UCLouvain

École polytechnique de Louvain

Algorithms for statistical model checking

Author: **Firmin CHENOY**
Supervisors: **Axel LEGAY, Maxime PARMENTIER**
Readers: **Axel LEGAY, Maxime PARMENTIER, Eduard BARANOV, Alexander GERNIERS**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Acknowledgements

I would like to express my sincere gratitude and appreciation to everyone who has contributed to the completion of this master thesis. Over the course of nearly a year, I have received invaluable support and assistance from numerous individuals, without whom this accomplishment would not have been possible.

First and foremost, I extend my heartfelt thanks to my supervisors, Axel Legay and Maxime Parmentier for their unwavering availability, guidance, and insightful feedback throughout this research endeavour. Their dedication, expertise, and willingness to share their knowledge have greatly enriched the quality of this work.

I am also grateful to the readers of this thesis, as well as those who will show interest in the findings and conclusions presented in this thesis.

Lastly, I an also grateful to my family and friends for their unwavering support, understanding, and encouragement throughout this journey.

Contents

1 Introduction

In an increasingly complex and interconnected world, ensuring the reliability and correctness of systems and processes is of utmost importance. One approach that has gained significant attention in recent years is statistical model checking. This innovative methodology combines principles from statistical analysis and formal verification to assess the behaviour of systems probabilistically. By harnessing the power of statistics, we can gain valuable insights into the performance and reliability of systems that exhibit stochastic characteristics.

Imagine a transportation network comprising autonomous vehicles operating in a bustling city [1]. To guarantee the safety and efficiency of this network, we need to assess various aspects such as collision probabilities, traffic flow, and optimal decision-making strategies. Statistical model checking allows us to analyse such complex systems, leveraging the power of statistical model checking algorithms to draw meaningful insights and guide decision-making.

The first step to achieve this goal is to establish a solid foundation in the fundamental concepts. This includes understanding essential notions such as model checking, statistical model checking, Markov decision processes, and Markov chains. By comprehending these concepts, we lay the necessary groundwork to delve into the core subject of this thesis.

Building upon this knowledge, we then shift our focus to a specific statistical model checking algorithm: the Smart Sampling algorithm. This algorithm employs intelligent sampling strategies to efficiently explore the vast state spaces of complex systems, facilitating the verification process. A key aspect of this exploration involves schedulers whose evaluation plays a vital role in guiding the execution and decision-making process of the algorithm.

Moving forward, we delve into the exciting realm of parameter modification. Understanding the behaviour of the smart sampling algorithm is essential for its effective and efficient application. By systematically modifying various parameters, we aim to unravel insights into the algorithm's behaviour, identify potential limitations, problems, and devise strategies to enhance its effectiveness. This phase of exploration opens new avenues for optimising the algorithm.

Finally, we venture into the realm of innovation and experimentation. By introducing variations to the existing algorithm, we seek to push the boundaries of its effectiveness and efficiency. These modifications aim to address specific challenges and shortcomings, leveraging novel approaches to unlock new levels of performance and utility. Through these innovative enhancements, we strive to contribute to the ongoing evolution of statistical model checking algorithms.

2 Background

2.1 Model Checking

Model checking [2, 3, 4, 5] is a collection of methods used to verify the correctness of a system or software by comparing it against a set of formal specifications or properties. They consist of analysing the system or software using mathematical and logical techniques to determine if it meets the desired properties or specifications. One can use model checking for a variety of reasons, including:

- Verifying the correctness of a system, model, software or parameter [6]: model checking can be used to ensure that a system, model, software or parameter behaves as expected, and that it meets the desired properties or specifications.
- Finding errors and bugs: model checking can be used to identify errors or bugs in a system, model, software or parameter that might not be immediately obvious, helping the creators to improve the overall quality of the system, model, software or parameter.
- Improving performance: model checking can be used to optimise the performance of a system, model, software or parameter by identifying bottlenecks or areas that can be improved.
- Automation testing: model checking can be used to automate the testing of a system, model or software, making it more efficient and less prone to human error.
- Improving safety and security: Model checking can be used to improve the safety and security of a system, model, software or parameter by identifying potential vulnerabilities or attack points.

To verify a property with model checking, we need to define the property that we want to verify. This is typically done using a formal language, such as temporal logic, that allows us to express properties in a precise and unambiguous way (see section 2.5 for more details). Once the property is defined, we need to build a model of the system or design that we want to verify. This model should accurately represent the behaviour and interactions of the system or design. We typically use transition state diagrams where every node represents a state of the system while the edges are the possible transition between the states. Then, we use a model checking algorithm to analyse the model and compare it against the defined property. The algorithm will explore multiple possible executions of the model, and will determine if the property is satisfied for all of these executions (called traces). If the property is satisfied for all possible traces, the model is considered to be correct and to meet the desired

properties or specifications. If the property is not satisfied far all possible executions, the model is considered to be incorrect and the system, the software or the design may need to be redesigned or modified.

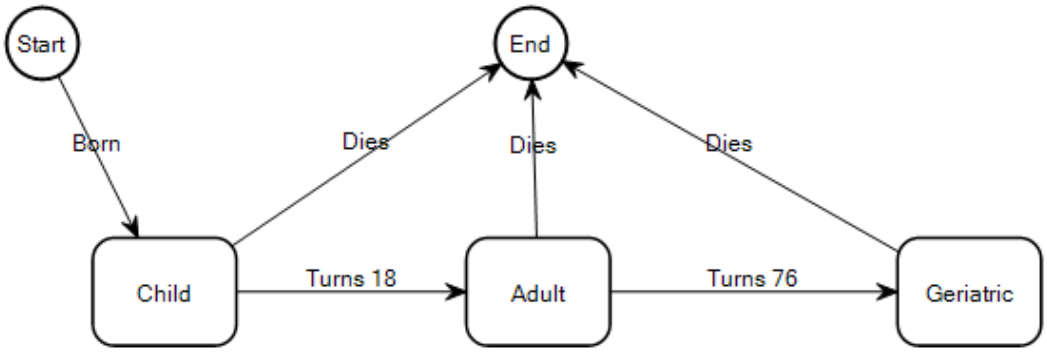

Figure 1: Transition State Diagram

An example of using model checking to verify a property would be checking whether a traffic light system is safe and operates correctly [7]. The property that we want to verify is: $\ll$ at no time will a red light be on at the same time as a green light at any intersection $\gg$. We first build a model of the traffic light system, including the states of the lights (red, yellow, green) and the transitions between these states. We then use a model checking algorithm to analyse the model and compare it against the defined property. The algorithm will explore all possible states of the lights, and will determine if the property is satisfied in all of these states. In this example, model checking is used to ensure the safety and correct operation of the traffic light system, by verifying that the property of no red and green light at the same time is met in all possible states of the system.

Model checking can also be used to evaluate the value of a parameter along every possible executions. To evaluate a parameter, we first need to define it, then, we need to specify a property using formal language, such as temporal logic. This property should specify the desired evolution of the parameter along each possible execution. By evaluating the state of the parameter along every possible trace of execution, model checking can provide a comprehensive analysis of the system's behaviour and help identify any issues or anomalies with regards to the parameter.

An example of using model checking to evaluate a parameter would be the order of the traffic light system. First, we must define the system model. A finite state machine can be used to model the behaviour of the traffic light. The states in the machine represent the different colours of the light (red, orange,

and green) and the transitions between states represent the changes in the light's colour. This is the property we want to verify would be "green light should always follow a red light". This can be expressed in a temporal logic formula as (see section 1.5 for more information about the operators):

$$G(red \Rightarrow Xgreen)$$

Then, a model checker is run on the finite state machine to verify the property. The model checker will explore all possible executions and evaluate the state of the light along each trace. The results of the model checking process will indicate whether the light transition satisfies the desired property along every possible execution. If the property is not satisfied, the model checker will provide a counterexample that shows when the property fails along a trace of execution. This example shows how model checking can be used to ensure that a system satisfies a desired property along every possible execution.

Model checking is a powerful tool for ensuring the correctness and quality of a system, model, software or parameter but it has downsides too:

- Complexity: model checking can be a complex and time-consuming process, especially for large and complex systems.
- Computational cost: model checking can be computationally expensive, requiring significant processing power and memory. This can be a limitation for large or complex systems, and can make model checking impractical for some applications.
- Limited expressiveness: model checking is based on formal languages, and some properties can be difficult to express in those languages.
- Limited scope: model checking is a verification technique, and it may not be able to identify all the issues that the system might have, it is not a debugging technique, for example.
- Assumptions made: model checking is based on a model, and the results depend on the quality of the model and the assumptions made with it, which can lead to a wrong conclusion.

These downsides limit the scalability of model checking as the number execution traces that need to be checked increases exponentially with the number of variables used to represent the system. This is a reason why model checking is typically limited to small or moderately sized systems. This problem is known as the state explosion problem.

The state explosion problem [8, 9] appears when the number of components or variables in the system increases significantly which makes the number of

possible states grows exponentially. This exponential growth leads to the state explosion problem, making it computationally infeasible to explore or analyse all possible traces of the system. This problem poses significant challenges for model checking tools and techniques. It severely limits the size and complexity of systems that can be effectively analysed. Even systems with a relatively small number of components or variables can lead to an explosion in the number of states, making the verification process impractical or intractable

2.2 Statistical Model Checking

Statistical model checking (SMC) [10, 11, 12, 13, 14] is an approach to formal verification that combines elements of model checking and statistical inference. While traditional model checking focuses on exhaustively analysing all possible states of a system, SMC leverages statistical techniques to analyse system behaviour based on a statistical sampling of system executions.

In statistical model checking, a model of the system under analysis is defined, typically represented as a probabilistic or stochastic model such as a Markov chain or a Markov decision process (section 3.3). The model captures the system's states, transitions, and probabilistic behaviour. Instead of exhaustively checking all possible states and transitions, SMC randomly samples system executions according to a predefined statistical distribution.

The key difference between traditional model checking and statistical model checking lies in the analysis process. In traditional model checking, exhaustive exploration of all possible states and transitions is performed, ensuring precise verification results but suffering from the state explosion problem. In contrast, statistical model checking relies on sampling a subset of system executions and inferring properties of the system based on the observed behaviour.

The advantage of statistical model checking is that it can provide insights into system behaviour with reduced computational complexity compared to exhaustive model checking. By leveraging statistical sampling, SMC allows for analysis of larger and more complex systems while still providing meaningful verification results. However, it is important to note that SMC introduces a trade-off between precision and computational efficiency, as statistical estimates are subject to sampling errors and confidence intervals.

There exist multiple approaches of statistical model checking, each with their own specific techniques and methods. Some common types of statistical model checking approaches are:

- Monte Carlo model Checking: This type of statistical model checking involves generating random inputs and simulating the system with these

inputs to gather statistics on the system's behaviour. The model checker uses these statistics to validate the results of formal verification methods, such as model checking.

- Stochastic model Checking: This type of statistical model checking involves modelling the system as a stochastic process and using statistical analysis to validate the results of formal verification methods.
- Hybrid model Checking: This type of statistical model checking combines traditional model checking with statistical analysis to provide a more comprehensive evaluation of the system's behaviour.
- Confidence Interval model Checking: This type of statistical model checking involves generating a confidence interval for the estimation of one parameter of the system and using this interval to validate the results of formal verification methods.

These are just a few examples of the different types of statistical model checking approaches, and there may be other variations as well. The choice of a specific type of statistical model checking strategy depends on the specific requirements of the system and the properties that needs to be verified

Statistical model checking (SMC) can be used for a variety of reasons, including:

- Handling large and complex systems: SMC can be used to verify properties of large and complex systems for which is not feasible to analyse using traditional model checking.
- Handling uncertain environments: SMC is particularly useful for systems that operate in uncertain environments, such as systems that involve randomness or noise. It allows us to estimate the probability that a property holds for a large number of traces, even in the presence of uncertainty.
- Improving efficiency: SMC can be more efficient than traditional model checking, as it does not require exploring all possible traces of the system. This can reduce the computational cost and time required to verify properties of a system.
- Improving robustness: SMC can be used to analyse the robustness of a system by estimating the probability that a property holds for a large number of traces, even in the presence of uncertainty or errors.
- Improving the reliability of results: SMC can be used to estimate the probability that a property holds for a large number of traces, which can provide more reliable results than traditional model checking.

