a simulation. Using this system we demonstrate why evaluating scheduling strategies as part of BPS studies becomes increasingly important when the models become complex. By demonstrating the importance of scheduling strategy choice, and through providing the tools necessary to carry out such evaluations in Proter, we hope that this practice could become more commonplace in this field of research.

7.1 Project Outcomes and Critical Evaluation

In this project we collected and carefully described a number of resource constraints which are important to BPS, which was presented in Chapter 3. The list is inspired from features found in existing BPS tools and the literature surrounding the resource perspective of BPS. It also introduces a few new resource constraint ideas which are grounded in reality and could be useful for certain simulations. The definitions for resource constraints found in the literature are sometimes vague and contradictory, so we make sure to carefully define each point on our list, and we propose that this set of resource constraints could be used as a standard guideline for the resource perspective. We believe the list is fair and unbiased, and it was not designed to try and make Proter look more sophisticated than other tools; Even after this project Proter has still only has support for under half of the listed points.

The project managed to implement two of the identified resource constraints into Proter: Resource Capacity / Quantity, and Multi-Capacity Assignment. Most of the schedulers in Proter were also updated to work with these new capacities. This implementation involved designing weighted schedules, which were tricky to get right but ended up working very well. It is a shame that the Lookahead scheduler was not fully working with the new capacities at the end of this project due to time limitations, given more time this would be the first thing to fix. This scheduler still currently works, but it assumes that resources still only have 1 capacity.

Originally we also hoped to implement more resource constraints. This did not end up happening because we discovered that a lot more attention had to be directed towards the choice-of-sheduler problems which we ended up tackling in this project. It was not immediately clear at the start how important the scheduler comparison and the evaluation framework would be to this thesis, and the project might have progressed more smoothly if these ideas were explored more in the early stages. It would also be good to have even more resource constraints in Proter, but this is a good direction to undertake for future work, especially now that a very thorough evaluation system is supported in Proter.

The framework for evaluation which we outline in this project is also a useful outcome. As far as we are concerned, this type of approach to evaluation has not been used in any BPS studies or tool comparisons. Proter was improved with the tools to generate random workflows given a wide range of parameters specifying the configuration of tasks and resources, enabling future work with Proter to easily gather large amounts of data.

The newly implemented features in Proter were also used to show why the choice of scheduling strategy is important to simulations. We evaluated the basic schedulers in Proter to show how much they differ when it comes to some basic key performance indicators. This evaluation also demonstrates that adding additional complexity to the simulator (such as the resource capacities implemented during this project) might make the impact of scheduling strategy more important to the simulation for certain metrics. We hope this serves as motivation for future BPS studies as to why the choice of scheduling strategy deserves attention, and that by providing the tools to carry out such evaluations in Proter this practice can become commonplace.

As a critique, there is room for more exploration in our scheduler evaluation, and given more time on this Project it would have been good to see more detail in this section. We had many interesting ideas for exploring how different parameters of workflows affect the outcome, for instance exploring the effect of the branching factor (the ratio of "and" nodes to "then" nodes), or varying the ratio of tasks to resources.

Overall we think that this project was successful in improving Proter in a number of ways, and we are glad it also has some outcomes which have positive contributions to the ongoing discussion in the literature surrounding resource constraints and scheduling in addition providing tools which enable new research with Proter in the future.

7.2 Future Work

Using the list of resource constraints identified in this project gives a good potential direction for the development of Proter. It would be great to see more of these resource features implemented in future work.

Thanks to the evaluation framework which is now supported in Proter more work can be done in exploring the effect of various workflow properties. Future work could study the impact of branching and resource number on workflows of different scales. Work could also evaluate more metrics such as establishing a balanced score which accounts for multiple results including makespan, priority error, and resource utilisation.

Another compelling research direction would be to implement and evaluate more sophisticated scheduling strategies. Approaches including artificial immune systems, ant colony optimisation, and genetic algorithms, are common in related areas such as RCPSP, and it would be interesting to try and implement one or more such systems in Proter, and to compare it to the roster of existing baseline strategies.

