Inferring Robust Plans with a Rail Network Simulator

Master's Thesis

Reuben Gardos Reid

Inferring Robust Plans with a Rail Network Simulator

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Reuben Gardos Reid

Algorithmics Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Inferring Robust Plans with a Rail Network Simulator

Author: Reuben Gardos Reid

Abstract

Over 700 trains in the Netherlands are used daily for passenger transportation. Train operations involve tasks like parking, recombination, cleaning, and maintenance, which take place in shunting yards. The train unit shunting problem (TUSP) is a complex planning problem made more difficult by uncertainties such as delays. Most existing approaches overlook these disturbances and the approaches that consider them incorporate heuristics to enhance the robustness of their solutions to disturbances. This thesis proposes an alternative approach: utilizing probabilistic programming to turn an existing planning algorithm and simulator into a generative model of the TUSP. The model introduces disturbances without the need to modify the planning algorithm or simulator. Through two types of inference, we infer a distribution of robust solutions for the TUSP. Empirical results demonstrate the effectiveness of our approach for inferring robust plans in small-scale scenarios.

Thesis Committee:

Chair:	Prof. Dr. M.M. de Weerdt, Faculty EEMCS, TU Delft
Daily Supervisor:	Dr. S. Dumančić, Faculty EEMCS, TU Delft
Committee Member:	Prof. Dr. R.M.P. Goverde, Faculty CEG, TU Delft
Daily Co-Supervisor:	ir. I.K. Hanou, Faculty EEMCS, TU Delft

Preface

The completion of this thesis report marks the end eight months of work, beginning in November 2022, and marks the culmination of my time as a master's student in Delft.

The thesis may have started eight months ago, but its real origins reach back several years. Outside of computer science, I am fascinated by urbanism–especially by the ways that people move around their environments, whether by public transit or by bike. Throughout my bachelor's and master's, I have been looking for ways to combine these interests with my computer science education in a meaningful way. This thesis is another step on that journey.

This project would not have come about were it not for Sebastijan Dumančić. His seminar on probabilistic programming inspired my interest in the subject and ultimately brought about the conversation that led to my choosing this thesis. Thank you to Sebastijan, Issa Hannou, and Mathijs de Weerdt for your collective supervision throughout this project. Week in and week out, I appreciated the calm, unwavering support and guidance.

Thank you to my friends and teammates from Force Elektro. Having such a tight-knit and welcoming community for ultimate frisbee here in Delft has been a highlight of the last two years. The trainings and tournaments were a very welcome change of pace after long days of studying.

Many of those long days were spent in the CS offices of Echo or B28. Thank you to the friends who were there, day after day. Whether they were about our projects or our lives, the hours of discussions kept me going throughout the project.

Finally, thank you to my family for your emotional and financial support. To my parents especially, thank you for your trust and support, even when the journey takes me far from home.

Reuben Gardos Reid
Delft, the Netherlands
July 17, 2023

Contents

List of Figures

Chapter 1

Introduction

The Dutch Railways (Nederlandse Spoorwegen, or NS) aims to move 1.4 million people around the Netherlands daily. In their 2021 report, NS asserts that more than 700 trains are required to accomplish such a task. Throughout the day, those trains need to be cleaned and maintained. They might also be recombined to adjust their size to match the changing demand during peak hours of the day. Most of these tasks are performed at centralized locations called shunting yards. Creating a schedule for such a shunting yard is a complex and time-consuming task for human planners to carry out and a computationally intense task for computers to complete. It is, however, a task that planners must complete daily to meet the rail network's demands and maintain a quality service for passengers.

Sahinidis [2004] describes that, since exploration began on applying optimization to real-world domains like transportation, it was recognized that most problems would exhibit uncertainty. Solutions created without taking this into account are likely to fail at some point when used in a real-world context with uncertainty. Ideally, the solutions should exhibit robustness against such effects–they should remain valid even as the scenario changes.

Making the shift from the deterministic to the stochastic problem is difficult. For many problems, seeing how something like a delay affects the rest of the scenario is not trivial. In the context of operations research, these small-scale events are referred to as disturbances [Cacchiani et al., 2014]. If an algorithm is to address such events, it is often the case that the programmer has to guess how disturbances propagate through a scenario. In the case of a disturbance like a train entering a shunting yard late, how will its tardiness interact with the rest of the trains already present? What are the correct rules to introduce in order to minimize that effect?

Is it possible to avoid directly answering such questions while still considering the stochastic elements of a problem? This thesis introduces an approach to augment existing planning algorithms to extract plans that are robust to stochasticity.

1.1 Why Probabilistic Programming?

The approach introduced here is mainly built on the Probabilistic Programming (PP) paradigm. At the most basic level, it consists of programs that contain random choices. These programs can be thought of as statistical models. What benefits does that offer? Why use PP as a starting point?

There exist other attempts at constructing robust plans for the problem at hand. Existing techniques, such as those developed in van den Broek [2022], modify the planning algorithm itself in order to increase the robustness of its output. This thesis takes a different approach, attempting to increase the robustness without modifying the planning algorithm, thereby decoupling the planner from the need to consider uncertainties. A further benefit of this choice is that the process becomes agnostic to the planning algorithm used. As long as the planning algorithm is suitable for the deterministic version of the problem, it fits within the framework of this approach.

Probabilistic Programming is relevant in this case for two main reasons. The first benefit it offers is an approach to modeling the problem in a way that takes advantage of existing models. The idea is to augment a deterministic simulator with non-deterministic capabilities using a probabilistic programming language (PPL). It is a technique that has gained attention recently. Wood et al. [2022], for example, made use of an agent-level epidemiological simulator to characterize how different health policies could affect rates of COVID in a population. In this case, defining a generative model would consist of building an agent-level simulator in a PPL. This process is greatly simplified because the simulator encapsulates the complex logic behind simulating an entire population.

The problem of planning schedules for shunting yards–referred to in the literature as the train unit shunting problem (TUSP), is a similar, complex problem. Defining a purely statistical model of the problem would be difficult. PPLs afford us enough flexibility to program the entire TUSP from scratch. While this flexibility is great, the complexity of the problem means that this programming process would be time-consuming. Luckily, one of the advantages of defining a statistical model in a PPL is that it is possible to make arbitrary calls to external code within the definition of the model. In this manner, the external code can encapsulate the logic of the problem.

For the TUSP, an existing simulator, Treinonderhoud-en-rangeersimulator/Train Maintenance and Shunting Simulator (TORS), is used. TORS models the operations of a shunting yard and includes a framework for implementing and applying planning algorithms to solve instances of the TUSP. Being able to call the simulator from within the generative function means that the entire model of the TUSP can be neatly encapsulated within a single function call. In the same way that the approach is agnostic to the planner used, the exact simulator is not essential either. This freedom theoretically extends the method to other problems beyond the TUSP.

The second benefit of PP is one of Bayesian inference in general. After applying some inference methods to the model, the output is not a single solution but a distribution over solutions. This behavior differs significantly from the standard

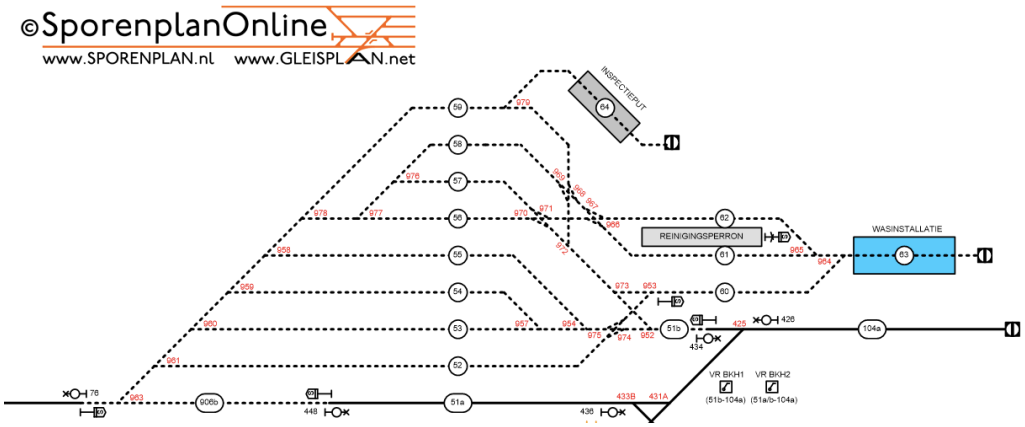


Figure 1.1: The Kleine Binckhorst shunting yard in The Hague, Netherlands. Diagram from SporenPlan.nl[1].

optimization approaches to planning problems in that they only output one best solution. For the TUSP, the distribution over solutions translates to a distribution over plans. The idea is that the density of this distribution should correspond to the robustness of the plan at that point in the distribution.

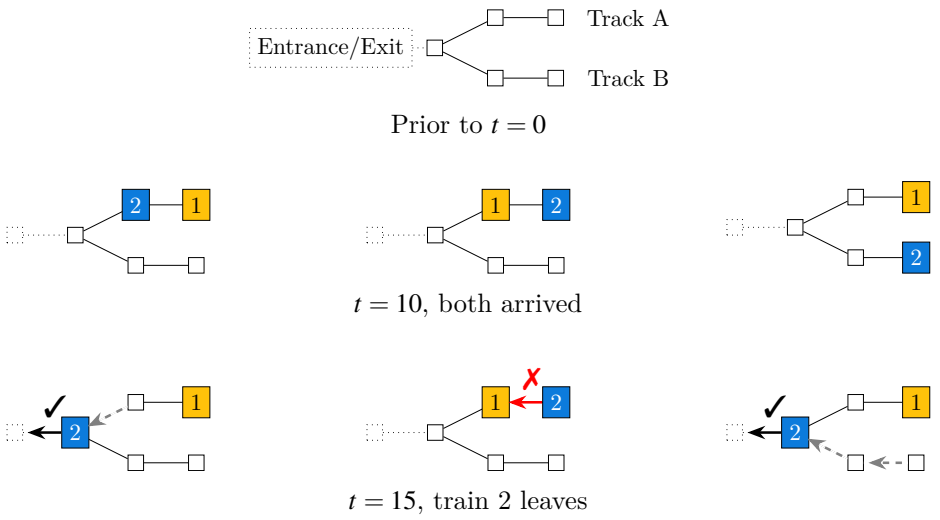
1.2 Robustness

What is robustness in this case? Given that, in real life, a human performs nearly every action represented in the problem, many variables within the TUSP could assume some uncertainty to simulate disturbances. For example, some tasks involve cleaning the inside of a train unit. The duration of this task will vary from execution to execution.

For this thesis, the scope of the TUSP is limited, eliminating service tasks and recombinations, thereby limiting the points at which uncertainties can exist to the arrival and departure times. Therefore, the simulated uncertainty will come from stochastic arrival times. When supplied to the planner and simulator, the delays will correspond with some departure times that may or may not be delayed from the scheduled departure times. A robust plan is then defined as one that results in the smallest possible outgoing delays in the event of incoming delays.

Relating this definition to the distribution of plans, the denser, more robust portions correspond to plans that maintain minimal outgoing delays under some uncertainty in arrival times.

[1] http://www.sporenplan.nl/html_nl/sporenplan/ns/ns_nummer/gvc-bkh.html



(a) Feasible plan, on-time (b) Feasible, train 1 late (c) Robust plan, train 1 late

Figure 1.2: Scenario displaying the utility of a robust plan in the case of a late train. The leftmost column shows the case where both trains are on time, but the plan is not robust. The middle column shows the same plan, but with train 1 late, blocking train 2. The rightmost column shows the robust plan, where train 1 is late, but train 2 is unaffected.