- Improving the scope of analysis: SMC can be used to analyse systems that exhibit both discrete and continuous behaviour, by estimating the probability that a property holds for a large number of traces, even if the system has a continuous behaviour.

Overall, SMC provides a powerful tool for the verification of complex and uncertain systems, it can be more efficient and reliable than traditional model checking, and it can be used to analyse systems that exhibit both discrete and continuous behaviour.

SMC is used in plenty of situations, for example to analyse the reliability of a power grid [15]. A power grid is a complex system with a large number of components and interactions. The property that we want to verify in this example is that "the power grid will not experience a blackout for more than 1 hour in any given year." We first build a model of the power grid, including the states of the components and the interactions between them. We then use a statistical model checking algorithm to analyse the model and estimate the probability that the property holds for random execution of the system/model. The algorithm generates execution traces of the power grid model and will determine the probability that the property is satisfied with each trace. For example, the algorithm might find that the probability of a blackout lasting more than 1 hour is 0.01%. This would indicate that the power grid is highly reliable, and that the property is likely to be satisfied in 99.99% of cases. However, if the algorithm finds that the probability of a blackout lasting more than 1 hour is, for example, 10%, this would indicate that the power grid is not reliable and that the property is likely to be satisfied in only 90% of cases. This could lead to a redesign of the power grid to improve its reliability.

In statistical model checking, precision and bound are important concepts that relate to the accuracy and scope of the analysis. Precision (epsilon) refers to the level of detail or granularity in the analysis. It measures how accurately the statistical model checking technique can capture the behaviour of the system being analysed. Higher precision means that the analysis can capture more subtle aspects of the system's behaviour, while lower precision may result in a coarser approximation. Increasing precision often requires more computational resources and can lead to longer analysis times. On the other hand, bound (delta) refers to the scope or range of the analysis. It defines the extent of the system or properties that are considered during the analysis. The bound can be specified in terms of time steps, iterations, or any other relevant metric. Setting a larger bound allows for a more comprehensive analysis, covering a longer duration or larger state space. However, increasing the bound typically increases the computational complexity and can make the analysis more time-consuming.

Both precision and bound need to be carefully considered based on the

specific requirements of the analysis. Striking a balance between precision and bound is important to ensure that the analysis is both accurate and efficient. Adjusting these parameters appropriately can help achieve a reasonable trade-off between the level of detail captured and the computational resources required for the analysis.

Overall, statistical model checking can provide a powerful tool for the verification of complex and uncertain systems, and it can be a valuable addition to the software engineering process. However, it is important to keep in mind that it is based on estimation and assumptions, and it may not be as accurate or reliable as traditional model checking.

2.3 Markov Decision Process, Markov Chain and Schedulers

2.3.1 Markov Decision Process

A Markov Decision Process (MDP) [16, 17, 18] is a mathematical model commonly used in statistical model checking for modelling decision-making in situations where the outcome is uncertain. It consists of a 4-tuple (S, A, P_a, R_a):

- S is a set of states that represent the possible states of the system. S is called the state space and contain the initial state.
- A is a set of actions that represent the decisions that can be made by an agent in each state. These actions can have multiple branches with different probabilities associated to them. A is called the action space. A_s is the set of actions available from the state S.
- P_a Defines the probability of moving from one state to another after taking a specific action. $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action an in-state s at time t will lead to state s' at time t+1
- R_a is the reward received after transitioning from state s to state s' due to action a. This reward function is not mandatory and will not be used in this thesis.

In an MDP, an agent takes actions in each state to maximise its expected cumulative reward over time. The goal of an MDP optimisation algorithm is to find an optimal policy, which is a mapping from states to actions that maximises the expected cumulative reward.

There exist two types of MDP: continuous Markov Decision Processes (MDPs) and discrete MDPs. Their difference lies in the nature of the state and action spaces. In discrete MDPs, both the state space and action space are finite and

discrete. This means that there is a finite and countable set of states and actions available to the agent. The agent transitions from one state to another by taking discrete actions, and the transitions between states are governed by transition probabilities. Examples of discrete MDPs include grid world problems, chess games, and inventory control. However, in continuous MDPs, either the state space, action space, or both are continuous. This means that the states and/or actions can take on real-valued or continuous values. In continuous MDPs, the agent can choose from a potentially infinite number of actions and the state transitions are typically described by continuous dynamics or differential equations. Continuous MDPs are commonly used to model problems in robotics, control systems, and optimisation. In This master thesis, we will focus on discrete MDPs.

MDPs are used in a variety of applications, including robotics, resource allocation, and game theory. They are useful because they provide a framework for modelling decision-making problems that are uncertain, dynamic, and involve multiple interacting components. They can be used to evaluate the performance of different policies or to compare different strategies.

A simple example of a Markov Decision Process is the coffee machine example [19]. We use a MDP to model a system where the states represent the different configurations of the coffee machine (e.g. empty,"coffee in the pot", "empty cup", "full cup"), the actions represent the different operations that can be performed (e.g. "fill the pot", "empty the pot", "put a cup", "press the button"), and the transition probabilities represent the likelihood of moving from one state to another based on the chosen action. The initial state is the starting configuration of the coffee machine.

The goal of a Markov Decision Process (MDP) for the coffee machine example can vary depending on the specific scenario and objectives. It could be minimising the total time to make a cup of coffee: In this scenario, the reward function would incentivise quick and efficient actions to make a cup of coffee as soon as possible. Or it could be maximising the number of cups made per hour: In this scenario, the reward function would incentivise actions that result in the maximum number of cups made per hour, such as filling the pot and putting a cup as quickly as possible. Or it could also be minimise waste: In this scenario, the reward function would incentivise actions that result in the least amount of coffee being wasted, such as making a cup only when there is a demand for it. Or it could be maximising profit: In this scenario, the reward function would incentivise actions that result in the maximum profit, such as charging more for each cup of coffee made. The specific goal of the coffee machine MDP can be adjusted based on the desired outcomes, and the Markov Decision Process can be used to find the optimal policy to achieve those goals.

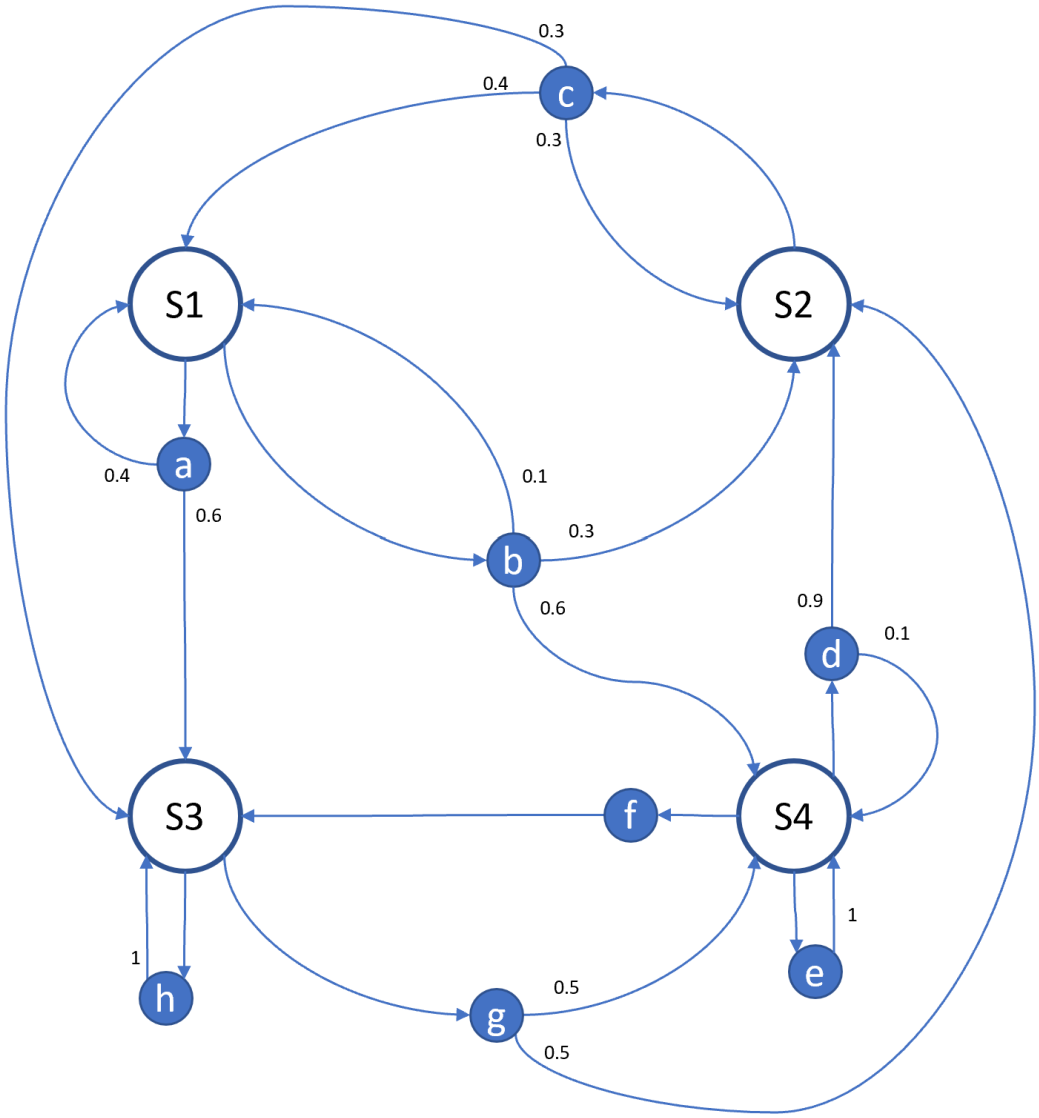

Figure 2: Markov Decision Process

2.3.2 Markov Chain

A Markov Chain [20, 21, 18] is a mathematical model for a sequence of events in which the probability of each event depends only on the state attained in the previous event. In other words, it is a mathematical model for a system that moves from one state to another state based on fixed probabilities, and the probability of transitioning from one state to another is independent of the states that came before it. A Markov Chain is represented as a directed graph, with each node representing a state and each edge representing a transition

between states with a specified transition probability.

A Markov Chain can be obtained from a Markov Decision Process by removing the decision-making component and fix the transitions between states. This can be done by selecting a specific policy that determines which action to take in each state. The policy can be deterministic or stochastic, and once it is chosen, the transitions between states become fixed, and the resulting model becomes a Markov Chain. For example, consider an MDP model of a coffee machine with two states: "Full" and "Empty". The agent has two actions: "Refill" and "Do Nothing". If the policy is to always refill the machine when it's empty, then the resulting Markov Chain would have two states and one transition between them, with a fixed probability of transitioning from "Empty" to "Full". In summary, a Markov Chain can be obtained from a Markov Decision Process by fixing the transitions between the states based on a chosen policy, resulting in a system that can only be modelled as a sequence of events without any decision-making.

Markov Chain and Markov Decision Process (MDP) are two related concepts in the field of probability and operations research. Both are mathematical models used to describe and analyse the behaviour of systems over time. A Markov Chain is a type of mathematical model that represents a sequence of events over time, where the probability of each event depends only on the state of the previous event. It is a memoryless system, meaning that the future state of the system is independent of its past history. On the other hand, a Markov Decision Process is an extension of the concept of Markov Chain, where the system is controlled by the decisions made by an agent. In an MDP, the state of the system is influenced by decisions made by an agent at each time step. The agent can choose its actions based on a reward function, which assigns a reward to each state and action. His goal is to maximise the accumulated reward over time. The agent can also choose its actions randomly or with predefined policy in which case the goal is to verify a property or parameter of the system. In summary, while both Markov Chain and MDP are used to model systems over time and that one is policy restriction of the other, the key difference between them is that a Markov Chain is a passive system that only depends on the previous state, while an MDP is an active system that is influenced by the decisions made by an agent.