Lastly, Proter could be applied in new real-world practical industry settings to use it for simulation and analysis of real systems. Some past work has seen WorkflowFM (the platform which includes the Proter simulator) be used in healthcare and manufacturing applications, and it would be good to see more work in these directions which can benefit from the newly added resource constraints.

Bibliography

[1] Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

[2] Rockwell Automation. Arena simulation software. `https://www.arenasimulation.com/`. Retrieved 9 April 2021.

[3] Michal Baczun. *Enhancing Simulation Capabilities in Proter*. PhD thesis, The University of Edinburgh, 2021.

[4] Bizagi. Simulation in bizagi. `https://help.bizagi.com/bpm-suite/en/index.html?simulation_in_bizagi.htm`. Retrieved 9 April 2021.

[5] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete applied mathematics*, 5(1):11–24, 1983.

[6] Bonitasoft. Bonitasoft. `https://www.bonitasoft.com/`. Retrieved 9 April 2021.

[7] Arianna Borrelli and Janina Wellmann. Computer simulations then and now: an introduction and historical reassessment, 2019.

[8] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

[9] BOC Group. Adonis process simulation. `https://knowledge.boc-group.com/en/module/adonis-process-simulation/`. Retrieved 10 April 2021.

[10] Michael Hammer. What is business process management? In *Handbook on business process management 1*, pages 3–16. Springer, 2015.

[11] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[12] Monique Jansen-Vullers and Mariska Netjes. Business process simulation–a tool survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark*, volume 38, 2006.

[13] Kurt Jensen. Coloured petri nets. In *Petri nets: central models and their properties*, pages 248–299. Springer, 1987.

[14] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.

[15] W David Kelton. *Simulation with ARENA*. McGraw-hill, 2002.

[16] Visual Paradigm International Ltd. Visual paradigm. `https://www.visual-paradigm.com/`. Retrieved 10 April 2021.

[17] OMG. Business process modeling notation (bpmn) (2011). `https://www.omg.org/spec/BPMN/2.0/PDF`. Retrieved 23 October 2020.

[18] MABEL OÜ. Bimp simulator. `https://bimp.cs.ut.ee/simulator/`. Retrieved 9 April 2021.

[19] Nathaniel Palmer. What is bpm? `https://bpm.com/what-is-bpm`. Retrieved 23 October 2020.

[20] Petros Papapanagiotou and Jacques Fleuriot. Workflowfm: A logic-based framework for formal process specification and composition. In *International Conference on Automated Deduction*, pages 357–370. Springer, 2017.

[21] José Luís Pereira and António Paulo Freitas. Simulation of bpmn process models: Current bpm tools capabilities. In *New Advances in Information Systems and Technologies*, pages 557–566. Springer, 2016.

[22] Sander PF Peters, Remco M Dijkman, and Paul WPJ Grefen. Advanced simulation of resource constructs in business process models. In *International Conference on Business Process Management*, pages 159–175. Springer, 2018.

[23] WorkflowFM Petros Papapanagiotou. Proter. `http://docs.workflowfm.com/proter/`. Retrieved 9 April 2021.

[24] PLG2. Multiperspective processes randomization and simulation for online and offline settings. `https://plg.processmining.it/`. Retrieved 7 April 2022.

[25] A Alan B Pritsker, Lawrence J Waiters, and Philip M Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management science*, 16(1):93–108, 1969.

[26] Nick Russell, Wil MP van der Aalst, Arthur HM Ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *International conference on advanced information systems engineering*, pages 216–232. Springer, 2005.

[27] Wil MP Van Der Aalst. Business process simulation survival guide. In *Handbook on Business Process Management 1*, pages 337–370. Springer, 2015.

[28] Wil MP Van Der Aalst, Arthur HM Ter Hofstede, and Mathias Weske. Business process management: A survey. In *International conference on business process management*, pages 1–12. Springer, 2003.