1.3 A Motivating Example

Before clearly defining the research questions, the following provides concrete motivation for the need for valid and robust plans. Previously, it was mentioned that trains must visit shunting yards to be repaired and cleaned, among other tasks. Figure 1.1 shows the layout of one such yard called "Kleine Binckhorst" in The Hague. Shunting yards are generally connected to the greater rail network at several points and expand to many tracks within the yard. For this example, let us consider a simpler example with only two track sections and one connection to the rest of the rail network.

Any shunt plan operating this theoretical yard is sensitive to internal and external delays. As discussed previously, external delays will be the focus of this work, i.e., trains arriving at the yard later than anticipated. Ideally, the plan should absorb this delay and still have all trains ready to leave the yard with as little delay as possible. Suppose a yard cannot fulfill this role. In that case, the resulting delays can have cascading effects on the rest of the schedule inside the yard and throughout the greater network–an undesirable situation for passengers and the railway operator.

Consider the following example illustrated in Figure 1.2. Two trains are scheduled to arrive at a shunting yard at timestep $t = 0$ and $t = 5$. They must leave at

$t = 20$ and $t = 15$ respectively. Assume that the original shunt plan calls for both trains to be parked on track A while they are in the yard. If the first train has a delay of more than five timesteps, the plan suddenly becomes invalid because train 2, which was supposed to arrive at 5 and leave at 15, becomes stuck behind train 1, which arrived sometime after 5 and will not leave until $t = 20$.

A robust plan is to place the trains on separate tracks so their departures do not depend on one another. In the context of this example, this thesis aims to infer a distribution over possible plans for moving the two trains in and out of the yard, where the distribution is denser at the robust plan, placing the two trains on separate tracks.

1.4 Assumptions

For readers already familiar with the TUSP, it may interesting to note that the following assumptions are made. All problem instances considered in this work consist of single train units, and those train units do not require any servicing. Furthermore, there are no personnel. These assumptions mean that the matching, servicing, and personnel scheduling problems do not apply. For those unfamiliar, chapter 2 provides the necessary background.

1.5 Research Questions

In order to explore the creation of robust plans, this thesis attempts to answer the following questions.

- RQ 1 How can a distribution of robust shunt plans be inferred given a distribution over arrival times?
  - RQ 1.1 How can the modeling and inference process capture the cycle of planning and replanning?
  - RQ 1.2 Which inference technique can infer the distribution of plans the best?
  - RQ 1.3 How much more effective are plans characterized by the distribution as robust compared to those not?

To answer the research questions, the following chapter gives a more in-depth description of the train unit shunting problem and other existing software packages built upon within the methodology. A background on Probabilistic Programming and inference is given, followed by an overview of related works on solving the train unit shunting problem or similar problems. Before detailing the method implemented in this thesis, an abstract framework for the problem, method, and evaluation is described in chapter 3. chapter 4 follows with a description of the concrete implementation of that framework before finally evaluating the proposed approach in chapter 5.

Chapter 2

Literature

As this thesis is the result of an application of probabilistic programming to the train unit shunting problem, there is some background knowledge that readers unfamiliar with either area might find helpful. Thus, an overview concerning the TUSP, the simulator and planning package used, probabilistic programming, and inference techniques are given. Next, the approach of this thesis is compared with existing work relating to probabilistic programming in the context of simulation based-models and attempts aimed at solving the TUSP.

Figure 2.1: An example train or shunting unit consisting of two VIRM train units, with 4 carriages each. Illustration from Wikimedia[1].

2.1 Background

2.1.1 The Train Unit Shunting Problem

Three relevant terms to discuss the problem are trains, train units, and carriages. Train carriages are the smallest unit and may or may not be permanently connected with other carriages to form train units. In the problem, trains might also be referred to as shunting units within the context of the shunting yard. With that terminology in hand, the problem is as follows.

The train unit shunting problem (TUSP) is a problem in which incoming trains must be received and routed through a shunting yard so that the correct trains can leave the yard in the correct order at a scheduled time. As per the literature, there are several sub-problems that more concretely define and subdivide the problem, including sub-problems that extend the original.

[1] https://commons.wikimedia.org/wiki/File:NS_VIRM_train_side_profile.svg

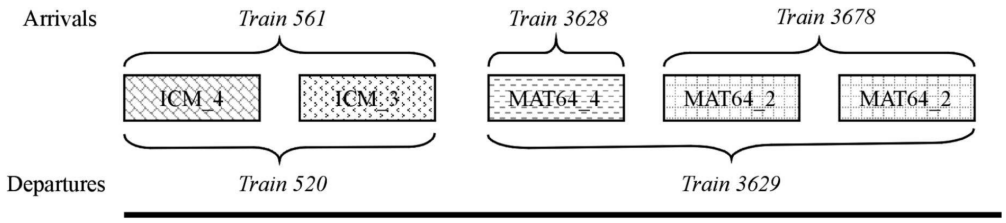


Figure 2.2: Example of matching incoming and outgoing trains from Freling et al. [2005]

The original definition of the TUSP is given by Freling et al. [2005]. It defines the matching and parking sub-problems. Matching refers to pairing incoming and outgoing train units to meet train type and schedule requirements. Figure 2.2 illustrates the matching of train units in arriving shunting units (top) with train units in departing shunting units (bottom). Based on the figure, the incoming train 3678 and train 3628 will need to be coupled at some point between the arrival of the two incoming trains and the departure of the outgoing train. The parking sub-problem requires that each train unit, or combination of train units, is assigned to a parking track that is both vacant and long enough for the train unit(s) to sit on. Lentink [2006] introduces the routing sub-problem. Solving the routing sub-problem requires finding a route through the shunting yard for a shunting unit such that it does not collide with other units.

Moving beyond general train movements, van den Broek et al. [2020] introduces the service-scheduling sub-problem. The servicing sub-problem can be viewed as a special case of the parking sub-problem, where specific shunting units must go to servicing tracks between arrival and departure to be cleaned, inspected, or repaired. Each service track also has several constraints, such as the type of tasks it can complete, the number of trains it can handle, and how long it takes to complete a task. A human must perform each train movement and service task. To consider this, van den Broek [2022] describes the addition of personnel scheduling to the problem. Such a solution must then factor in both the availability of personnel and the time they spend moving from task to task.

2.1.2 Simulator and Planning Package

In order to craft and validate solutions to the TUSP, a simulator and accompanying planning module are used. Started as a bachelor project at Utrecht University and further expanded in J. G. V. D. Linden et al. [2021], the Treinonderhoud-en-rangeer-simulator/Train Maintenance and Shunting Simulator (TORS) package provides an environment in which shunting yard layouts can be loaded and used to execute scenarios. In order to execute a scenario, the simulator provides an interface with which the planning module can interact.

First, an overview of the simulator on which the generative model is built is given.

As discussed in the previous two chapters, one of the benefits of simulation-based models is that nearly all complicated business-related logic can be encapsulated inside the simulator.

To build a generative model around the simulator, it is not essential to know the internal workings of the system, though the code is available on GitHub[2]. It is sufficient to know the inputs and outputs and whether the simulator represents the problem correctly. In other words, TORS can be treated as a black box.

As input, TORS needs four pieces of information. A configuration file lists several business rules that govern the simulation, dictating which actions are valid from a given state. A location definition defines the tracks, how they connect, and their properties, such as whether they are electrified and whether a train is allowed to park on them. An agent file instructs TORS on which planning agent to use when executing the scenario. Finally, a scenario file defines the times at which trains arrive, their properties, and the scenario's duration.

The output of TORS contains the plan created during a run. The plan is represented by a list of the actions supplied by the planning agent at each point of the simulator. Examples include Arrive, Move, and Exit actions.

The agent is an interchangeable piece of TORS, allowing for any planning algorithm to be implemented to guide the simulator. Location and scenario information is available to the planner from the beginning of the simulation, so that it can plan for the entire scenario. At each simulation step, TORS offers the agent a list of valid actions to take, and the agent must choose one to continue the simulation. In theory, any planning algorithm could be used, and a number of possible methods are discussed in subsection 2.2.1, and the exact planner used in this thesis is discussed in chapter 4.

2.1.3 Probabilistic Programming

As briefly mentioned in the introduction, Probabilistic Programming (PP) offers a way to describe random processes in a rich manner. Instead of describing them in mathematical terms, it becomes possible to describe them in the form of functions using a probabilistic programming language (PPL). These are the same as a normal function but include points in the program where the program makes random choices, meaning the function generates different outputs each time it is called–a generative function or generative model. While the output is random, it is distributed according to the model defined by the function.

The utility of the generative function comes from the implementation of the random choices within it. In a PP, the stochastic choices within the function are special sample statements. They are special in that they allow the programmer to query the model. A query is done by conditioning some random choices to specific values, often referred to as observations. With a model and observations, an inference method can characterize the distributions of the other unobserved random choices in the model.

[2]https://github.com/AlgTUDelft/cTORS

2.1.4 Inference

Given a generative model implemented in a PPL, the goal is often to provide some constraints, referred to here as "observations", to the model to investigate the resulting posterior distribution. This process is referred to as inference, and there are several approaches from which to choose. For an excellent in-depth explanation and demonstration, the reader may refer to the "Algorithms for Inference" chapter in Goodman et al. [2016].

In the most basic models that only contain discrete variables, one option is to enumerate all of the combinations of choices for values within the model and count how many times the provided observation conditions are met. While this is guaranteed to produce an exact and correct answer, it is inefficient on discrete models with many possible combinations of assignments for random variables. It is also intractable on models with continuous random variables. Rejection sampling is a simple alternative. By taking many samples of the model and only keeping those that match the observations, it is possible to get an estimate of the posterior distribution. While both options are simple to implement and useful in models with a few discrete variables, they do not perform well on large, continuous models.

Another class of inference methods falls under variational inference. These methods approximate the posterior distribution with a continuous set of simpler distributions. The distance between the approximation and the actual distribution is optimized using gradient descent to fit the approximation. This descent is possible, even without direct access to the posterior–which is a necessary property because the whole point of inference is to find some approximation of the posterior.

Monte Carlo (MC) methods form a class of inference techniques where the guiding principle is to repeatedly take random samples of a model in order to characterize the posterior distribution in question. One of the most straightforward flavors of MC methods is importance sampling (IS). Similarly to rejection sampling, IS takes several random samples. However, instead of rejecting samples that do not match the observations, it uses all samples and weights each sample based on how well it matches the observation.

If it is possible to execute the model partially–that is, run the generative function up to a certain point–then it is also possible to use particle filtering. Also referred to as sequential Monte Carlo, this technique takes several samples in parallel, each representing a particle. It pauses execution at points in the function where there are observed random values and assigns each particle a weight based on how well it matches the observations before resampling the population based on the weights–filtering out particles that do not conform well to the observations and extending the lives of those that do.

Another common MC method is Markov chain Monte Carlo (MCMC). Instead of taking many random, independent samples, the idea is to iterate on a single sample, changing a subset of the random choices within the model. By taking iterative steps, this builds a chain where each step depends only on the previous (a Markov chain). Over a large number of inference steps, the accumulation of the states of the chain

begins to resemble the posterior. There are a large number of variations on this method. The main design decisions are: how to select the subset of random choices to update and how to update those choices–also known as the proposal.