2.3.3 Schedulers

To get a Markov Chain from a Markov Decision Process (MDP), we use a scheduler [18] (also called policies) which select a particular action to perform in each state. The scheduler can be either deterministic or probabilistic. Deterministic schedulers operate based on predefined rules or algorithms without

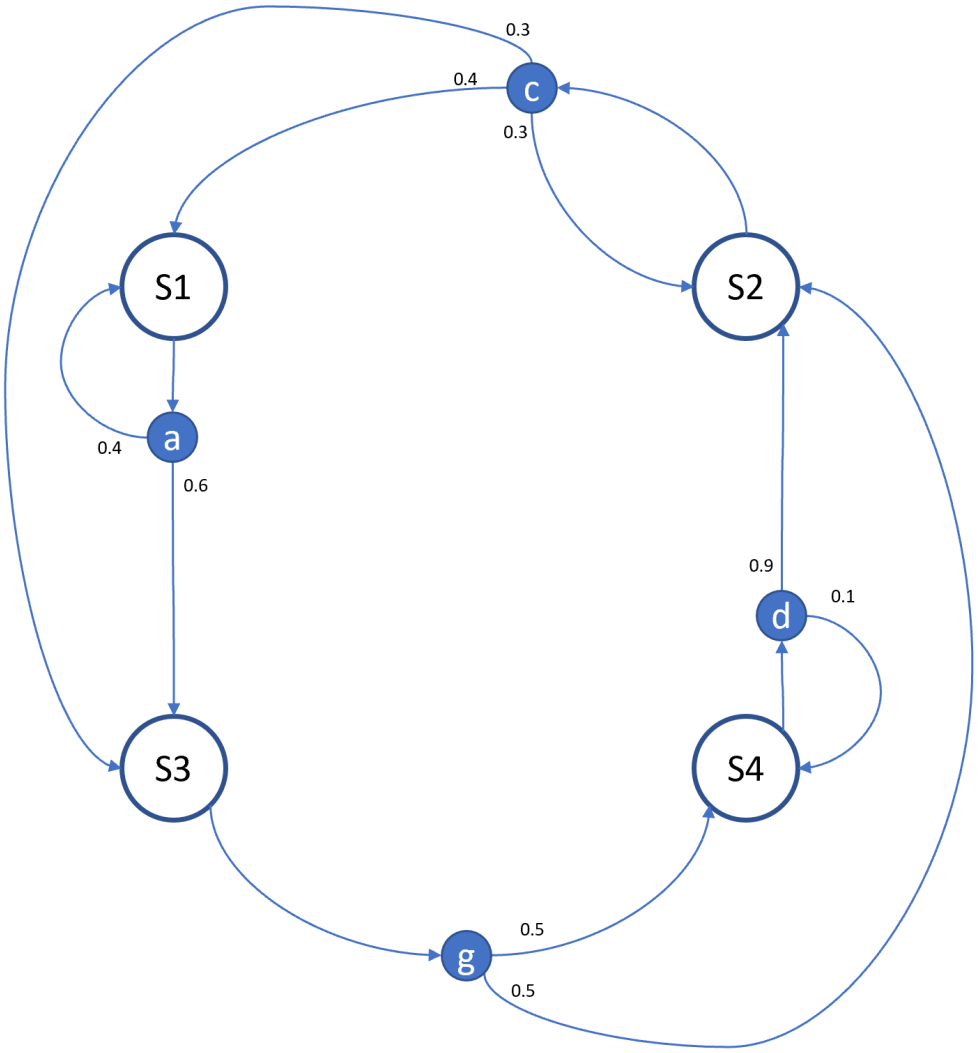

Figure 3: Markov Chain

any randomness or uncertainty. They follow a fixed scheduling policy that determines the execution order. The decisions made by deterministic schedulers are predictable and repeatable, as they are solely based on the current system state and predefined rules. On the other hand, probabilistic schedulers introduce randomness and uncertainty into the scheduling decisions. Instead of relying solely on predefined rules, probabilistic schedulers use probability distributions or statistical models to make scheduling choices. The decisions made by probabilistic schedulers may vary each time, as they take into account probabilistic factors and stochastic behaviour. These schedulers consider factors such as task arrival patterns, execution times, and resource availability probabilities to determine the scheduling order. Overall, the scheduler is a

crucial aspect of MDPs, as it determines the type of Markov Chain that is obtained from the MDP.

In this thesis, we will focus on memoryless schedulers. A memoryless scheduler determines the order and allocation of resources based solely on the current state of the system, such as the availability of resources, the arrival of new tasks, or the priorities of the tasks. It does not take into account any information about the past states, previous actions, or the history of task execution.

An example of finding the best scheduler for a Markov Decision Process could be an industrial manufacturing plant. The goal is to minimise the cost of production while maintaining high quality standards. Markov Decision Process can be used to model the different production processes, the costs associated with each process, the quality control measures, and the probability of success for each process. Using this model, one can try to find the schedulers that determine the optimal combination of processes to minimise the cost of production while ensuring that the quality standards are met. The best scheduler will be the one that results in the lowest cost of production while maintaining high quality standards.

In the coffee machine example, a scheduler could be used to optimise the process of brewing and serving coffee. The goal of the scheduler would be to find the best sequence of actions to take, given the current state of the coffee machine and its environment, in order to serve the maximum number of customers in the shortest amount of time. For example, the scheduler might choose to prioritise cleaning and restocking the machine over brewing a new pot of coffee if the machine is running low on supplies. This decision would be based on the current state of the machine, such as the level of beans, water, and cups available. To solve this problem, a statistical model checking algorithm could for example iteratively evaluate different schedules in order to find the best one. This would involve calculating the expected reward for each possible action, given the current state of the machine and the environment, and choosing the action with the highest reward.

2.4 Execution Traces

Execution traces refer to the sequence of states and actions that occur as a system or process is simulated. It represents a single run of the system and captures the behaviour of the system over time. In the context of Markov Decision Processes (MDPs), the execution traces describe the path that the system takes from the initial state to other states. The trace includes the sequence of states visited by the system, the actions taken by the decision maker, and the rewards obtained at each step. The execution traces may be finite or infinite

and are used to evaluate the performance of the system and to identify the optimal policies. The analysis of execution traces is a key component in model checking, as it allows us to validate the correctness and reliability of the system under consideration.

In the coffee machine example, an execution trace would refer to a specific sequence of events that takes place from the initial state of the machine to a final state. For example, one execution trace could be: the customer orders coffee, the machine dispenses a cup, grinds the beans, pours the coffee into the cup, and then the customer receives their coffee. Another execution trace could be: The customer orders coffee, the machine dispenses a cup, grinds the beans, but then the customer cancels the order before the coffee is poured into the cup. These two execution traces represent different sequences of decisions and actions taken by the coffee machine, and they can be used to evaluate the performance and efficiency of the machine.

2.5 Bounded Linear Temporal Logic

Linear Temporal Logic (LTL) [22] is a formal language that can be used to express properties of systems that evolve over time, such as finite state machines, timed automata, and hybrid systems. Bounded Linear Temporal Logic (BLTL) is an extension of Linear Temporal Logic (LTL), a model logic extending propositional logic that allows for reasoning about temporal properties within a bounded time horizon. BLTL is designed to handle systems with finite or bounded behaviours, where the analysis focuses on a specific time interval rather than an infinite sequence of states.

The main difference between BLTL and LTL lies in their expressiveness and the scope of temporal reasoning. LTL is concerned with properties that hold over an infinite sequence of states or events. It allows reasoning about properties like "always" (G), "eventually" (F), "next" (X), and "until" (U) that can hold at any point in time. LTL is suitable for systems with unbounded behaviours or when the analysis needs to consider properties that hold indefinitely.

On the other hand, BLTL restricts the analysis to a finite time horizon. It introduces bounded temporal operators to specify properties that hold within a specific time interval. For example, the "always" property becomes (G_t) where "t" gives an upper bound of the time interval. By incorporating this bounded temporal operator, BLTL allows for more precise reasoning about temporal properties within a defined time frame. It is particularly useful when the focus is on finite-time behaviours or when the time horizon of interest is explicitly bounded.

The syntax of Bounded Linear Temporal Logic is defined as:

- Propositions: propositions are atomic statements that can be either true or false, often representing a state. They are typically represented by lowercase letters or predicates.
- Logical operators: BLTL includes logical operators such as "and" ($\wedge$), "or" ($\vee$), "not" ($\neg$), and "implies" ($\rightarrow$) to combine propositions.
- Temporal operators: BLTL includes temporal operators such as "next within t" (X_t), "until within t" (U_t, "always within t" (G_t), and "eventually within t" (F_t), where t is a time bound. These operators specify properties that hold within a certain time bound.
- Parentheses: Parentheses are used to indicate the order of operations and to group propositions together

More formally, the syntax of Bounded Linear Temporal Logic can be defined as:

```
Property ::= ( ('declare' Variable (';' Variable)*)?
               ('optimize' Variable (';' Variable)*)?
               'end')? Formula
Formula ::= 'F <=' Bound Formula
    | 'G <=' Bound Formula
    | Formula 'U <=' Bound Formula
    | Formula 'W <=' Bound Formula
    | 'X' ('<=' Bound)? Formula
    | Formula '&' Formula
    | Formula '|' Formula
    | Formula '=>' Formula
    | '!' Formula
    | '(' Formula ')'
    | 'true'
    | 'false'
    | Numerical Operator Numerical
Bound ::= ('#')? Numerical
Variable ::= ident ':=' (Interval | number)
Interval ::= '['number';'number';'number']'
Numerical ::= ident | number
Operator ::=  '<' | '<' | '!=' | '=' | '>=' | '>' | '+' | '-' | '*' | '/'
```

Figure 4: BLTL syntax

The syntax refers to the set of rules and structure that govern the formation and arrangement of elements. It defines the rules for constructing valid expressions and specifies how different elements such as keywords, variables, operators, and punctuation marks should be combined and arranged to form meaningful and well-formed constructs.

On the other hand, the semantics of Bounded Linear Temporal Logic (BLTL) defines the meaning of the formulas in the language. It defines how the formulas relate to the behaviour of a system over time. it is based on Linear Temporal Logic (LTL) and it is defined over infinite-length sequences of states, which, in the context of model checking, corresponds to possible executions of the system (traces). . Each state in a trace represents the status of the system at a particular point in time. The semantics of BLTL is defined as follows:

- A BLTL formula is satisfied by a trace if and only if the formula holds at the first step of the trace.
- the semantic interpretation of logical operators (and (&), or (|), not (!), ...) is similar to other logic systems like LTL
- The temporal operator "next within t" (X_t) holds if the property holds in the next state within the bound t.
- The temporal operator "until within t" (U_t) holds if the property holds in all states up to and including the bound t, or until the second part of the formula holds.
- The temporal operator "always" (G_t) holds for a trace if the property holds for all future states, within the bound t.
- The temporal operator "eventually" (F_t) holds for a trace if the property holds for at least one future state, within the bound t.

Here's an example of a Basic Linear Temporal Logic (BLTL) formula:

$$\varphi = G_{t_1}((p \rightarrow F_{t_2}(q)) \wedge (r \rightarrow F_{t_3}(p \wedge q)))$$

In this formula, p, q, and r are propositional variables representing different conditions or events in a system. Let's break down the components of the formula: The "G" operator stands for "Globally" and is used to specify that a condition holds true in all future states. The subformula $(p \rightarrow F_{t_2}(q))$ states that if p is true in the current state, then eventually q will become true in some future state. The subformula $(r \rightarrow F_{t_3}(p \wedge q))$ states that if r is true in the current state, then eventually both p and q will become true in some future state. Combining these subformulas with the conjunction operator ($\wedge$), we obtain the complete BLTL formula φ. The formula φ states that in all future states, if p is true, then q will eventually become true. Additionally, if r is true, then both p and q will eventually become true.

2.6 Plasma Lab

The Plasma Lab software [23, 24, 25] is a tool created by Inria, a French research institute, to analyse and verify concurrent and reactive systems. It

uses model checking, abstract interpretation and theorem proving to check the safety, deadlock freedom, and performance of software systems. The software is used in various fields such as robotics, telecommunication, and industrial automation, and allows engineers to validate their systems and improve their performance, reliability, and safety.

The element of the Plasma Lab software we will use in this master thesis is a formal verification algorithm. It is used for proving the correctness and performance of algorithms and systems including the use of mathematical models and formal verification techniques. Plasma Lab architecture is based on plugins. Some are responsible of the model and perform the simulation while others manage the requirement and verify them on previously produced traces. There also exist a third king of plugin that manages the entire process and aggregates the results statistically. All these types of plugins can be integrated to Plasma by implementing a Java interface.