A common MCMC method is the Metropolis-Hastings (MH) algorithm [Hastings, 1970]. The algorithm works as described above, making a Markov chain. At each step, a proposal distribution is centered at the current value of each random choice. The algorithm then samples new values for the random choices from the proposal distribution to create the next step.

Two general drawbacks to remember when dealing with MCMC algorithms are that the samples are autocorrelated and initial samples might be non-representative of the real posterior. Autocorrelation means that samples close in time (inference time) are correlated. The samples are correlated by definition since they depend directly on their predecessors. The related effect to watch out for is that larger numbers of samples do not necessarily give larger amounts of information if most of the additional samples are correlated with existing samples or with each other. Initial samples being non-representative of the actual posterior means that many of the first samples might come from a low probability density region and thus should not be relied upon to characterize the posterior. This period in inference is commonly referred to as burn-in. To counteract it, one might throw out the samples from the burn-in period [Raftery and Lewis, 1996] to prevent them from influencing the resulting posterior.

In theory, all of these inference methods will deliver similar results. Some might be slow and accurate, others less accurate but quick to complete. The choice mainly comes down to the constraints made by the type of model used. Once the model is defined in chapter 4, the chosen inference methods and the motivation for choosing them are given.

2.2 Related Work

In general, probabilistic programming allows for defining probabilistic distributions that would otherwise be difficult or impossible to define. Writing a model as a program is often a much easier task. It allows the modeler to reason about the situation more clearly than formulating it purely mathematically [Goodman et al., 2016]. While using a program to define a statistical model allows for great flexibility, it can still be work-intensive, as with developing any piece of software. The process becomes incredibly time-consuming when the simulation logic is complex.

To this end, Baydin et al. [2019] introduced a framework for coupling scientific simulators to probabilistic programming languages to create models that incorporate the complex logic of the existing simulators. Baydin et al. [2020] applied this work to the particle physics domain using the SHERPA[3] simulator. Wood et al. [2022] demonstrate similar techniques in the epidemiological setting, using an agent-based

[3]Simulation of High-Energy Reactions of PArticles – https://sherpa-team.gitlab.io/

epidemiological dynamics model, FRED[4] Grefenstette et al. [2013], to create a generative model of the outcomes of public health policy decisions.

Traditionally, statistical methods like Bayesian inference require that a likelihood can be calculated for the model in question. Calculating the likelihood of generative models defined on top of simulators is difficult. In the case of a black-box simulator, the likelihood is completely unknown. In this case, it is necessary to use so-called "likelihood-free" inference methods, which often use a sampling procedure.

In their section on simulation-based inference, Lavin et al. [2022] highlight Approximate Bayesian Computation as a pioneering method for handling likelihood-free models. Gutmann and Corander [2016] give an overview of methods to perform inference on likelihood-free models and suggest a framework for reducing the number of samples needed. Reducing the number of samples, or the number of times the simulator must be run is significant in cases where a single simulation run is costly. Meeds and Welling [2015] also propose an inference approach for likelihood-free models that can be run in parallel, reducing the impact of long-running simulations.

2.2.1 TUSP Solution Methods

Until this point, most approaches to solving the TUSP have used a deterministic formulation of the problem. The first approach comes from the work by Freling et al. [2005], which introduced the TUSP. They split their approach into two steps: one to match the incoming trains to outgoing demand and the other to assign incoming trains to parking tracks. For the first step, they create a model and apply standard MIP (Mixed Integer Programming) solving techniques. The second step is formulated as a set partitioning problem, which they then solve using a column generation algorithm to assign trains to their parking tracks. With their approach, they can solve 24-hour scenarios involving hundreds of trains that pass through without parking and roughly 80 trains that must be parked. Their runtime for solving these scenarios is one hour.

Athmer [2021] approaches the problem from the evolutionary algorithms (EA) perspective. The TUSP formulation used includes the matching, servicing, parking, and routing problems. The report mentions that the diverse set of plans generated by the algorithm would be useful to human planners, giving them similar alternatives if the initially selected plan fails due to disruptions or disturbances in the schedule. However, this is only mentioned in passing and is not focused on in the methodology nor evaluated in the results. The method can solve a larger number of trains. Scenarios with 20 to 30 trains were possible within under 10 minutes.

van Cuilenborg [2020] takes a multi-agent pathfinding approach, where an agent represents each train. Their approach can solve scenarios with between three and ten trains within a five-minute timeout. With a 30-minute timeout, all of the variants of their approach solved all 4-train scenarios, with most of the variants solving nearly all of the seven and eight-train scenarios as well.

[4] A Framework for Reconstructing Epidemiological Dynamics – https://fred.publichealth.pitt.edu/

At the time of writing, there are few attempts at creating robust plans for the TUSP. Trepat Borecka et al. [2021] takes a multi-agent perspective as well, but with the addition of deep reinforcement learning. In their work, they use a heuristic for robustness against disturbances in the duration of service tasks. They use the time each train dwells on a service track as a measure of robustness. The longer a train is on a service track, the more leeway there is in the schedule if the task takes longer than scheduled or cannot be started on time. Their work includes a wide variety of results, but generally, their approach can solve around half the tested scenarios with six or seven trains. The runtime is quite quick as the agent acts on a predetermined policy, but that ignores the training time, which was more than a day in some cases.

In another approach to deal with disturbances, van den Broek [2022] extends his previous local search, simulated-annealing approach from 2016. To guide the search, van den Broek uses several surrogate robustness measures. The surrogate measures include measures of the slack in the schedule–where the slack is how long a particular action can be delayed in a schedule before it disrupts the subsequent action. Other measures quantify the number of actions that depend on previous actions and the lengths of those dependence chains. By feeding such measures to the search, the algorithm can provide plans that remain feasible in the event of disturbances, such as uncertain arrival times and varying duration in service tasks.

In comparison, this thesis introduces uncertainty to the arrival times and attempts to find solutions that continue to function in the presence of said uncertainty. However, that divergence in formulation does not mean that the deterministic approaches are not useful for the stochastic equivalent. Given that the approach of this thesis builds on top of a deterministic simulator and planner, the approaches mentioned above are all theoretically viable options to plug into the stochastic approach as a planner. The trait that could render some deterministic methods impractical in this work is their time to solution. The planner runs a large number of times during the search for a robust solution. If the runtime of the planner is not small enough, then the time to a robust solution becomes too long. The other downside of all approaches listed is that the implementations are not available; thus, from a practical standpoint, simply "plugging them in" requires a large amount of time for implementing them in a way compatible with TORS.

2.3 Conclusion

In this chapter, we have provided background on the TUSP, the TORS package, Probabilistic Programming, and an assortment of inference methods. In the related work, we touched on a number of works utilizing simulators within models to perform simulation-based inference. These approaches are similar to ours but applied to other contexts, such as physics and public health policy. Lastly, we gave an overview of approaches for solving the TUSP, all of which are relevant for our method because it uses a planning algorithm internally–the choice of which is covered in chapter 4.

Chapter 3

Problem Setting

Thus far, the various components of the methodology have been introduced: a simulator, a planner, probabilistic programming, and inference. However, the exact ways the components build upon one another still need clarification. To this end, this chapter describes the framework this thesis proposes at an abstract level. To what class of problems can this framework be applied? What must a method provide to solve the problem? How is the performance of a method evaluated?

3.1 Problem

The problem class that this framework applies to is quite large. Any planning problem should suffice as long as access to a simulator and a planning algorithm exists. The problem should have some deterministic variables in the simulator and planner but are stochastic in the real-world version of the problem.

For the variables within the problem where stochasticity should be introduced, it is vital to have control of their values via input to the simulator. It is irrelevant whether this is by controlling how they are sampled or directly supplying values to use, only that it is possible to control them externally. Regardless of the method chosen, it relies upon the ability to control such variables. It is also necessary to be able to relax rules and constraints in the simulator–by completely turning off a rule, for example. This is because the method will rely on the fact that solutions are not just valid or invalid but measurably better or worse than others continuously.

The requirements for the planner are flexible. It can be an offline planner that optimizes a complete solution all at once, or it could be online: acting on a policy while executing a scenario using the simulator. A correct or performant planner is not a constraint, but its runtime will affect on the convergence of the method. The possible implications are discussed in section 3.2.

3.2 Method

As a reminder, the method must use the simulator and planner to characterize some distribution over possible plans such that the plans in the densest points of the distribution are more robust to the uncertainty of certain chosen variables within the problem.

For a method to solve the problem at hand, it must be able to explore the space of uncertainty on the variables where stochasticity is introduced. This exploration would likely be done using samples from a probability distribution. The chosen distribution should ideally resemble the distribution of values for the variable in the real world. If the actual distribution is unknown, a more "uninformed" distribution, like a normal or uniform distribution, is also a possible starting point. The method must then be able to supply the values to the simulator/planner and run a scenario.

The chosen method must also be able to read the outcome of the simulation beyond a simple failure or success. The method must be able to assign a robustness score to the simulation result. This is the minimal requirement for creating a distribution over plans. The score assigned to the result of an execution of the planner and simulator is the basis for the density of the distribution.

It was previously mentioned that the correctness of the planner is not a hard constraint, and this ability of the method to score the result of the simulation is the reason. If the planner does not produce a viable plan, the method should score the execution of that plan poorly, giving it little or no weight in the distribution. In this way, the method can handle even a poorly performing planner. Though such a situation might not produce many robust plans–or any at all, the distribution will reflect that, having a low density for all of the resulting non-robust plans.

3.3 Evaluation

There are several possible metrics on which the method might be evaluated, and certain problems will have unique metrics. However, the following are more generally applicable to all problems and methods.

Examining the resulting distribution of robustness scores for a given problem instance gives some intuition as to the range of plans explored by that method. A method should explore many plans with a high score and avoid spending too much time exploring plans that are not robust–or even worse, exploring plans that are only valid after relaxing the constraints of the problem. For the evaluation, that means that the distribution to look for is skewed with the tail towards less robust plans.

As the method relies on taking many samples over time, it is also interesting to examine the scores of the plans it explores and how they trend over steps of inference. A performant method will likely converge towards more robust plans as inference proceeds. Methods that reach this point quickly are desirable because they will not spend time exploring less-robust plans.

Plan length is another metric of interest, as it is generally preferable to carry out less work–fewer steps in a plan–if it does not change the outcome. For this reason, examining the lengths of plans over inference steps might be informative. As the distribution of plans is created, it is interesting to see if there are any trends in the length of the plans. It is difficult to say whether more robust plans have a longer or shorter plan length, which might vary from problem to problem. For example, seeing the plan length trend upward as robustness increases is not necessarily bad. It might reflect a trait of the problem where longer plans are necessary to achieve robustness.

Lastly, it is interesting to see how the robust plans actually perform under the chosen uncertainty. There are two possible options from which to choose. The rules relaxed in the simulator should be returned to their original settings for both options. The first option is to take the best plan from the distribution for a single scenario and execute the scenario many times, resampling the uncertain variables each time and recording the percentage of times the plan fails. The other option would be to repeat the same but with the top n plans simultaneously. This alternative might shed some light on the utility of having a distribution of plans to choose from rather than one single plan.

3.4 Conclusion

Having this abstract definition of the framework aids in understanding our method and the ease of implementing future variations on our method, and we now have a framework for this thesis that we can implement in chapter 4.

Chapter 4

Methodology

This chapter details the components created to implement and test the inference of robust plans. First, a concrete definition of the generative model is given. This model represents the stochastic TUSP and is the model on which inference is performed. Given the model, the two chosen inference methods are motivated, and their application to the generative model is described.

4.1 TORS and Planner Set-up

In subsection 2.1.2, we introduced TORS. It will act as the simulator at the center of our method. It lets us directly include the TUSP in our model. In section 3.1, we described the necessary abilities for the simulator and planner. TORS provides both the ability to control the uncertain variables and turn off related business rules that govern the simulation.

In our case, it is necessary to be able to control the arrival times. We do so via the scenario definition file that dictates each train's arrival and departure times. The relevant business rule we must disable is the depart-on-time rule. Deactivating it allows for late departures, meaning our method can explore some plans that are not good but still get all of the trains out of the shunting yard.

We summarized many planners in chapter 2. Unfortunately, none of them are implemented in TORS. We attempted to implement the simulated annealing approach by van den Broek [2016] but could not complete our implementation due to time constraints.

Instead, we modified and used an example planner included in the TORS package– a greedy planner. The idea is relatively simple. At each point where the simulator offers the planner a set of actions to choose from, the planner assigns a priority to each action based on each active train. It then chooses the action with the highest priority. We made one modification to the planner to hopefully allow it to perform better over the course of many iterations. To do so, we added an ε factor such that it would take a random action with probability ε. We fixed this value to 0.2 as

```
1  @gen function solve_shunt_plan(scenario, location, agent)
2      sampled_scenario = @trace(sample_arrivals(scenario), :arrivals)
3
4      # TORS (Simulator/Planner)
5      plan = run_shunt_plan(sampled_scenario, location, agent)
6
7      @trace(calculate_delays(plan, scenario), :delays)
8
9      return plan
10 end
```

Listing 1: Generative function written using Gen.jl for the TUSP, wrapping the TORS simulator and planning package.

the planner began to perform worse with higher values. The pseudo-code for the algorithm is given in Appendix A.

4.2 Generative Model

In order to perform inference, a generative model around the simulator must first be defined. The model is defined using the generative function API from Gen[1] introduced and developed by Cusumano-Towner et al. [2019]. As discussed in subsection 2.1.3, a generative model is defined here to be a function that takes some input, performs some stochastic computation, records the stochastic choices, and returns both the return value of the function itself as well as the stochastic choices.

In the case of the TUSP, the generative function is defined as follows:

The @trace call assigns a sampled value to an address. When a function is called inside a @trace, all of the addresses assigned at @trace calls inside of the inner function are nested under the address of the outer @trace. For example, in line 2, all arrival times that are sampled within the inner function (for example a-1, a-2, ..., a-3 are nested under the :arrivals address.

The sampling process for arrivals is simple. For each arriving shunting unit, a sample is taken from a normal distribution centered at its scheduled arrival time. That sampled time is then taken as the new scheduled arrival time for that simulation run–simulating a delay.

The reason for using a normal distribution here is that there is little precedent set in the literature nor delay data to reference. van den Broek [2022] assumes a uniform distribution. However, this seems to be too simple an assumption. We assume here that trains are more likely to arrive at their scheduled time than early or late, leading us to choose a normal distribution over a uniform.

[1] https://www.gen.dev/

For delays, the process is nearly as simple. Each exiting shunting unit in the plan returned from TORS is matched with its scheduled departure. The model takes the difference between the two times as the delay of that shunting unit. The address for that delay is then assigned by sampling from a normal distribution with a small variance centered around the delay time. The reason for using a normal distribution here is more technical than anything. Of course, an exact delay value is calculated, but each random choice must be just that: random. It cannot be assigned directly. By centering a normal distribution on the delay value with a variance of 0.1, we essentially assign the delay to an exact value while meeting the requirement that all random choices must take a randomly sampled value. If a unit scheduled to depart does not do so before the end of the scenario, then the corresponding delay is infinite.

Thus we have a generative model of the TUSP with stochastic arrival times and observable exit times. The function produces a trace consisting of a plan (the function's return value), a series of sampled arrival times, and a series of sampled delays, which can be used to infer a distribution over robust plans.

4.3 Inference

Having defined the simulator, problem instances, and generative model, we can now describe the inference methods we will use. Both methods are implemented using Gen's standard inference library.

In general, the goal of performing inference is to characterize a posterior distribution. In this context, this is a discrete distribution over the plan space of the TUSP, given a location and scenario, conditioned on observing no delay in the executed plan. Given that enumerating the entire planning space is intractable, Monte-Carlo sampling methods are necessary. They will give an estimate of the distribution in question.

4.3.1 Importance Sampling

Importance sampling (IS) will act as a baseline inference method against which further methods can be compared. Out of the Monte-Carlo methods family, IS is the most straightforward. It is also supported out-of-the-box by Gen[2], and we make use of that provided functionality.

For the generative model of the TUSP, IS runs the generative model n times to create n traces or samples of the scenario. As defined in section 4.2, each run of the generative model resamples the arrival times from distributions centered around the arrival times of the starting scenario, runs the TORS using those arrival times, and records the outgoing delays that result from the simulator run. For each trace, Gen assigns a weight based on observing 0 outgoing delay from the executed plan. Traces with a delay for each train closer to 0 have a higher weight, and those with more

[2]https://www.gen.dev/docs/stable/ref/importance/

delay have a lower weight. After all of the samples are gathered, their weights are normalized. This entire process yields traces containing shunt plans with weights corresponding to the plan's ability to maintain 0 outgoing delays in the presence of stochastic arrival times relative to the other sampled plans.

4.3.2 Markov Chain Monte Carlo

While IS is straightforward, it does not take advantage of the structure of the problem in any way and relies purely on sampling enough plans from scratch in hopes of covering the distribution well enough. In comparison, Markov chain Monte Carlo (MCMC) methods iterate on a solution over time to characterize a distribution over plans. In each iteration, arrival times in the generative model are resampled from a proposal distribution, updating the scenario. Then the previous shunt plan is reused for as many steps/actions as possible. When the previous plan is no longer valid, we let the planning agent proceed with the remainder of the scenario, yielding an updated plan with updated outgoing delays. The previous and updated plans are compared based on their adherence to the 0 observed outgoing delay, and the better plan can be accepted with some probability. The details of the resampling and acceptance processes can change, but this is the general MCMC framework.

To find robust solutions to the TUSP, it makes sense to apply an iterative method like MCMC because we can let reality dictate the iteration process. The intuition is that resampling a scenario represents the real-time, continuous update of a situation where some units will arrive in the yard at the expected time and some will deviate. Each iteration mimics what might happen in the real world. The previous iteration's shunt plan represents the plan created before realizing any possible arrival delays. That existing plan is executed as long as possible until some disturbance occurs, like an early/late train unit arrival. At that moment, the human operator, represented by the planning agent, must take over and solve the scenario from the current time onward. This motivation addresses the first research subquestion: How can our modeling and inference process reflect the real-world cycle of planning and replanning?

For our implementation, we make use of the inference library implemented by Gen[3]. Gen provides functions that can be used for crafting MCMC inference methods. The mh function is utilized to construct the iterative process defined above using the Metropolis-Hastings algorithm for MCMC. For the proposal distribution, we use a normal distribution with a variance smaller than the variance of initial distribution over each arrival time. After the first trace is initialized, the subsequent arrival times vary from their previous time according to this normal proposal distribution, shifting slightly earlier or later at each inference step.

[3]https://www.gen.dev/docs/stable/ref/mcmc/

4.4 Visualizing the Distribution

With a model and inference methods in place, we already have enough to begin creating distributions over plans. However, a post-processing step is helpful to visualize and inspect the constructed plans. One helpful property of repeatedly sampling plans from the same scenario is that the plans naturally start with the same few actions. Whether the first train is late or early, its arrival will likely be the first action taken. This property makes the trie data structure is a fitting representation of the plans.

Also known as a prefix tree, the trie represents a set of words with common prefixes as a tree of letters, with the root being the first common letter of all of the words in question [Connelly and Morris, 1995]. If there are multiple first letters, as is likely the case with a random group of words, then the empty string is the root node, with each of the unique first letters as children.

We can map our distribution of plans to the trie data structure by taking each action as a letter. The Tries.jl package[4] makes this trivial by allowing arbitrary tuples in place of letters as keys to the trie. The weights of each of the branches of the trie are set to the weight of the plan in the distribution. In this way, it is possible to visualize the distribution of plans. Unfortunately, even for small problem instances, it is difficult to fit an entire visualization into a single figure, so this method is most well-suited for interactive plotting backends[5].

4.5 Conclusion

In the background in chapter 2, we discussed the TUSP and the TORS package, fulfilling the problem section laid out in chapter 3. This chapter described how we use the TORS simulator in our method to satisfy the requirements outlined in chapter 3. We then gave our method, describing the generative model and how we apply inference techniques. The first research sub-question has also been addressed: we can capture the real-world planning and replanning process in the inference method. What now remains to be seen is how the method performs.

[4] https://github.com/gkappler/Tries.jl

[5] The PlotlyJS backend of the Plots.jl package (https://docs.juliaplots.org/) provides one such interactive backend

Chapter 5

Results

5.1 Experimental Setup

Experiments are conducted on the DelftBlue Supercomputer provided by Delft High Performance Computing Centre [DHPC]. Each experiment runs on a single core on a "Standard" node whose characteristics can be found on the supercomputer's specification page[1]. Unless otherwise specified, each scenario is run with a time-out of 2 hours.

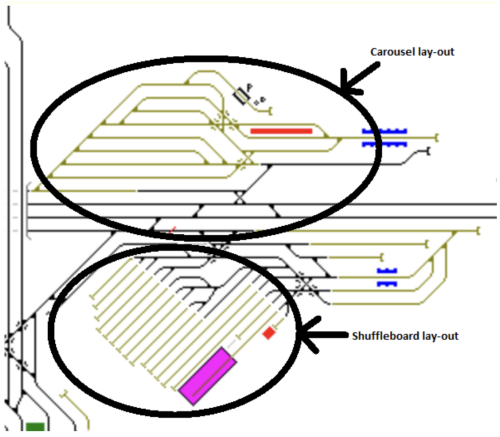


Figure 5.1: Examples of two common shunting yard layouts from Huizingh [2018].

5.1.1 Problem Instances

In order to evaluate the method, several locations and scenarios are needed. Huizingh [2018] describes the two most common types of shunt yard layouts: carousel and shuffleboard. For this evaluation, a graph representing a shuffleboard layout is constructed, arrival and departure sequences are generated, and a final conversion

[1] https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1

is made to add details necessary for compatibility with TORS. Such details include switches, track lengths, train characteristics, and scheduled arrival and departure times for each incoming and outgoing train.

The difficulty of the instance can vary in several ways, including how shuffled the arrival sequence of the trains are when compared with the departure sequence. If train A arrives before B and train A must leave before B, then the scenario becomes more difficult to solve. Assuming we are using an intelligent planner, it must be careful not to block A from leaving on time. If we use a more random planner, the instance becomes more challenging to solve because the planner is less likely to randomly come across the specific sequence of actions needed to arrange the trains in a way that allows all of them to leave on time.

With such complexities in mind, a number of scenarios are generated, varying the size and the amount of shuffling between arrival and departure sequences. The settings are summarized in Table 5.1. To keep the changes to the sequences consistent, they were not shuffled randomly but modified by reversing the order of a subsequence. The larger the subsequence, the more complex the problem. The exact function is in the appendix in section A.2.

As a reminder of the assumptions made in section 1.4, the instances have trains of only one train unit, none have service tasks, and no personnel to consider for any of the problem instances.