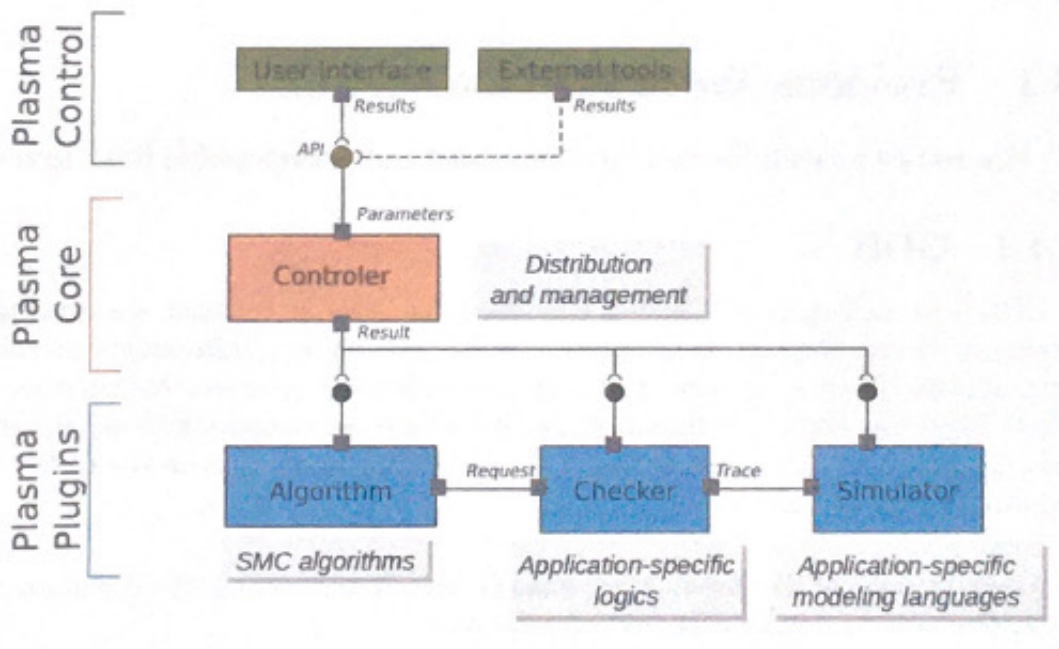

Figure 5: Plasma Lab architecture taken from [25]

There exist multiple other model checking tools such as PRISM [26] but we decided to work with Plasma Lab because the algorithm we have worked on (see Smart Sampling algorithm section) was already implemented using this tool.

2.6.1 Compile Plasma

The purpose of this subsection is to explain how to compile Plasma Lab using IntelliJ IDEA.

To achieve a compilation of Plasma Lab, we first need to open the plasma code into a new IntelliJ Project. Then, we must load the project in order to import all modules and dependencies needed using the maven integration tool available in IntelliJ. To do so, go to fr.inria.plasmalab.root → pom.xml. This pom file (Project Object Model) is an xml file that contains information about the project and configuration details used by Maven to build the project. Right click on that file and select maven → load project. You should now have a working project with all modules installed.

At this point, some plugins required to be installed manually like the bltl, the nested and the bio plugin. To achieve it, we need the readConf method of the Controller class (plasma-lab → fr.inria.plasmalab.root → fr.inria.plasmalab.controller → src →java → fr → inria → plasmalab → controller) which use a configuration file to import the missing package. If this configuration file does not exist in your project, create it at the root of the project with the name plasmalab.conf and complete it like following :

```
# PLASMA Lab default configuration file
# Version 1.4.2

# To use this file use the --conf option:
# e.g. ./plasmagui.sh launch --conf RELATIVE_PATH_TO_CONF
# or   plasmagui.bat --conf RELATIVE_PATH_TO_CONF

# Plugins directory
plugin_dir plugins/

#plugin file:///absolute_path_to_plugin/plugin.jar
#plugin http://url_to_plugin/plugin.jar

# Model directory
working_dir demos
```

Figure 6: Plasma Lab configuration script

You will also need to copy the .jar file of the missing plugins from the Plasma Lab website into a new folder named plugins. This folder must be created at the root of the project. The project is now complete and ready to run.

To compile the graphical user interface of the newly created project, you must execute the main method of the PlasmaLabTerminalUI class which is located at
fr.inria.plasmalab.uiterminal.PlasmaLabTerminalUI. Don't forget to use the "launch –conf plasmalab.conf" cli argument otherwise the compilation won't

work properly.

3 Smart Sampling Experimentation

The Monte Carlo Statistical Model Checking (MC-SMC) algorithm is a basic method used to check and verify properties of probabilistic models. It combines the principles of Monte Carlo simulation and statistical analysis to approximate the behaviour of these models.

To use the MC-SMC algorithm, we first need to define the probabilistic model that represents the system we are interested in. This model consists of different states and the probabilities of transitioning between these states. We also specify the properties that we want to check, such as safety or liveness properties. The MC-SMC algorithm then generates a large number of simulations or runs of the system. These simulations randomly explore the possible paths through the model. During each simulation, we collect data and observations about the system's behaviour. Once the simulations are completed, we perform statistical analysis on the collected data. This analysis helps estimate the probabilities or statistical properties we are interested in. Techniques like hypothesis testing or confidence interval estimation are used to analyse the data. Next, we compare the estimated probabilities or statistical properties with the thresholds or requirements specified for the properties. This comparison tells whether the properties hold or if they violate the given criteria.

The MC-SMC algorithm can be iterative, allowing for the refinement of the analysis. By increasing the number of simulations or adjusting the statistical analysis techniques, you can improve the accuracy of the property estimation. It's important to note that the MC-SMC algorithm provides approximate results rather than exhaustive guarantees. The accuracy of the results depends on factors such as the number of simulations and the statistical analysis techniques used. However, MC-SMC offers a practical approach for verifying probabilistic properties when analysing the entire execution traces space of a system is infeasible

3.1 Smart Sampling algorithm

Like the Monte Carlo algorithm, the Smart Sampling algorithm [18] is a statistical model checking algorithm. His purpose is to provide a more efficient and lightweight approach to verify properties in Markov Decision Processes (MDPs). The algorithm aims to reduce the computational complexity and resource requirements associated while still providing accurate results. The main motivation behind the algorithm is to address the scalability challenges faced when verifying large-scale MDP models (state explosion problem). Traditional

verification methods such as the Monte Carlo algorithm often require exhaustive or random exploration of possible schedulers, which becomes infeasible for complex MDPs with a large number of states and transitions. The smart sampling algorithm addresses this challenge by selectively sampling relevant schedulers, thereby reducing the verification effort without sacrificing the accuracy of the results. A scheduler is considered relevant if its probability to satisfy a given property is higher compared to other schedulers probabilities. The probability is computed using a statistical model checking approach.

Overall, the purpose of the smart sampling algorithm is to provide a practical and scalable approach for verifying properties in MDPs, making the verification process more accessible and feasible for large-scale systems.

The Smart Sampling algorithm takes as input a Markov Decision Process, a bounded property ϕ and an iteration budget N_{max}, while it return the estimated probability using the best scheduler candidate $\hat{p}_{\overline{max}}$. This estimation can be used as a real value when the true probability using the best current scheduler ($p_{\overline{max}}$) is lower than the truth notional maximum probability (p_{max}) and the probability that the difference between the true probability using the best current scheduler ($p_{\overline{max}}$) and the estimated probability ($\hat{p}_{\overline{max}}$) is greater than the error (ϵ) must be lover than the confidence interval (δ) :$P(|p_{\overline{max}} - \hat{p}_{\overline{max}}| \geq \epsilon) \leq \delta$

Input:
$\mathcal{M}$: an MDP
φ: a property
ϵ, δ: the required Chernoff bound
$N_{\max} > \ln(2/\delta)/(2\epsilon^2)$: the per-iteration budget
Output: $\hat{p}_{\overline{\max}} \approx p_{\max}$, where $p_{\overline{\max}} \approx p_{\max}$ and
$\mathrm{P}(|p_{\overline{\max}} - \hat{p}_{\overline{\max}}| \geq \epsilon) \leq \delta$

Figure 7: SSA input output

The algorithm is separated in three stages: an initial random sampling simulation is first performed to discover the nature of the problem and to generate a first set of schedulers. This section is implemented from lines 1 to 5 in the algorithm. The N parameter is the number of simulations for each scheduler while M is the number of schedulers. For now, these two parameters are set to be equal and their product must be the per-iteration budget. In these first 5 lines, we first generate a set of scheduler, the selection is random and the number of schedulers chosen is equal to M. These schedulers are then simulated and we keep track of execution traces satisfying the property ϕ by

mapping into an array called R. Lastly, we compute the estimated maximum probability $\hat{p}_{\overline{max}}$ which corresponds to the maximum ratio of traces that verify the property among the schedulers for which traces have been generated.

1 $N \leftarrow \lceil \sqrt{N_{\max}} \rceil$; $M \leftarrow \lceil \sqrt{N_{\max}} \rceil$
2 $S \leftarrow \{M$ seeds chosen uniformly at random$\}$
3 $\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
4 $R : S \to \mathbb{N}$ maps scheduler seeds to number of traces satisfying φ:
$R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
5 $\hat{p}_{\overline{\max}} \leftarrow \max_{\sigma \in S}(R(\sigma)/N)$

Figure 8: SSA Lines 1 to 5

The second stage attempts to resample the schedulers set based on the estimated probabilities of the initial set. The new set of scheduler is generated by setting the simulation budget $N = 1/\hat{p}_{\overline{max}}$ and the number of schedulers $M = N_{max}\hat{p}_{\overline{max}}$. Note that these numbers depend on the estimated probability computed before. We then recreate and re-simulate a set of schedulers chosen randomly like in the first part of this algorithm in order to obtain a second set of schedulers. Then, we drop the schedulers that does not satisfy the property for at least one execution trace and keep all other for the following of the algorithm.

6 $N \leftarrow \lfloor 1/\hat{p}_{\overline{\max}} \rfloor$, $M \leftarrow \lfloor N_{\max}\, \hat{p}_{\overline{\max}} \rfloor$
7 $S \leftarrow \{M$ seeds chosen uniformly at random$\}$
8 $\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
9 $R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
10 $S \leftarrow \{\sigma \in S \mid R(\sigma) > 0\}$

Figure 9: SSA Lines 6 to 10

The purpose of the first two parts of the Smart Sampling algorithm is to generate a first sample of schedulers while the third part which is the main section of this Smart Sampling algorithm consists of an iterative refinement of the candidates schedulers based on their merit. This means that the schedulers with the worst estimated probabilities are excluded and their simulation budget is reallocated to the best schedulers. This refinement process reduces the number of simulated schedulers while increasing the confidence of their estimations. This process runs while the confidence interval of the estimated probability is greater than δ and while they are still schedulers in the set. At

each iteration, the set of scheduler is simulated and evaluated like in sections 1 and 2 of the algorithm. Then, the algorithm estimated the probabilities similarly to the first initial computation. Finally, we refine the set of schedulers by keeping the best half of them. These are the schedulers with the highest number of traces satisfying the property.

11 $\forall \sigma \in S,\ R(\sigma) \leftarrow 0;\ i \leftarrow 0;\ conf \leftarrow 1$
12 **while** $conf > \delta \wedge S \neq \emptyset$ **do**
13 $\quad i \leftarrow i + 1$
14 $\quad M_i \leftarrow |S|$
15 $\quad N_i \leftarrow 0$
16 $\quad$ **while** $conf > \delta \wedge N_i < \lceil N_{\max}/M_i \rceil$ **do**
17 $\quad\quad N_i \leftarrow N_i + 1$
18 $\quad\quad conf \leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$
19 $\quad\quad \forall \sigma \in S : \omega^{\sigma}_{N_i} \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
20 $\quad R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{j=1}^{N_i} \mathbf{1}(\omega^{\sigma}_j \models \varphi)\}$
21 $\quad \hat{p}_{\overline{\max}} \leftarrow \max_{\sigma \in S}(R(\sigma)/N_i)$
22 $\quad R' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
$\quad\quad \forall (n, \sigma), (n', \sigma') \in R',\ n > n' \implies R(\sigma) \geq R(\sigma')$
23 $\quad S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$

Figure 10: SSA lines 11 to 23

The algorithm stops if the scheduler set is empty or if the confidence interval is smaller the the confidence bound δ

3.1.1 Parameter Variation

The goal of the first part of this master thesis is to modify parameters of the smart sampling algorithm and to analyse the results of these variations in order to find an optimal version of the algorithm. Such optimised algorithm would ideally find better maximum or minimum probabilities in less time than the original algorithm. The goal of these variations is also to better understand the original algorithm and the parts that play a role in the optimisation process.

As we saw before, the smart sampling algorithm is not a long or complex algorithm, in fact, the pseudo-code given in section 3.1 is only 23 lines long, but there are still many parameters to consider for variation. One of them is the size of simulation budget per scheduler defined at line 6 of the Smart Sampling

algorithm. This parameter represents the number of different traces that can be evaluated for a single scheduler, thus the higher the value of the parameter, the more likely it is to find execution traces that verify the property. On the other hand, the higher the value of the parameter, the more time is spent on each scheduler which may prove to be useless depending on the merit of the scheduler to find execution traces that satisfy the property.

Another parameter considered modifying is the total number of schedulers defined at line 6. This parameter describes the number of scheduler in the initial set. Modifying this parameter has an impact on the Smart Sampling algorithms. Either we have more initial schedulers which may increase the chances to find one scheduler that produces good traces to satisfy the property, but the algorithm risk to spend more time evaluating bad schedulers, or we have fewer schedulers which allows the algorithm to spend more time on chosen schedulers but we may miss out on some potential good schedulers dropped from the initial set. The stated goal of varying this parameter is to evaluate the importance of the initial set of scheduler.

In this master thesis, we focused on modifying the third parameter which is the number of schedulers dropped after each iteration of the Smart Sampling algorithm called reduction factor (line 23). This parameter is critical in the third section of the algorithm (the loop section) because it resamples the schedulers set. The initial value of 2 keeps half of the schedulers after each iteration (keeping obviously the best half). We modified this parameter from small numbers (1.2, 1.5) that keep more schedulers after each iteration to large numbers (20, 50) that drops lots of schedulers. The goal of varying the reduction factor is to evaluate the impact on the algorithm performances of a more or less aggressive refinement of the schedulers set. Varying this parameter should also bring a response to the question: Is it better to keep fewer schedulers but allow more iterations to each of them or is it better to have more scheduler with less iteration budget ?

3.1.2 Expected Results

The objective of modifying the parameter explained in the previous section is to evaluate the impact of this change on the smart sampling algorithm and to see how the estimated computed probabilities will be altered. A time variation directly correlated with the parameter is expected as the higher value of the reduction factor, the lower time needed to obtain a result. It is because the algorithm stops when there is only one scheduler remaining so dropping more schedulers at each iteration brings the algorithm faster to its termination. An estimated probability variation is also expected because as explained in the previous section, dropping too many schedulers could penalise the objective to find one that satisfies the property. The final objective of this parameter

variation is to find the best value that maximises the estimated probability while minimising the time needed to compute it.

3.2 Case Studies

To evaluate the impact of the parameter variation presented above, we have performed a number of experiments on standard model taken from plasma case studies. These well-known models come from the numerical model checking literature.

3.2.1 Wireless LAN Protocol

Wireless LAN (Local Area Network) protocol [27] is a communication standard for connecting devices wirelessly in a local network. It provides the specifications and rules for data transmission over short distances, such as in a home, office, or campus environment. The goal of Wireless LAN (Local Area Network) is to provide a communication network that enables devices to connect and exchange data wirelessly. The main objectives of Wireless LAN include:

- Data transmission: To provide a means of transmitting data between devices in a local network, such as in a home, office, or campus environment.
- Connectivity: to provide a way for devices to connect to the network and access shared resources, such as the internet, printers, and servers.
- Mobility: to allow users to move around freely while maintaining a connection to the network, providing a seamless user experience.
- Ease of use: to make it easy to set up and use wireless networks, reducing the need for technical expertise.
- Scalability: to provide a solution that can easily be expanded or reconfigured as the needs of the network change over time.
- Security: to ensure that data transmitted over the network is protected from unauthorised access and theft.

Overall, the goal of Wireless LAN is to provide a reliable, flexible, and secure means of communication for devices in a local network. It is designed to meet the growing demand for wireless connectivity and to provide a cost-effective alternative to traditional wired networks.

Wireless LAN comes with collisions that occur when two or more devices attempt to transmit data at the same time, resulting in interference that causes the data to be corrupted or lost. This can lead to decreased performance and reduced overall network efficiency. To mitigate the effects of collisions, Wireless LAN protocols use a number of techniques, including carrier sensing, collision

avoidance, collision detection or priority-based scheduling. The goal of these techniques is to minimise the number of collisions and reduce their impact on the network in order to provide reliable and efficient data transmission even in busy environments.

We will use the Smart Sampling algorithm to analyse how tho protocol deal with the collisions. The parameters used for the are $\delta = \epsilon = 0.01$, the simulation budget = 10^5. The per-iteration budget is set to 10^5. To perform the experiment, we use the BLTL property $F \leq \#k\ col = COL$ where $k, the number of maximum allowed collisions \in \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ and the reduction factors
$\in \{1.2, 1.5, 2, 3, 5, 10, 20, 50\}$. For each couple of (k, reduction factor), we computed the maximum expected probability with the duration of execution (in seconds) and the minimum expected probability also with the execution duration.

3.2.2 Results

This section present the result of the wireless LAN protocol experimentation. This analyse of the results will be done in the following section 3.3. The table below show the result of the experimentation. We can see that the minimum probability is always 0 thus we will only analyse the result for the maximum probabilities (the gray lines in the table)

	k	0	10	20	30	40	50	60	70	80	90	100
reduction factor												
1,2	Max Proba	0	0	0	0	0,03936	0,07868	0,12084	0,15996	0,18568	0,185	0,18512
...	Time (s)	1,185	55,56	119,238	136,78	800,101	1105,72	1395,147	1741,9	1818,147	2237,7	2317,408
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,294	13,487	22,47	42,182	986,141	1388,22	66,951	3086,618	2272,56	378,715	1561,571
1,5	Max Proba	0	0	0	0	0,04108	0,08016	0,11936	0,15905	0,18164	0,1826	0,18724
...	Time (s)	1,47	57,928	121,973	142,634	508,388	562,293	731,912	791,492	841,675	1022,21	1701,941
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,238	9,265	18,184	24,548	272,814	80,265	583,76	95,574	768,8	869,6	144,88
2	Max Proba	0	0	0	0	0,03872	0,0736	0,11616	0,15744	0,184387	0,187147	0,183367
...	Time (s)	0,803	44,834	84,241	122,651	247,792	326,458	380,318	567,815	591,119	654,874	741,266
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,313	14,304	24,338	56,406	465,74	563,947	1013,83	813,789	137,961	891,329	1861,139
3	Max Proba	0	0	0	0	0,04038	0,07689	0,1189	0,16047	0,18837	0,18522	0,18965
...	Time (s)	0,986	48,852	85,557	124,733	171,22	241,26	280,212	309,716	338,922	362,575	494,816
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,273	12,346	23,708	33,621	258,956	448,855	193,629	558,18	335,7	566,653	777,293
5	Max Proba	0	0	0	0	0,0396	0,07764	0,1208	0,15793	0,18648	0,1864	0,182707
...	Time (s)	0,736	43,523	81,683	124,255	118,884	147,429	222,668	288,706	287,984	288,999	326,687
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,206	9,338	17,658	24,935	83,256	148,698	240,328	374,275	509,766	259,47	380,958
10	Max Proba	0	0	0	0	0,0363	0,0766	0,12348	0,15568	0,1956	0,18575	0,193
...	Time (s)	0,821	46,65	81,682	123,102	102,202	133,978	146,647	164,582	186,169	203,556	217,344
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,453	18,936	37,397	27,456	136,832	225,662	241,702	317,78	92,147	204,2	213,809
20	Max Proba	0	0	0	0	0,04169	0,0752	0,1218	0,15949	0,19665	0,1954	0,19789
...	Time (s)	1,491	63,293	119,211	166,887	121,161	143,395	145,416	192,69	168,68	239,402	308,931
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,149	9,465	15,959	24,806	70,407	131,32	235,289	186,537	252,5	182,77	174,153
50	Max Proba	0	0	0	0	0,0437	0,09035	0,1151	0,15296	0,1843	0,18464	0,1823
...	Time (s)	1,208	64,625	93,652	179,176	86,263	112,221	217,92	169,863	178,127	220,735	367,198
...	Min proba	0	0	0	0	0	0	0	0	0	0	0
...	Time (s)	0,16	8,283	15,696	25,464	74,5	95,213	109,825	142,474	117,863	202,11	119,629

Figure 11: Wireless LAN protocol result table

The graphic in figure 12 shows the maximum probabilities depending on the parameter k for every reduction factor tested. The purpose of this graph is obviously to show that the maximum probabilities varies with the parameter k (maximum number of collisions allowed) and that the result are similar to the simulation realised in [18]. Another parameter to observe on this graph is the changes of probability depending on the reduction factor variation.

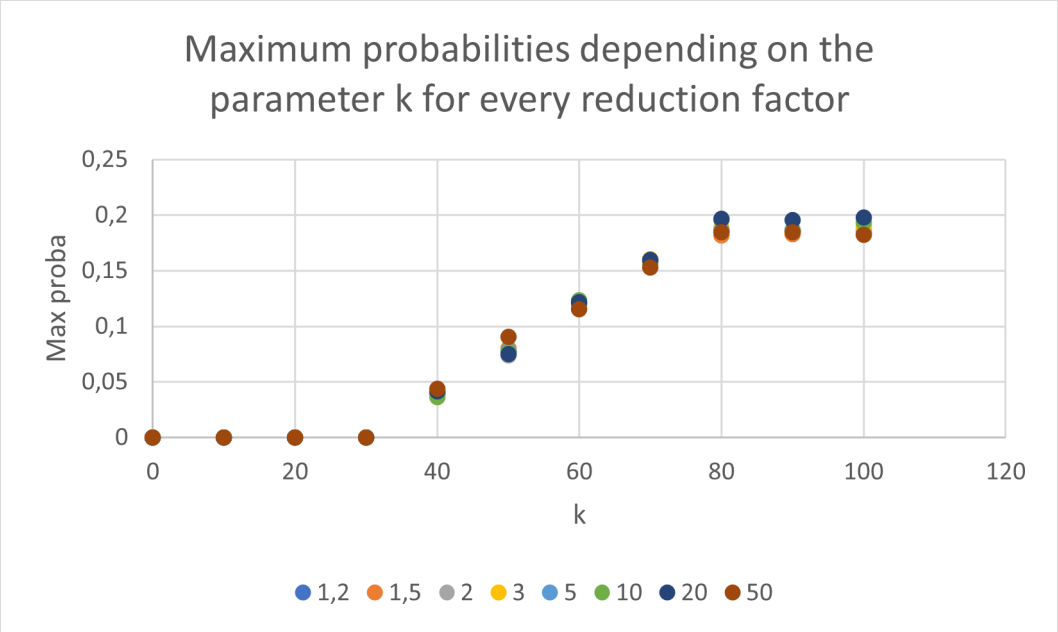

Figure 12: Maximum probabilities depending on the parameter k for every reduction factor

Other interesting results presented by the two graphics below show the time needed to compute the maximum probability depending on the reduction factors and the value of the parameter k, for every value of the parameter k/reduction factor respectively. These graphics are useful to determine the time needed to compute the probability but more importantly to identify the relation between this computational time and the reduction factor and parameter k. They show the computation duration impact of different reduction factor and k.