5.1.2 Inference Settings

In addition to varying the size and sequence of scenarios, two settings relating to inference may be varied.

The first setting is the variance of the distribution of each arrival time–the priors of the generative model. While this is a setting for the model and inference, it is tightly coupled to the difficulty of a scenario. In theory, a larger variance in the arrival time should make a scenario more complicated because the plan has to deal

N Trains	Shuffle	Init. Var.	Prop. Var.	Inference	Iter.	Time Gap
2	N trains	100	0.5	IS	5000	400
3	N trains − 1	50	0.1	MH		
4	N trains − 2	10				
5	...					
10						
15						
20						

Table 5.1: List of values for each variable during experiments. The time gap is the time between each arriving train and each departing train. All combinations of values are used. Example: 3 trains, shuffle of 2, initial variance of 100 and proposal variance of $0.5 \times 100 = 50$ is be an example of a single run.

with a wider range of interactions between trains in the yard, possibly making a robust plan more challenging to construct.

The second setting only applies to MH: the variance of the proposal distribution. Similarly to the variance of the initial distributions of arrival times, this setting is coupled with the scenario's difficulty. However, it should also affect on the ability of MH to characterize the distribution over plans. Thus, a several are tried, each set as some percentage of the initial variance.

Table 5.1 lists each variable that changes throughout experiments and the values used. An exhaustive list of combinations generates a list of experiments to run.

5.2 Revisiting the Motivating Example

In section 1.3, an example was given involving two trains. In that section, the robust plan was discerned to be a plan that placed the two trains involved on separate parking tracks upon their arrival; that way, one of them being late would not block the other from leaving on time. This example helps test whether the method works because there are only a few possible plans, and it is easy to reason about the robustness of each of them, meaning we can verify the robustness of the outcome manually. In the experiments, the problem instances involving two trains differ slightly from the example given in the introduction: there are three parking tracks, not two. A diagram of the layout is given in Appendix B. However, the reasoning that the two trains should end up on different tracks still stands. What remains to be seen is whether the method produces a distribution that reflects that.

In Figure 5.2, the distribution of log-likelihoods for both inference methods is displayed. The log-likelihood here represents the likelihood that a sample comes from the posterior: the distribution of plans with 0 outgoing delays and varying incoming delays. Note that it is the log likelihood. That means that less negative values closer to 0 represent a higher likelihood–closer to a probability of 1 in non-log space.

In the IS (top) plot of Figure 5.2, the violin plots are added to further accentuate the difference in the distributions of likelihoods. One interesting thing to note regarding the distributions is that the Shuffle: 1 scenarios are all more distributed than the Shuffle: 0 scenarios. This difference is because Shuffle: 1 means the two trains are swapped when comparing the incoming and outgoing sequence. The swap makes the repeated random sampling of plans by IS spend more inference time on plans where one train blocks the other. In the Shuffle: 0 scenario, the incoming and outgoing sequence is not swapped, meaning the last train to enter the yard is also the first one to leave, making plans that allow both trains to leave on time more likely in general.

The bottom plot for MH omits the violin plots because the values are so tightly packed near 0. A small window is shown on one of the scenarios to highlight the fact. The pattern shows that, while MH spends a small number of steps at very unlikely samples, it quickly converges to the higher likelihood space by retaining the

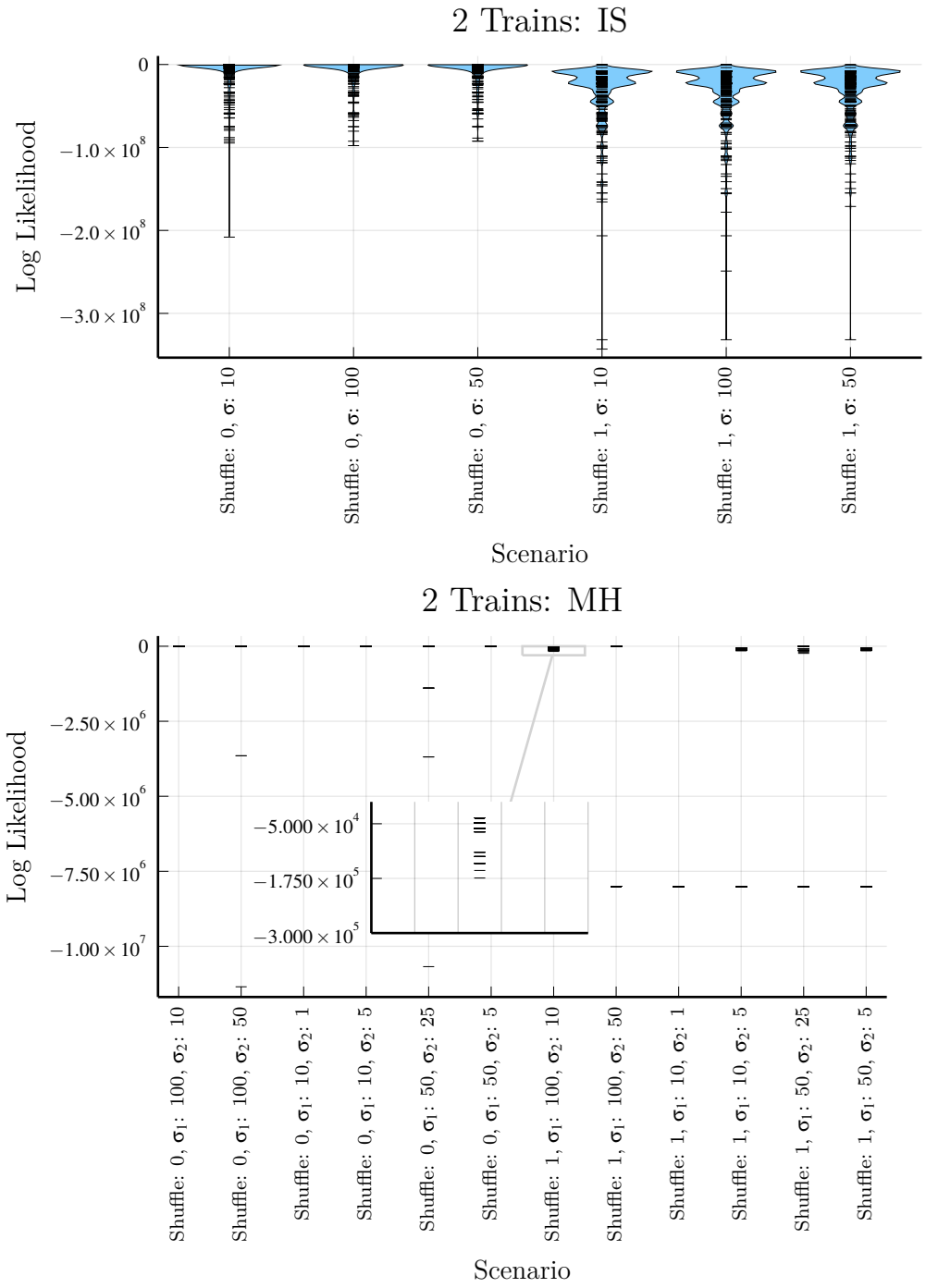


Figure 5.2: Log likelihood for 2 train scenarios with different levels of arrival and departure sequence shuffle as well as different levels of arrival time variance–denoted by σ_1 in the labels. σ_2 is the proposal variance for MH. "Shuffle: 0" are not shuffled and thus less difficult, the higher the shuffle value, the more difficult the scenario.

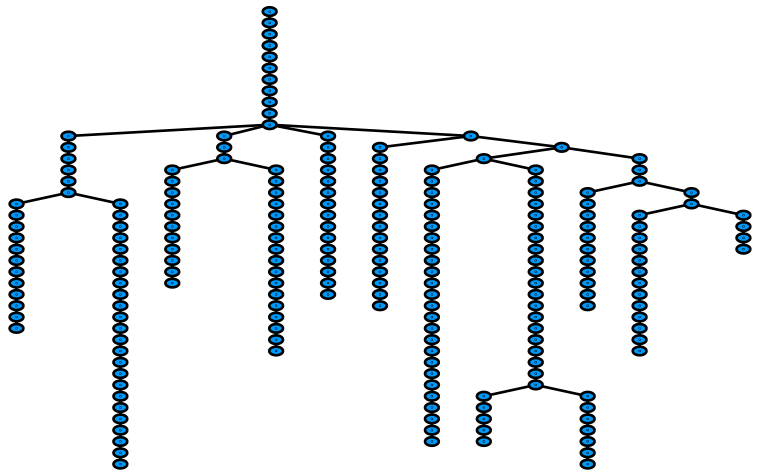


Figure 5.3: Overview of a trie representing the plans inferred using MH for the 2-train instance Shuffle: 1, σ_1: 100, σ_2: 100. The root node is the arrival action for the first train in the arrival sequence. Paths down the trie represent alternative plans in the distribution.

plan from the preceding, likely sample. The convergent behavior is the reason for using a MCMC method for this problem, so this pattern is a great outcome.

Note that one of the scenarios in the lower plot (Shuffle: 1, σ_1: 10, σ_2: 1) failed to reach the higher likelihood region that the other scenarios reached. The reason that it failed to do so is likely because of the low variance of the proposal distribution. If the proposal variance is very small and the initial trace is in a low likelihood area of the planning space, then the random walk that MH must follow will take a large number of samples. In this case, the number of samples was not high enough. In MCMC-related terms, we might say that the burn-in period is long when setting the variance of the proposal distribution too low.

The next question is whether the plans produced are recognizable as the robust plan discussed in section 1.3. There are many scenarios to choose from, and inspecting and discussing plans produced by each would take far too long. We will look at Shuffle: 1, σ_1: 100, σ_2: 50 using MH. We chose this one because it reached the high likelihood area and is slightly more difficult with the order of the two trains swapped.

To visualize the distribution, we can use the trie method described in section 4.4. Even with a 2-train scenario, this is a fairly large tree, so it would be ideal to view it interactively. Nevertheless, the overview is shown in Figure 5.3. We examine the

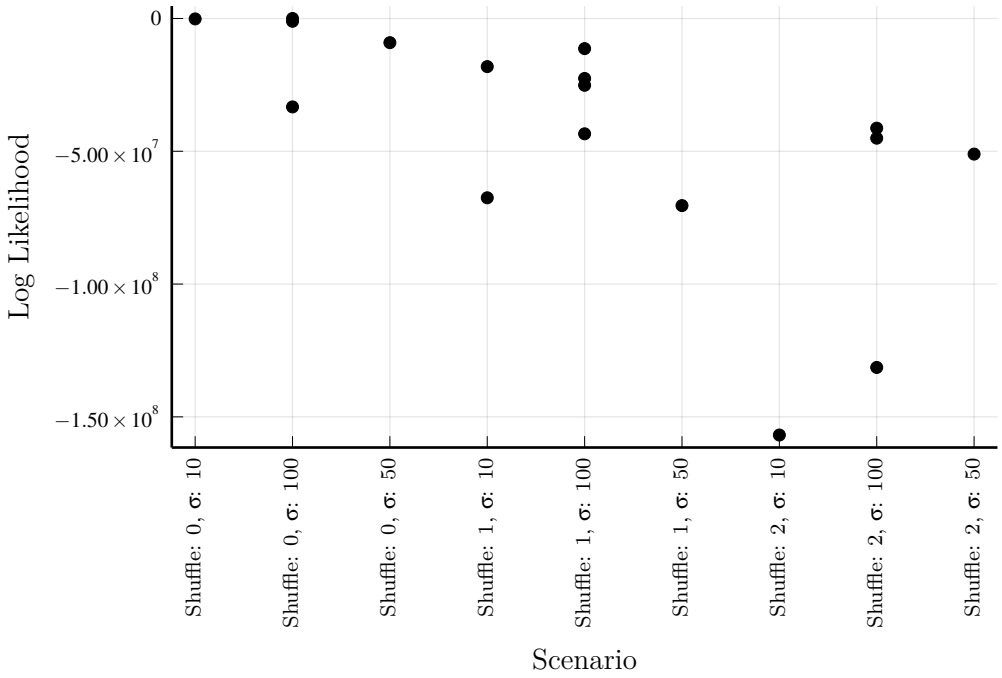


Figure 5.4: Log likelihood using importance sampling for 3 train scenarios with different levels of arrival and departure sequence shuffle as well as different levels of arrival time variance–denoted by σ in the labels. Shuffle: 0 scenarios are not shuffled and thus less difficult, Shuffle: 2 is the most difficult with the second two trains in reverse order.

most robust plan with the highest likelihood to gauge whether the distribution acts as expected. In the chosen scenario, the highest log-likelihood was -8.28. The list of actions from this selected plan is included in Appendix B. The plan does not exactly match the "ideal" robust plan introduced in chapter 1. It includes some unnecessary extra moves–a side-effect of using a planner with some randomness involved. However, it does reflect the idea that placing trains on separate branches is ideal for avoiding conflicts when disturbances occur. In this way, it resembles the robust plan discussed in chapter 1.