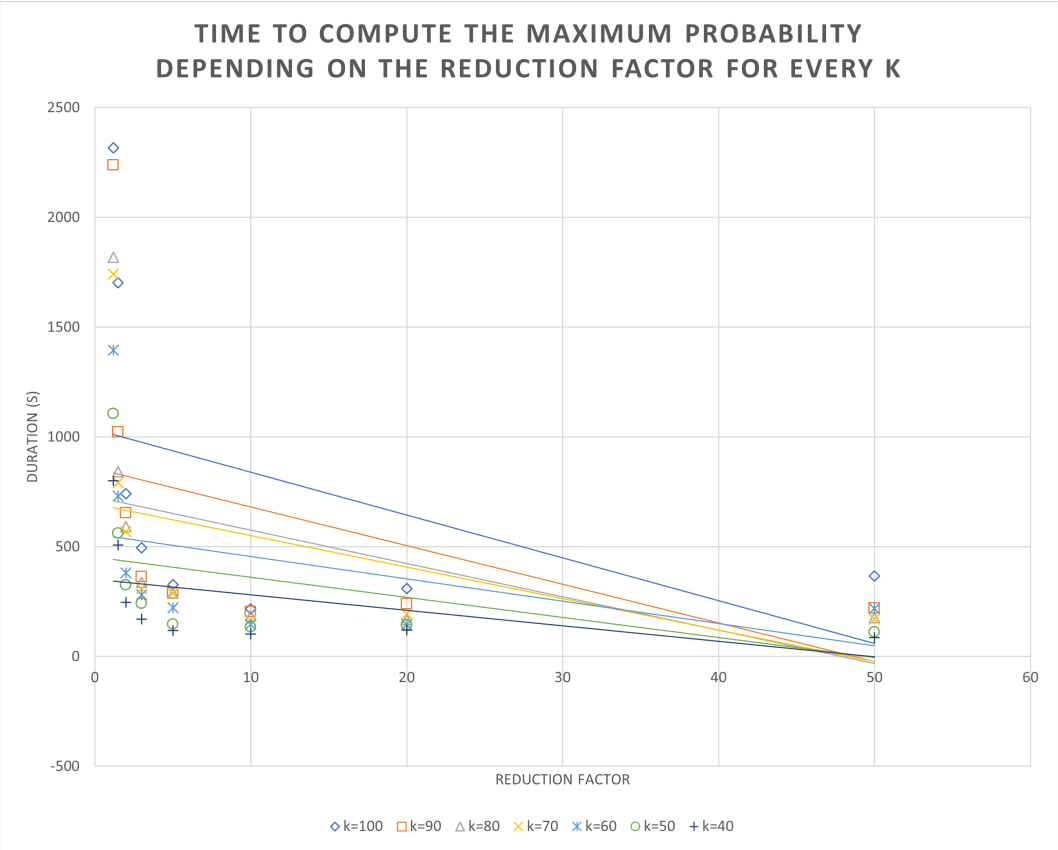

Figure 13: Time to compute the maximum probability depending on the reduction factor for every k

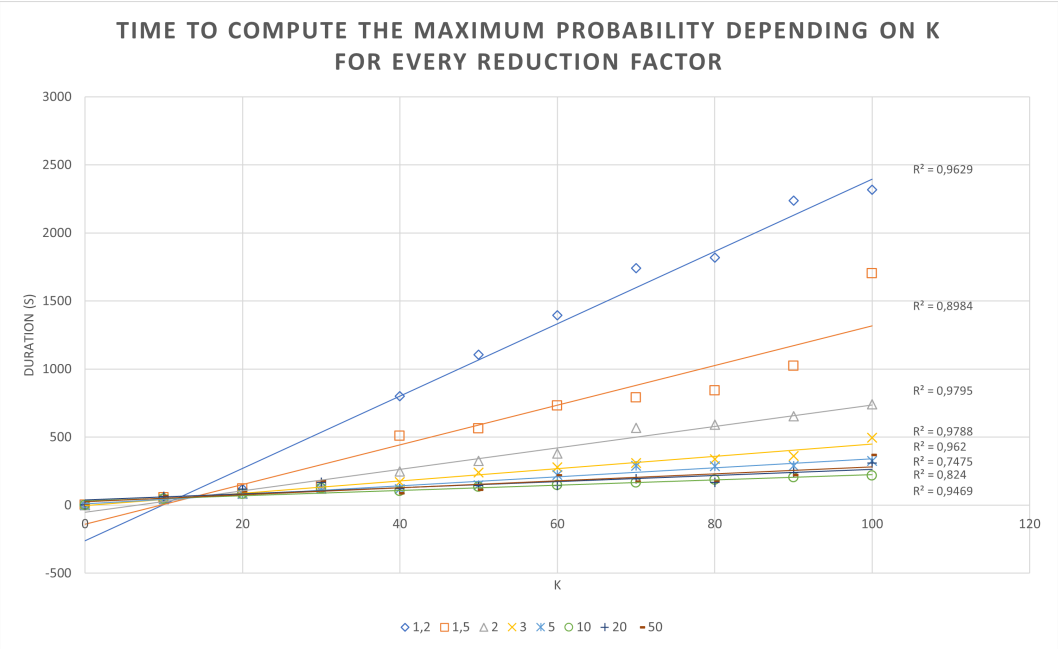

Figure 14: Time to compute the maximum probability depending on k for every reduction factor

The last interesting results extracted from the table in the figure 11 are represented in the graphic below. The graph display the maximum probabilities depending on the duration of the computation. This graphic shows the impacts of reduction factor variations on the time needed to compute the maximum probability. The more a line is vertical, the less time needed to obtain the result.

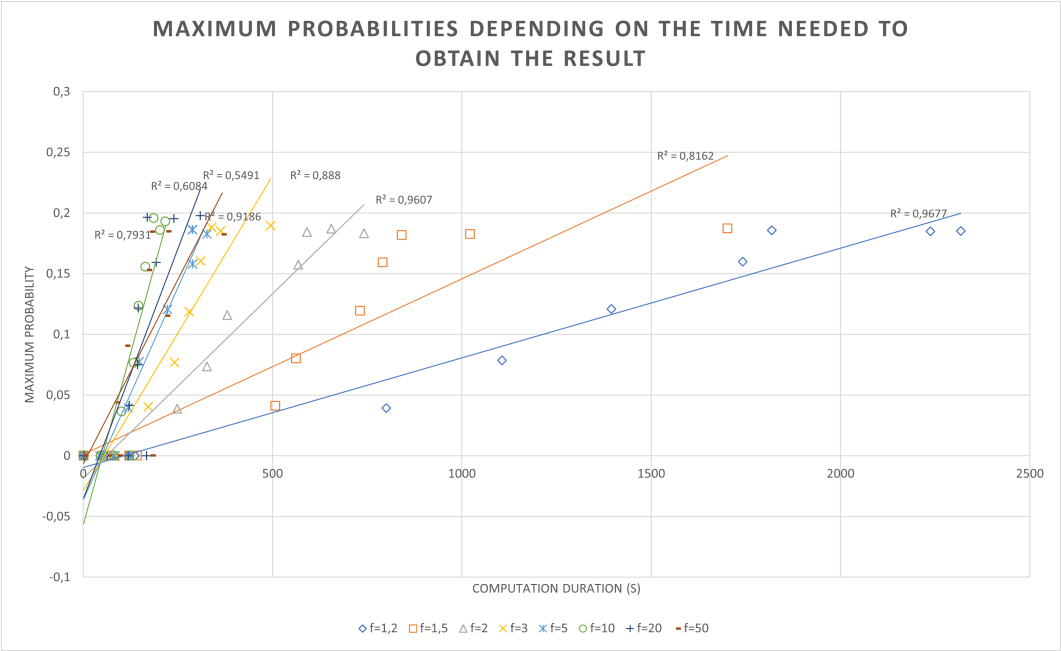

Figure 15: Maximum probabilities depending on the time needed to obtain the result

3.2.3 Gossip Protocol

The gossip protocol [28] is a decentralised communication mechanism used in distributed systems. It involves the exchange of information or updates between nodes in a network without relying on a centralised authority or coordinator. Each node in the network periodically sends a message to a randomly selected neighbour, who then forwards the message to another randomly selected neighbour, and so on. This process continues until all nodes in the network have received the message. The gossip protocol is used to achieve tasks such as broadcasting, data dissemination, network synchronisation, and network topology detection.

The path length in a gossip protocol network refers to the number of hops (intermediate nodes) that a message needs to travel from one node to another node in the network. In general, a shorter path length is desirable in a gossip protocol network because it reduces the overhead of message transmission and increases the efficiency of information dissemination. The path length is influenced by various factors such as network topology, node connectivity, and node behaviour. In a gossip protocol, we would like to minimise the path length by selecting the most appropriate nodes to transmit the message.

We used the Smart Sampling algorithm to estimate the minimum and maximum probabilities that the maximum path length in the network is less than

4. The parameters used for the algorithm are $\delta = \epsilon = 0.01$, the simulation budget = 10^5. The per-iteration budget is set to 10^5. To perform the experiment, we use the BLTL property $F \leq \#T\ max\_path\_length \leq 4$ where $T \in \{0, 20, 30, 50, 80, 100, 120, 150, 180, 200\}$ and reduction factor
$\in \{1.5, 2, 3, 5, 10, 20, 50\}$. The T parameter represents a time range which means that for example we compute the maximum probability that the path length is lower than 4 happens less than T time. For each couple of (T, reduction factor), we compute the maximum expected probability with the duration of execution (in seconds) and the minimum expected probability also with the execution duration.

3.2.4 Results

In this section, we present the result of the Gossip protocol experimentation but as in the result section of the Wireless LAN protocol, we do not analyse them. This analyse is done in the following section 3.3 The table below show the result of the experimentation. We can see that unlike the Wireless LAN experimentation, the minimum probability is not always 0 thus we have each data and graphic twice, one for the maximum probabilities and one for the minimum probabilities. For some values of the parameter T, the minimum and maximum probabilities computed where 0 which explain the empty cases in the table bellow.

	T	**0**	**20**	**30**	**50**	**80**	**100**	**120**	**150**	**180**	**200**
reduction factor											
1,5	Max Proba	0	0	0,50285	0,7492	0,87755	0,94215	0,97095	0,98565	0,9933	0,9966
...	Time	2,471	208,369	899,656	1266,165	1552,264	1839,319	1631,843	1547,502	1832,702	1739,443
...	Min proba				0	0,24844	0,62315	0,8098	0,9049	0,95055	0,97575
...	Time				1451,609	1975,343	2408,693	2188,629	2042,837	2481,249	2290,156
2	Max Proba		0	0,50198	0,7558	0,8794	0,941	0,97011	0,9855	0,993	0,9965
...	Time		200,514	525,663	843,2	1001,4	1087,993	997,91	1296,114	1281,511	1211,311
...	Min proba				0	0,2465	0,617	0,8078	0,90156	0,9507	0,9753
...	Time				786,18	1115,669	1409,677	1305,131	1365,726	1306,91	1430,251
3	Max Proba		0	0,5042	0,7548	0,87405	0,94055	0,97095	0,9865	0,99205	0,9967
...	Time		169,581	423,934	566,507	715,213	765,114	787,381	802,754	790,149	790,936
...	Min proba					0,2465	0,61705	0,807	0,9044	0,9511	0,9757
...	Time					761,386	919,306	966,9	941,193	869,14	969,93
5	Max Proba		0	0,50438	0,7532	0,8798	0,9415	0,9707	0,9871	0,9932	0,9963
...	Time		119,7	315,8	467,9	524,672	554,121	588,367	595,274	528,958	549,6
...	Min proba					0,24224	0,619	0,8094	0,9201	0,95	0,9747
...	Time					488,668	624,87	703,653	651,19	592,12	592,178
10	Max Proba			0,5062	0,7518	0,8779	0,9431	0,9715	0,987	0,9937	0,997
...	Time			181,6	357,1901	407,783	451,1	439,996	448,781	410,634	451,771
...	Min proba					0,2384	0,6173	0,8096	0,9112	0,9501	0,9715
...	Time					317,645	410,6	459,161	581,193	457,604	421,912
20	Max Proba			0,5132	0,7524	0,8818	0,9418	0,9732	0,98612	0,9952	0,9975
...	Time			137,515	274,281	367,11	370,926	383,8	372,013	329,993	326,304
...	Min proba					0,2189	0,6201	0,8019	0,9142	0,9492	0,9738
...	Time					242,93	375,399	448	438,91	365,994	355,18
50	Max Proba			0,5172	0,7498	0,8888	0,9501	0,9756	0,9881	0,9964	0,9976
...	Time			110,317	200,113	267, 869	259,903	250,451	276,146	288,818	262,973
...	Min proba					0,2454	0,6097	0,7936	0,92	0,9512	0,972
...	Time					289,965	290,88	295,186	310,103	312,466	282,915

Figure 16: Gossip protocol result table

The graphics below shows the maximum and minimum probabilities depending on the parameter T for every reduction factor tested. The third graph shows the maximum and minimum probabilities for a reduction factor of 2. As with the first experimentation, the probabilities vary with the parameter F but there are also changes for the computed probability values in function of the reduction factor variation.