5.3 3-Train Instances

The log-likelihood distributions of the 3-train scenarios already change significantly compared to those of the 2-train scenarios. In Figure 5.4, depicting the distribution produced by importance sampling, the number of data points is far lower than the number we saw in the two train scenarios. Given that the complexity of the planning space increases each time the number of trains increases, it is not surprising that the number of successful runs decreases. One extra factor playing a prominent role

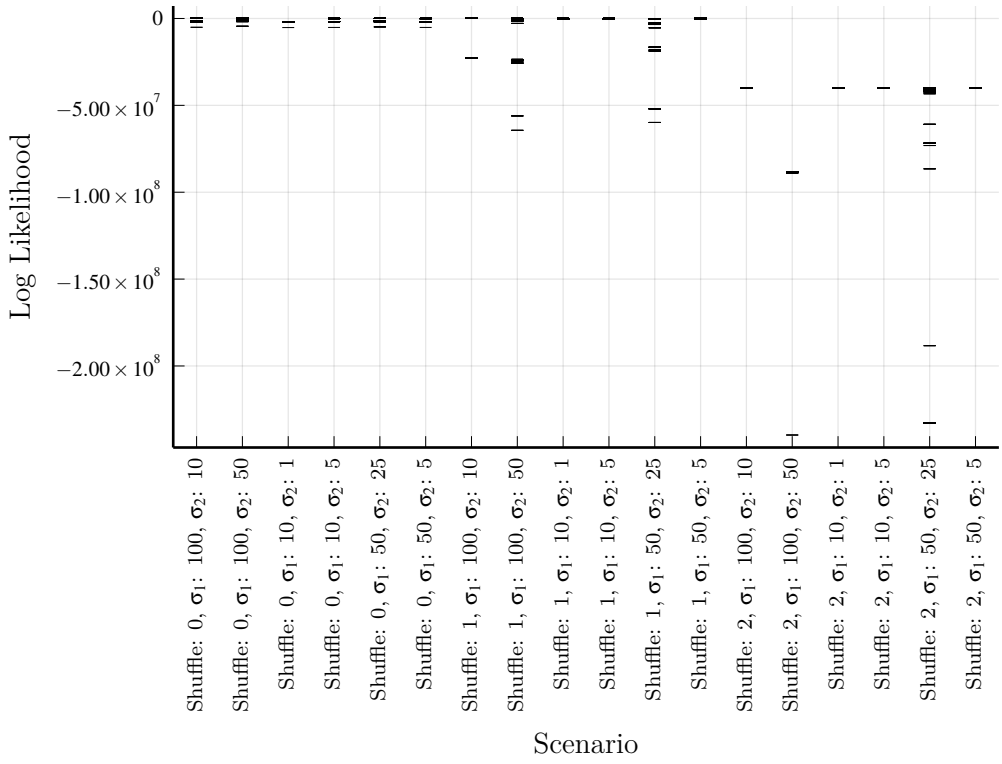


Figure 5.5: Log likelihood over time using Metropolis-Hastings for 3 train scenarios with different levels of arrival and departure sequence shuffle as well as different levels of arrival time variance–denoted by σ_1 in the labels. σ_2 is the variance of the proposal distribution. Shuffle: 0 are not shuffled and thus less difficult, 2 is ths most difficult with the second two trains in reverse order.

here is the planner's performance. Since the planner is so simple, it appears that larger instances quickly become too difficult for it to solve.

As we saw in the 2-train scenarios, the higher the "shuffle" value of the scenario, the lower and more varied the distribution of log-likelihoods is. It is difficult to make a definitive judgment regarding the trend in Figure 5.4 because of how sparse the data points are.

It is easier to see the downward trend of log-likelihoods from MH in Figure 5.5. Much like the 2-train scenarios, the distributions produced by MH are much more clustered closer to zero. Again, This is a positive outcome because the inference steps are mostly spent exploring higher-quality, robust plans.

For MH, looking at the log-likelihood over time is interesting. Figure 5.6 shows this for two selected instances. In Figure 5.5, we can see that the scenario that for the scenario Shuffle: 2 σ_1: 50, σ_2: 25, there are a number of lower log-likelihood samples scattered below the main cluster near the top, but without Figure 5.6, it is unclear how the samples occurred. With the visualization over time, we can see

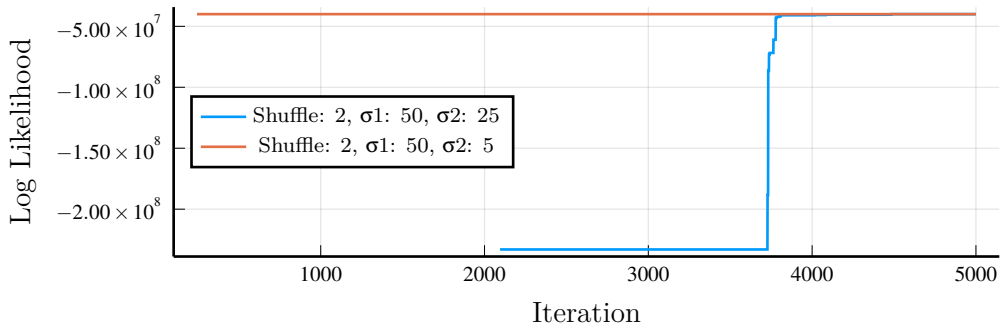


Figure 5.6: Log likelihood of two selected instances of MH over inference steps.

that, compared to the other selected scenario, this one did not find a viable solution until around 2000 steps in, while the other instance found a viable plan to iterate on nearly instantly. The other thing to note with the scenario that takes longer to find a viable plan is that it does not immediately find a robust plan. It first spends some steps stuck at the viable but non-robust plan before evidently traversing the gap to the more robust part of the planning place.

The lag before finding a robust plan could be viewed as both burn-in and autocorrelation. It is another example of burn-in because the iterations before 2000 are spent in the non-viable plan space–essentially an infinitely unlikely part of the distribution of robust plans. It is an example of autocorrelation because once one viable plan is found, it spends more than 1000 steps of inference before breaking out to higher log-likelihoods. As discussed in chapter 2, this is because subsequent steps are correlated. This means that if a previous step was in a bad part of the planning space, the next one is also more likely to be there as well. When using MH, our method is not immune to the weak points of MCMC methods in general, and we should not ignore their possible effects.

5.4 Large Scenario Performance

The obvious results missing are those with more than three trains present. The inability to scale is an unfortunate drawback of the choice of the planning agent. While it should explore the planning space as the number of inference steps approaches infinite, the increase in trains from 3 to 4 already appears too difficult for the agent to provide usable plans within a reasonable time.

For this method, a plan must be reached where all of the trains exit–it is acceptable if they are all late, but they must at least leave the shunting yard. Such a plan must be reached because the model penalizes trains that do not depart with an infinite delay. For inference, the log-likelihood becomes negative infinite, corresponding to zero likelihood.

For IS, this means that all traces are equally bad, giving no information regarding robustness whatsoever–which is no surprise given that a plan should ideally be valid

before its robustness is considered, and no valid plans are found. For MH, the likelihood of 0 means that it never accepts any steps. Even though the MH algorithm does accept some steps at random, this random acceptance is still tied to the log-likelihood of the step. In Gen, this acceptance criterion compares $log(rand())$ with the log-likelihood, and if $log(rand())$ is smaller than the log-likelihood, it accepts the step. The condition is never met with a negative infinite log-likelihood, meaning no steps are even accepted at random.

5.4.1 Attempts to Address Performance

In order to investigate the large-scale performance of the method, two alternatives are attempted.

The first alternative is to run the inference with a longer time limit. In this case, we ran a 4- and 5-train scenario for 8 hours. During that time, neither encountered a plan where all the trains left the yard. This results in neither inference method producing any interesting results. As before, all plans produced by IS are equally bad. It appears that the combinatorial nature of the problem means that the number of possible plans increases too quickly for an agent as poorly performing as the greedy agent in use for these experiments.

The second alternative is to replace the infinite delay for the trains that do not depart with a large, finite number. For example, this could be implemented by assigning a delay of 10000 timesteps to any train that does not depart. It might seem like this should at least aid MH, given that the likelihood of invalid plans will no longer be exactly 0. Non-zero likelihood it should mean that at least some steps should get accepted at random.

It is the case that some steps get accepted at random. Depending on how large of a number is used for the delay, the point at which the MH inference process begins to accept steps varies. However, no further progress is made toward valid plans after that point. Unfortunately, this modification does not bring the method closer to working on larger instances. The only thing that this changes is that MH inference shifts from not accepting any steps whatsoever to accepting steps with invalid plans. The resulting distribution of plans is the same. The distribution is either empty when setting the penalty to infinite or filled with invalid plans with very low likelihoods when setting the penalty to a large number.

Unfortunately, neither alternative produced results for the 4-train scenarios and larger. The planner remains a limiting factor, significantly limiting our method's scalability.