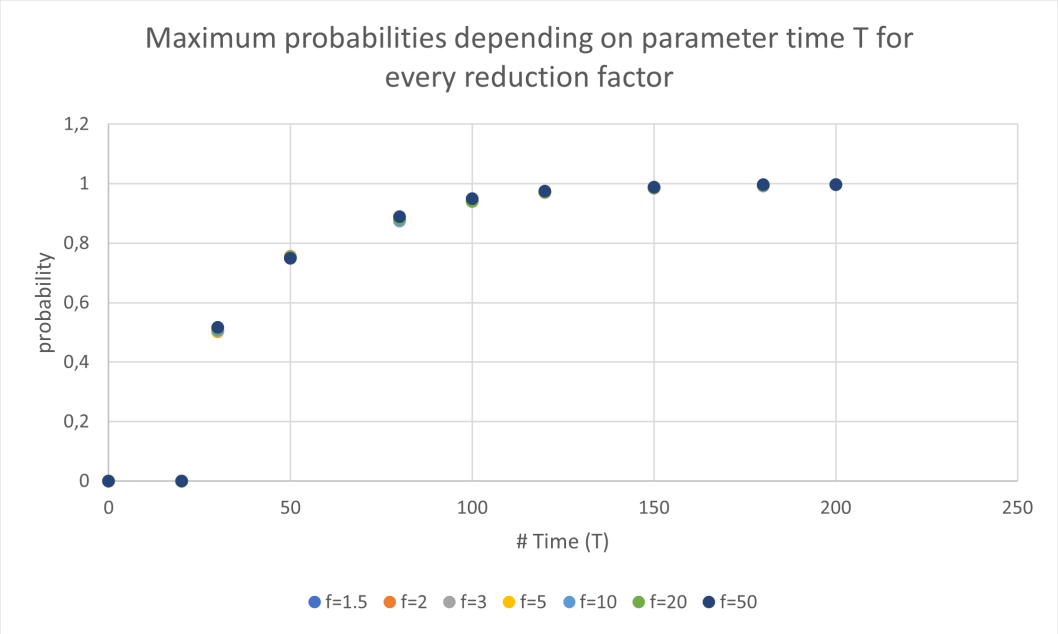

Figure 17: Maximum probabilities depending on the parameter time T for every reduction factor

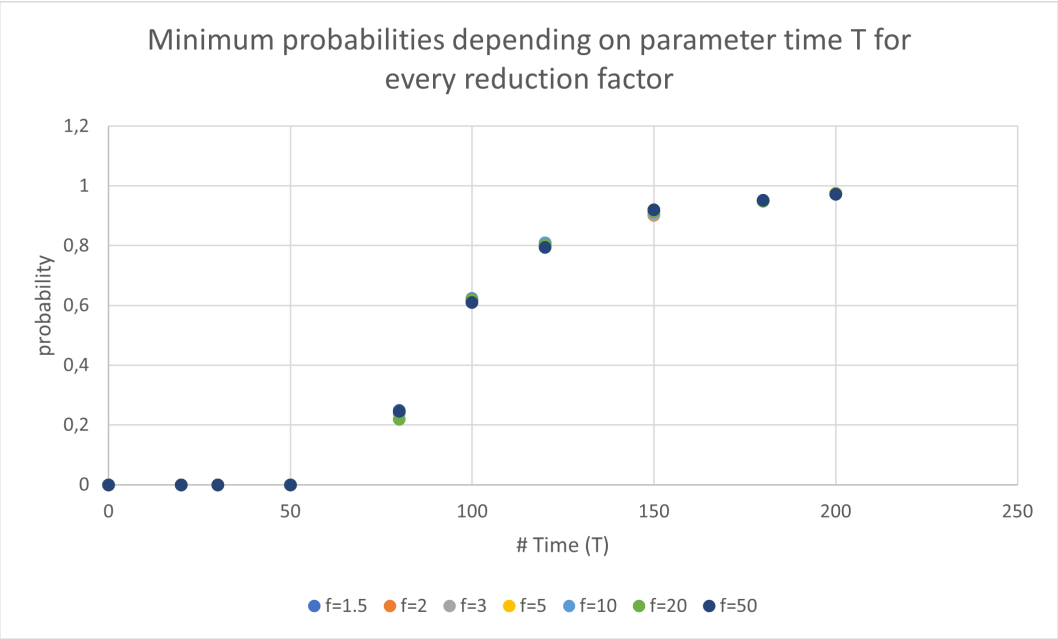

Figure 18: Minimum probabilities depending on the parameter time T for every reduction factor

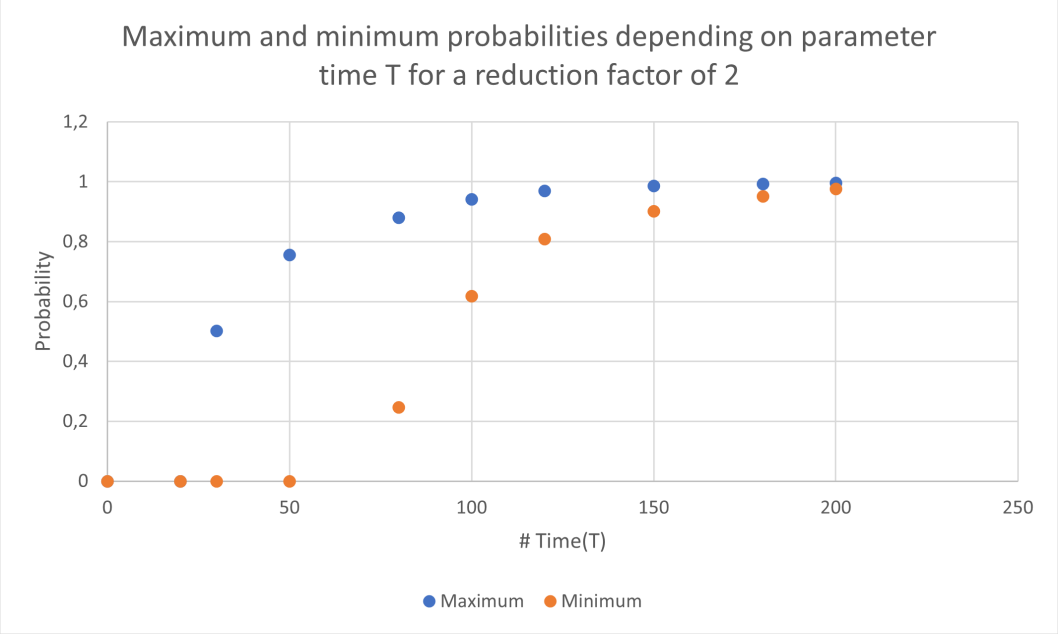

Figure 19: Maximum and minimum probabilities depending on the parameter time T for a reduction factor of 2

The next four graphs are similar to the one presented in the result section of the Wireless LAN protocol experimentation. Note that there are four because

the minimum probabilities are not zero for the Gossip protocol experimentation.

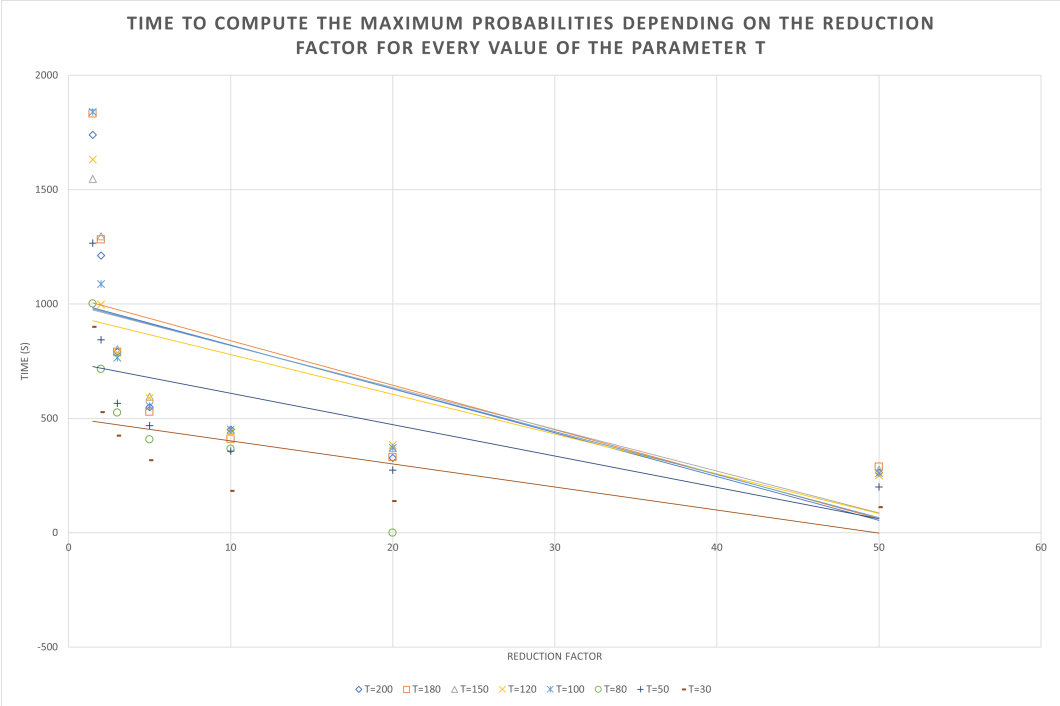

Figure 20: Time to compute the maximum probabilities depending on the reduction factor for every value of the parameter T

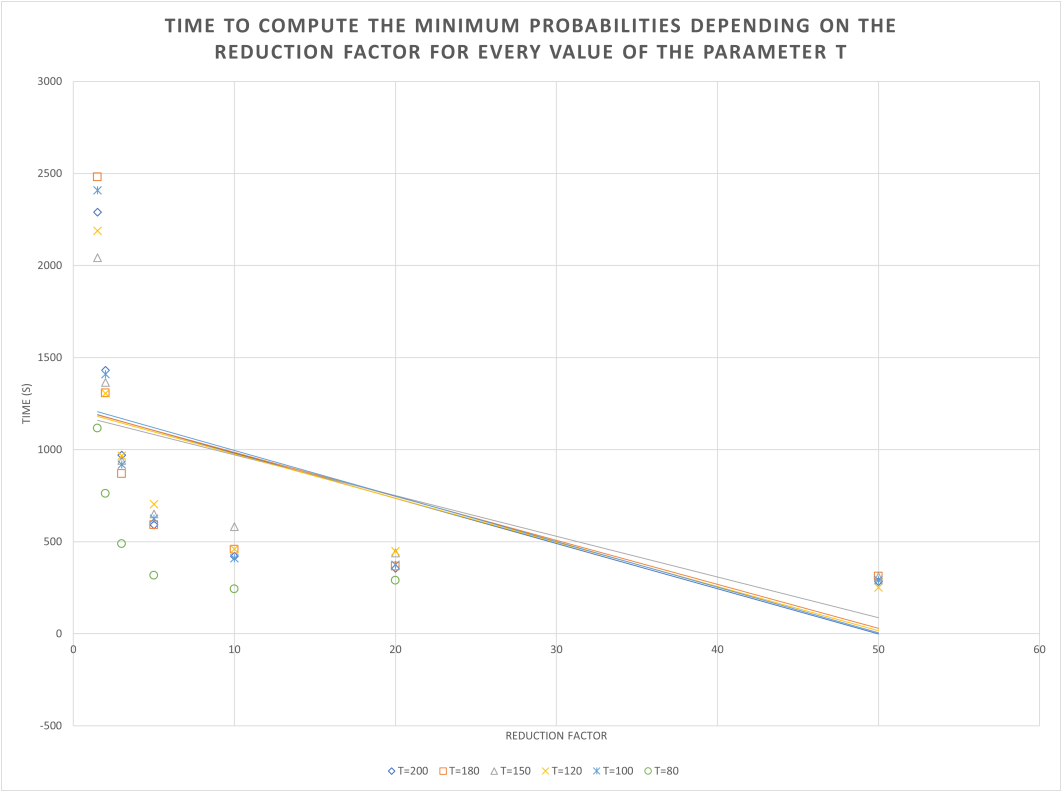

Figure 21: Time to compute the minimum probabilities depending on the reduction factor for every value of the parameter T