5.5 Robustness Evaluation

While our approach failed to produce results for larger scenarios, we can still evaluate the robustness of the three train scenarios. We use a fixed plan, running it over many sampled arrival times and recording the percentage of times the plan succeeds in solving the scenario. The idea is that, up until this point, we have claimed that

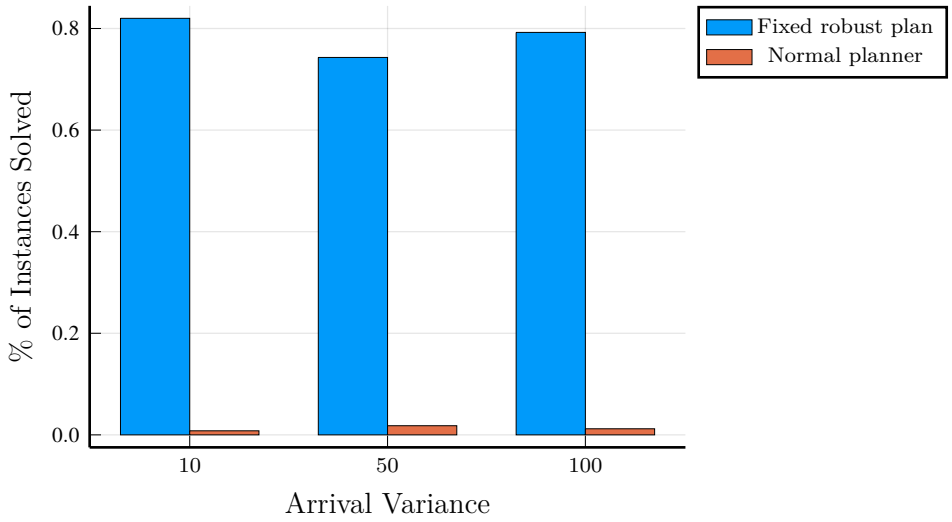


Figure 5.7: A direct evaluation of the robustness of three different 3-train plans. The blue represents the success rate of the inferred robust plan, and the orange represents the success rate of the greedy planner running without any prior plan.

the weight of the distribution corresponds to the robustness of a plan, but we have yet to test that directly.

For this evaluation, the sampling process for the arrival times matches that of inference. Each arrival time is sampled from a normal distribution centered at its corresponding scheduled arrival time. The planning agent replays the chosen plan as it does during MH inference. The difference is that if some action from the plan is not possible, the scenario fails rather than the planner picking up from where the plan failed. The other difference from inference is that the rule for on-time departures is switched back on, meaning that a late departure will also trigger a failed scenario.

Figure 5.7 shows our evaluation of three different arrival variances. We take the plan with the highest likelihood for each one from the corresponding distribution inferred by MH. We use the 3-train scenario with Shuffle: 1 and σ_1 equal to the corresponding arrival variance we are testing. Then we take 5000 samples and record the success rate of both the fixed plan and the greedy planner.

The results may appear impressive–the greedy planner can barely solve any instances. However, it is a somewhat unfair comparison because the fixed robust plan is the best sample out of 5000 steps of inference, whereas the greedy planner is forced to plan from scratch each iteration. While it may not be the most interesting comparison between the robust plan and the plain greedy planner, it is still interesting to see that the percentage of instances solved stays relatively steady across the different arrival variances. This steadiness might mean that the scenarios we are dealing with are not actually made more difficult by the increase in arrival variance. Since the number of trains is still low, it is possible that even with an arrival variance

of 100, it is not that difficult of a problem. The other option is that the method is creating robust enough plans that they continue to perform well at higher variances.

5.6 Conclusion & Answers to Research Questions

Throughout this evaluation of our method, we have seen that, while it performs on the smallest instances, it only scales to those with three trains or fewer. While that is unfortunate in the interest of making practical use of our method, it is still important to note that it did perform as expected on the 2-train scenario we first introduced in chapter 1.

At this point, can also review and address our research questions. The implementation of our method in chapter 4 addressed the first sub-question. There we showed that modeling and inference processes can be set up in such a way as to replicate the real-life planning and replanning process.

The second sub-question (1.2) asks: which inference method can best infer the distribution of plans? We answered this question by comparing IS and MH. From the small-scale results we collected, MH appears to be the favorable choice here. Especially once we moved from 2 to 3-train scenarios, it was hugely beneficial for MH to be able to "remember" the plan it was exploring in the previous step of inference. This behavior meant that once it made it to a robust part of the planning space, it stayed there. MH is the inference method that better infers the distribution over robust plans.

The last research question (1.3) is difficult to answer conclusively. While we did see in section 5.5 that robust plans performed better than the raw planner, it was impossible to compare less and more robust plans. The reason is that there was not much variance in the log-likelihood within the robust part of the planning space–the higher log-likelihood regions. This lack of variance meant only a handful of definitively robust plans were in the distribution–likely because we could only explore small scenario sizes. It seems plausible that there would be more alternatives for robust plans with larger scenarios. To answer the sub-question: while it appears that the high likelihood plans are indeed more robust, there is not sufficient evidence to fully back up that claim.

Chapter 6

Conclusions and Future Work

This thesis has explored a novel way of introducing uncertainty to planning problems built upon simulators and planners that do not initially have the capability to take uncertainty into account. As laid out in the introduction, the main goal of introducing such uncertainty is to infer distributions of solutions robust to the uncertainty. In this case, the TUSP has been used as a concrete example. Many previous solution methods for the TUSP have not considered uncertainty, and those that do must carefully modify the planning algorithm, adding heuristics to guide the search towards more robust solutions. Our work offers a flexible alternative to such methods.

6.1 Key Findings

In our evaluation of the proposed method, we found that, for 2-train scenarios, we can infer a distribution that reflects our idea about what robust plans should look like. Plans from the denser parts of the distributions move trains into separate branches in the shunting yard. This pattern makes trains less likely to conflict with one another, even in the presence of delays. For 3-train scenarios, we found that the inferred robust plans had a much higher success rate than plans produced directly by the greedy planner.

For both 2 and 3-train scenarios, we found that Metropolis-Hastings was more efficient in exploring the planning space. Once it has entered a robust region of the plan space, it does not return to worse regions. importance sampling could not match the efficiency of MH. Since each sample is independent of the others, the plans are constructed from scratch, meaning that IS spends more time exploring worse plans. For this reason, MH is the preferred inference method here.

6.2 Contributions

This work introduced a framework for introducing uncertainty to planning problems. The main benefit of our approach is the flexibility regarding the choice of the

simulator and planner. There are few constraints on the choice of a simulator. So long as the parts of the problem subject to uncertainty can be exposed via input to the simulator, and the rules governing the simulation can be changed easily, it is possible to use the simulator as the core of the generative model.

The same applies to the constraints for the planner as well. The only hard constraint for the planner is that it can interact with the simulator to attempt to provide a solution. The planner's quality will, of course, affect the quality of the outcome of our approach. However, even a naïve planner that does not perform well will not break the approach; it will merely decrease the quality of the resulting distribution of plans.

Thus, in situations where a simulator and planners are available, and they do not yet take uncertainty into account, the approach we have defined here offers a flexible way to introduce uncertainty without having to take the time to modify either the simulator or planner.

6.3 Limitations

The flexibility enabled by our approach does come with some drawbacks. The most critical points that can make the approach unusable are the runtime and performance of a planner. First, a longer runtime for a planner can quickly render our approach too slow for real-world use. Because inference must execute the planner and simulator possibly thousands of times over the course of one run, a planner runtime on the order of a few minutes already becomes an issue. This drawback might be countered by the fact that a slower planner would likely produce higher-quality plans. If inference does not have to waste steps exploring many low-quality, non-robust plans, then fewer steps are needed overall. If that is the case, then the runtime is less of an issue. However, we have not evaluated that scenario, so the trade-off is purely hypothetical.

The second limitation is the planner's performance and how that affects the ability to scale up to larger problem instances. While a poorly performing planner does not stop our method from performing inference on the model, we have seen that the resulting distributions of plans can quickly become sparse when using the greedy planner on larger problem instances. At a certain scale (four trains in our case), the distribution is completely empty, even after a significant number of samples. In this sense, the planner's performance can limit the scalability of our method.

Lastly, a limitation of this thesis is that we did not explore the utility of creating a distribution of plans as opposed to one best plan. The idea of offering many alternative plans is that a human planner can better use the result rather than only having a single option. The human planner could choose from the available plans using a combination of the plans' reported robustness and their domain expertise. However, we first need to scale to larger problem instances before we can investigate the utility of such a feature because there are few alternative plans to consider on the smaller scenarios.

6.4 Future Work

Several future avenues of research have come up throughout this project. The first and most obvious is experimenting further on the TUSP with our framework using better planners. By doing so, we would get a better idea about the capability of our approach and how well it can handle larger-scale, real-world scenarios.

Another exciting idea is to incorporate the use of different planners for different steps of inference. While motivating the use of MH, we described the iterative process as representative of the real world. A plan is created, and then if something changes during the execution of the plan, someone must replan to reflect the new scenario. We represent that in the proposal steps of MH. Each proposal step begins with the most recent plan and executes that until either the end of the scenario or the plan fails. In our current implementation, if the current plan fails, the original planner resumes control, creating a new plan from that point forward. That point where the planner takes over can be referred to as the replanning stage, and there is some work dedicated to creating specialized replanning algorithms that might be interesting to apply in this case. Onomiwo [2020] proposes one such approach. In the context of our framework, it would be interesting to use such a replanning agent for the proposal steps of the MH inference method.

When creating our model, we chose a normal distribution to represent the arrival delays. The real-world distribution likely looks different, and having our model reflect that would be ideal. If we had access to the actual measurements of arrival delays, we could construct a distribution empirically. More realistic arrival times would give us a more accurate distribution of robust plans.

In this work, we focused only on arrival delays and the corresponding departure delays they can cause. As discussed in chapter 2, many variables within the TUSP are stochastic in the real world. Such variables might be the duration of service tasks, the time to couple and decouple trains, the time to start and stop a train–even the time different personnel take to walk between points in the yard. Compared to arrivals and departures, making any of these variables stochastic would add uncertainty within a scenario, not just at the beginning and end. These internal uncertainties would open up the opportunity for incremental measures of success. It would be helpful for incremental inference methods like Metropolis-Hastings (MH) to understand that one plan is closer to the robust part of the planning space than another, even if neither of them are good plans.

Incremental success measures also allow the use of another class of inference methods discussed in chapter 2: particle filtering. We could not use particle filtering with our model because there were no intermediate points in the model to pause and evaluate the likelihood of the partially-completed plan. Particle filtering has the benefit of evaluating many alternatives in parallel, something that an approach like Metropolis-Hastings (MH) does not offer as easily. Parallel evaluation would become especially useful if a slower planner is used because it would lessen the impact of the planner's runtime. Therefore, exploring the addition of internal uncertainties and applying particle filtering would be interesting next steps for future research.

Acronyms

IS importance sampling. 10, 21, 22, 26, 27, 30, 32, 33, 35, 37

MC Monte Carlo. 10

MCMC Markov chain Monte Carlo. 10, 11, 22, 29, 32

MH Metropolis-Hastings. vii, 11, 22, 26–29, 31–35, 37, 39

PP Probabilistic Programming. 2, 5, 9, 13

PPL probabilistic programming language. 2, 9, 10

TORS Treinonderhoud-en-rangeersimulator/Train Maintenance and Shunting Simulator. 2, 8, 9, 13, 19–21, 23, 26

TUSP train unit shunting problem. 2, 3, 5, 7, 8, 12, 13, 19–23, 37, 39

Bibliography

Casper Athmer. An Evolutionary Algorithm for the Train Unit Shunting and Servicing Problem. Master's thesis, Delft University of Technology, 2021. URL http://resolver.tudelft.nl/uuid:0a65376e-bbb1-440b-95a2-586a369d9e98.

Atilim Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip Torr, Victor Lee, Kyle Cranmer, Prabhat, and Frank Wood. Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–24, Denver Colorado, November 2019. ACM. ISBN 978-1-4503-6229-0. doi: 10.1145/3295500.3356180. URL https://dl.acm.org/doi/10.1145/3295500.3356180.

Atılım Güneş Baydin, Lukas Heinrich, Wahid Bhimji, Lei Shao, Saeid Naderiparizi, Andreas Munk, Jialin Liu, Bradley Gram-Hansen, Gilles Louppe, Lawrence Meadows, Philip Torr, Victor Lee, Prabhat, Kyle Cranmer, and Frank Wood. Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model, February 2020. URL http://arxiv.org/abs/1807.07706.

Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. Transportation Research Part B: Methodological, 63:15–37, May 2014. ISSN 0191-2615. doi: 10.1016/j.trb.2014.01.009. URL https://www.sciencedirect.com/science/article/pii/S0191261514000198.

Richard H Connelly and F Lockwood Morris. A generalization of the trie data structure. Mathematical structures in computer science, 5(3):381–418, 1995. Publisher: Cambridge University Press.

Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019,

pages 221–236, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314642. URL https://doi.org/10.1145/3314221.3314642.

Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

Richard Freling, Ramon Lentink, Leo Kroon, and Dennis Huisman. Shunting of Passenger Train Units in a Railway Station. Transportation Science, 39 (2):261–272, May 2005. ISSN 0041-1655. doi: 10.1287/trsc.1030.0076. URL https://pubsonline.informs.org/doi/abs/10.1287/trsc.1030.0076.

Noah D Goodman, Joshua B. Tenenbaum, and The ProbMods Contributors. Probabilistic Models of Cognition, 2016. URL http://probmods.org. Edition: Second.

John J. Grefenstette, Shawn T. Brown, Roni Rosenfeld, Jay DePasse, Nathan TB Stone, Phillip C. Cooley, William D. Wheaton, Alona Fyshe, David D. Galloway, Anuroop Sriram, Hasan Guclu, Thomas Abraham, and Donald S. Burke. FRED (A Framework for Reconstructing Epidemic Dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations. BMC Public Health, 13(1):940, October 2013. ISSN 1471-2458. doi: 10.1186/1471-2458-13-940. URL https://doi.org/10.1186/1471-2458-13-940.

Michael U. Gutmann and Jukka Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. Journal of Machine Learning Research, 17(1):4256–4302, January 2016. ISSN 1532-4435. MAG ID: 2964129402.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1):97–109, April 1970. ISSN 0006-3444. doi: 10.1093/biomet/57.1.97. URL https://doi.org/10.1093/biomet/57.1.97.

E. H. R. Huizingh. Planning first-line services on a NS service station : an exact approach. info:eu-repo/semantics/masterThesis, University of Twente, Twente, NL, May 2018. URL https://essay.utwente.nl/74968/.

J. G. V. D. Linden, J. Mulderij, B. Huisman, Joris W. Den Ouden, M. Akker, H. Hoogeveen, and M. D. Weerdt. TORS: A Train Unit Shunting and Servicing Simulator. AAMAS, 2021. doi: 10.5555/3463952.3464237.

Alexander Lavin, David Krakauer, Hector Zenil, Justin Gottschlich, Tim Mattson, Johann Brehmer, Anima Anandkumar, Sanjay Choudry, Kamil Rocki, Atılım Güneş Baydin, Carina Prunkl, Brooks Paige, Olexandr Isayev, Erik Peterson, Peter L. McMahon, Jakob Macke, Kyle Cranmer, Jiaxin Zhang, Haruko Wainwright, Adi Hanuka, Manuela Veloso, Samuel Assefa, Stephan Zheng, and Avi Pfeffer. Simulation Intelligence: Towards a New Generation of Scientific Methods, November 2022. URL http://arxiv.org/abs/2112.03235.

Ramon Lentink. Algorithmic Decision Support for Shunt Planning. PhD Thesis, Erasmus Research Institute of Management, February 2006. URL http://hdl.handle.net/1765/7328.

Edward Meeds and Max Welling. Optimization Monte Carlo: Efficient and Embarrassingly Parallel Likelihood-Free Inference, December 2015. URL http://arxiv.org/abs/1506.03693.

NS. NS Annual Report 2021. Technical report, Nederlandse Spoorwegen, 2021. URL https://www.nsannualreport.nl/annual-report-2021/reading-guide.

E. a. S. Onomiwo. Disruption Management on Shunting Yards with Tabu Search and Simulated Annealing. Master's thesis, Utrecht University, Utrecht, NL, 2020. URL https://studenttheses.uu.nl/handle/20.500.12932/37757.

Adrian E Raftery and Steven M Lewis. Implementing MCMC. In David J Spiegelhalter, editor, Markov chain Monte Carlo in practice, pages 115–130. Chapman & Hall, 1996.

Nikolaos V. Sahinidis. Optimization under uncertainty: state-of-the-art and opportunities. Computers & Chemical Engineering, 28(6):971–983, June 2004. ISSN 0098-1354. doi: 10.1016/j.compchemeng.2003.09.017. URL https://www.sciencedirect.com/science/article/pii/S0098135403002369.

Jacob Trepat Borecka, Nikola Bešinović, Yousef M. Maknoon, Rob M. P. Goverde, and Wan-Jui Lee. Solving the train unit shunting problem using multi-agent deep reinforcement learning with routing optimization. In 2021 INFORMS Annual Meeting, page 12 p., October 2021. doi: 10.3929/ETHZ-B-000513034. URL http://hdl.handle.net/20.500.11850/513034.

D. A. van Cuilenborg. Train Unit Shunting Problem, a Multi-Agent Pathfinding approach. Master's thesis, Delft University of Technology, 2020. URL http://resolver.tudelft.nl/uuid:c9a0f41c-de6f-4ef4-8d94-6d0a89df3eec.

Roel van den Broek, Han Hoogeveen, and Marjan van den Akker. Personnel Scheduling on Railway Yards. Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, page 15, 2020. doi: 10.4230/oasics.atmos.2020.12.

Roel Wemarus van den Broek. Towards a Robust Planning of Train Shunting and Servicing:. PhD Thesis, Utrecht University, May 2022. URL https://dspace.library.uu.nl/handle/1874/420470.

R.W. van den Broek. Train Shunting and Service Scheduling: an integrated local search approach. Master's thesis, Utrecht University, January 2016. URL https://studenttheses.uu.nl/handle/20.500.12932/24118.

Frank Wood, Andrew Warrington, Saeid Naderiparizi, Christian Weilbach, Vaden Masrani, William Harvey, Adam Ścibior, Boyan Beronov, John Grefenstette, Duncan Campbell, and S. Ali Nasseri. Planning as Inference in Epidemiological Dynamics Models. Frontiers in Artificial Intelligence, 4, 2022. ISSN 2624-8212. doi: 10.3389/frai.2021.550603. URL https://www.frontiersin.org/articles/10.3389/frai.2021.550603.

Appendix A

Assorted Pseudocode

A.1 Greedy Planning Agent

Pseudo code for the greedy planning planning agent.

```
def greedy_planner(self, state, epsilon, existing_plan, possible_actions):
    # Apply existing plan, if specified
    if existing_plan is not None:
        next_action_from_existing_plan = # get next action from existing plan
        if next_action_from_existing_plan is not None:
            return next_action_from_existing_plan

    # Otherwise continue with greedy plan
    action_priority = sort([
        get_action_priority(train, possible_actions)
        for train in state.trains.values()
    ])

    # If there is an arrival or exit action, take it
    for action in action_priority:
        if action is ExitAction or action is ArriveAction:
            return action

    # Choose random action with probability epsilon
    epsilon_choice = random.random()
    if epsilon_choice < epsilon:
        action = random.choice(possible_actions)
        return action

    # Otherwise, choose the best action
    return action_priority[0]
```

```
def get_action_priority(train, state, possible_actions):
        priority = [(0, possible_actions[0])]
        if state.time == train.arrival_time:
            # Add arrival action with priority 100 if it is an available action
        if state.time >= train.arrival_time:
            if (
                train.current_track() == train.end_track
                and train.end_side_track in train.current_track().next_tracks()
            ):
                # If the train is at its end track, facing the right way, and
                if train.is_moving():
                    # If it is still moving, add end move action with priority 5 instead
                else:
                    # If the train is at its exit track
                    # add exit action with priority 100
            else:
                if not train.is_moving():
                    # Add a move action with priority 5
                else:
                    # If already moving, get path to the end track and add a move action
                    # with priority 5 to the next track on the path. If the train is facing
                    # the wrong way, add a setback action with priority 20 instead.
            # Finally, add a wait action with priority 1 in case all of the other cases fail
        return priority
```

A.2 Sequence Shuffling

Function used to shuffle scenario arrival/departure sequences to vary the difficulty reverses the last number_of_reversals in the list of trains.

```
def shuffle(trains: list[int], number_of_reversals: int):
    if number_of_reversals == 0:
        return trains
    return trains[:-number_of_reversals] + trains[-number_of_reversals:][::-1]
```

Appendix B

Selected 2-Train Plan

The following provides an example shunt plan and track layout inferred for a 2 train scenario as discussed in chapter 5.

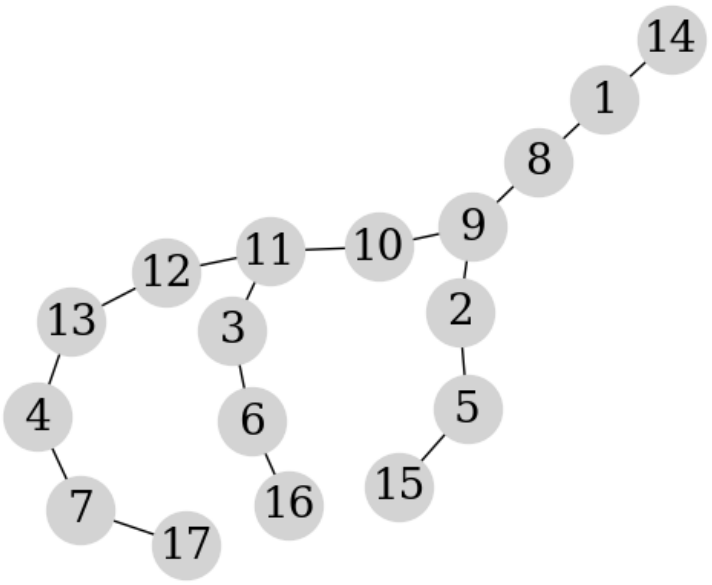


Figure B.1: Shunt yard layout for the 2 train scenarios. Node 14 acts as the entrance to the yard and does not count as a normal track. Entrance and exit actions correspond with node 1 instead.

Train ID #	Action	
0	Arrive	
0	BeginMove	
0	Walking	
0	EndMove	
0	Walking	
0	BeginMove	
0	Move	Path: 1A→8B
0	Move	Path: 8B→9→10B
0	Move	Path: 10B→11→3B
0	Walking	
1	Arrive	
1	BeginMove	
1	Move	Path: 1A→8B
1	Move	Path: 8B→9→10B
1	Move	Path: 10B→11→12B
1	Walking	
0	EndMove	
0	BeginMove	
0	Move	Path: 3B→11→10B
0	Move	Path: 10A→9→8B
0	Move	Path: 8A→1B
0	EndMove	
0	Walking	
1	Move	Path: 12A→11→10B
1	Move	Path: 10A→9→8B
1	Walking	
0	Walking	
1	Move	Path: 8B→9→2B
1	EndMove	
1	BeginMove	
1	Walking	
1	Move	Path: 2B→9→8B
1	EndMove	
0	Exit	Time: 1300
1	BeginMove	
1	Move	Path: 8A→1B
1	EndMove	
1	Exit	Time: 1700

Table B.1: Sequence of actions in a selected 2-train plan. The associated layout and track IDs for the movement actions can be found in Figure B.1. Train 0 is scheduled to arrive at timestep 400, and train 1 at 800. They are scheduled to depart at timestep 1300 and 1700 respectively.