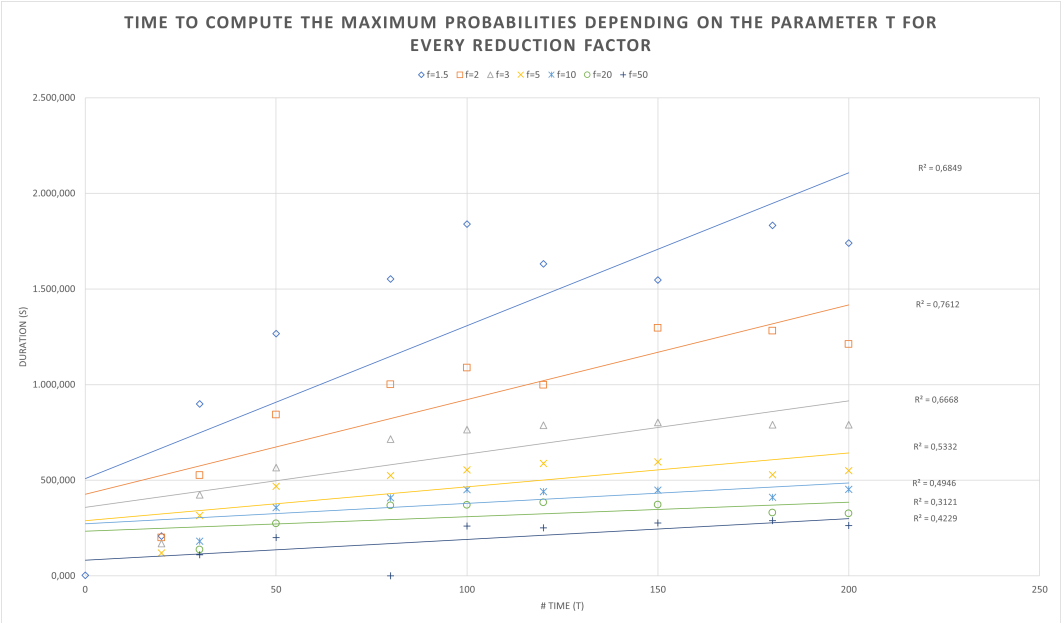

Figure 22: Time to compute the maximum probabilities depending on the parameter T for every reduction factor

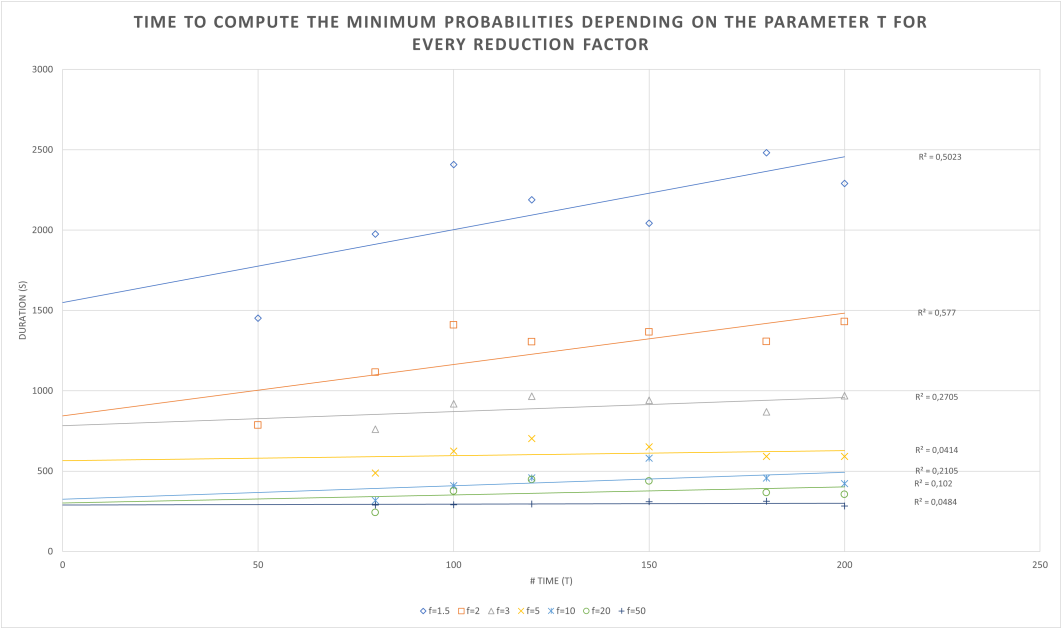

Figure 23: Time to compute the minimum probabilities depending on the parameter T for every reduction factor

These last two graphics are also similar to the last graphic presented in the result of the first experimentation. These graphics show the impact of varying the reduction factor on the time needed to compute the maximum and minimum probabilities.

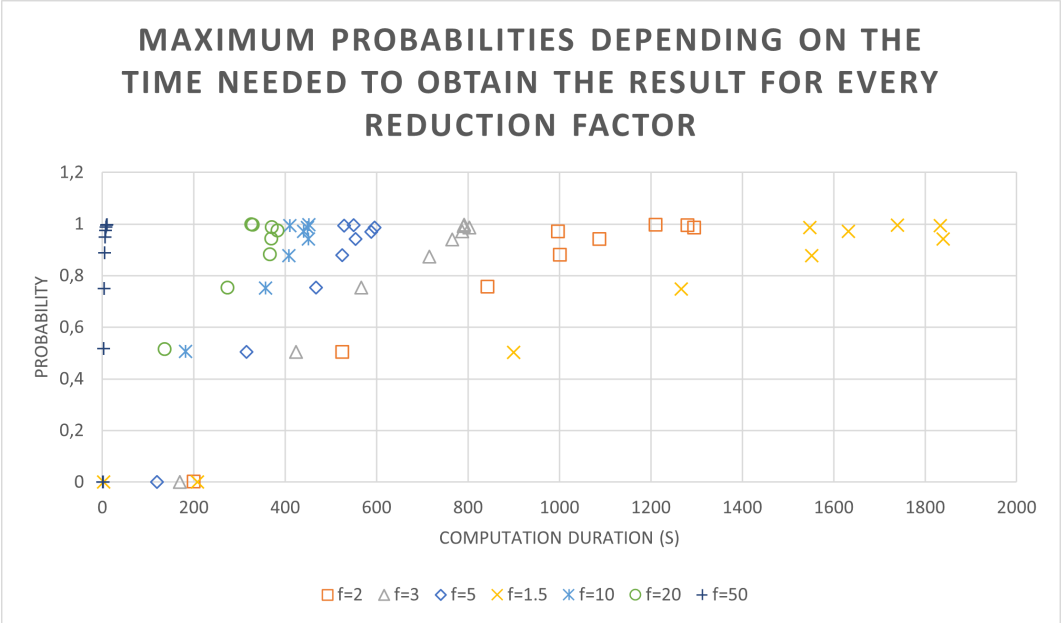

Figure 24: Maximum probabilities depending on the time needed to obtain the result for every reduction factor

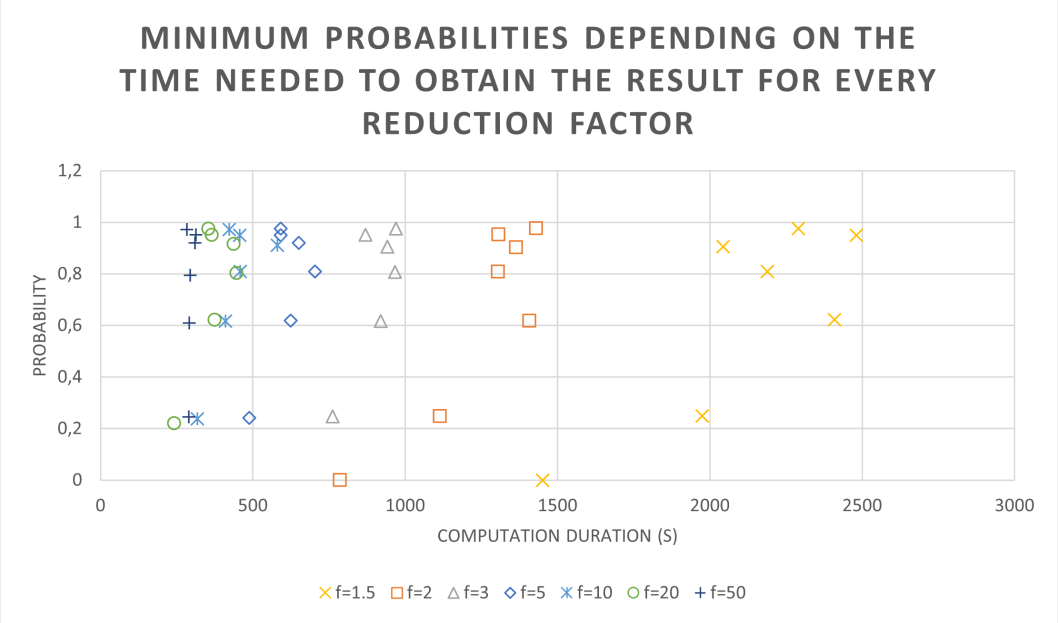

Figure 25: Minimum probabilities depending on the time needed to obtain the result for every reduction factor

3.3 Discussions

In this section, we will discuss the results of the two experimentations presented in the previous sections. As a reminder, we ried to evaluate the impact of modifying the reduction factor parameter of the Smart Sampling algorithms for the computed maximum or minimum probabilities. The case studies for which these modifications were tested are the Wireless LAN protocol and the Gossip protocol and we expected a variation of the computed probabilities but also a variation of the time to compute these probabilities.

While the experiments are different, the result obtained have the same meaning and can be interpreted together, that's why we only do one discussion for both experimentation. The first thing to say is that the reduction factor does not impact the maximum or minimum probabilities computed. We see clearly in the figure 12, 17, 18 and 19 that the estimated probability is the same for every reduction factor tested. This result is quite surprising because we expected to have worse probabilities with higher reduction factor due to the increasingly large deletion of possibly good schedulers at each step of the algorithm. In fact, the algorithm drops only the worse schedulers while keeping the better ones (those who produces the traces with the highest probabilities) and allocating more time to a scheduler to find execution traces of those schedulers that satisfy the property proved to be unnecessary. Thus while no new schedulers are injected into the scheduler set, dropping the worse of them does not help to obtain better results.

The second observation we can make about the result of the two experimentation is that the reduction factor impact the Smart Sampling algorithm's computation duration as demonstrated in figures 13, 14, 22 and 23. We can see on figures 13, 21 and 22 that the computation duration decreases significantly with the reduction factor, thus the higher he is, the faster the algorithm produces a result. This result was expected as explained in the section 3.1.2 because the highest the reduction factor is, the faster the iteration process of the Smart Sampling algorithm will be stopped. We clearly see this trend in figures 15, 24 and 25 because the more the points are on the left, the faster the algorithm produces a result (the probabilities stay the same) and the leftmost points are the ones with the highest reduction factor value.

In conclusion, with those experiments only the computation duration is altered by the reduction factor parameter variation. That observation tells us that the refinement of the scheduler set does not necessarily improve the results of the algorithm. The answer to the previously asked question: is it better to keep fewer schedulers but allow more iterations to each of them or is it better to have more scheduler with less iteration budget is that it doesn't always matter. What matters most is the initial set of schedulers, which is randomly generated in the Smart Sampling algorithm. Some possible ways to improve the algorithm could be to generate a better initial set of schedulers, or to better evaluate the initial set or even to generate new schedulers during the algorithm process to lower the importance of the initial set ...

4 Model improvement

In the last section, we have found that the results of our previously tested models are not entirely satisfying. While we were able to modify the reduction factor parameter, the only impact this had was on the time it takes to compute the minimum and maximum probability values to verify the give property. As a result, we have decided to create a new extreme model that will help us better test the variation of the reduction factor parameter of the smart sampling algorithm. Therefore, the goal of this section is to create new extreme models to better test the smart sampling algorithm and see how they impact the probabilities computed when modifying the reduction factor parameter.

It is important to note that an extreme model does not necessarily mean that it is more complex. The property to verify also plays an important role in identifying extreme cases. For example, an extreme case may involve a very simple model, but with extreme values for the inputs or outputs. In such a case, we would expect to see significant variations in the probabilities computed by the smart sampling algorithm.

To conduct our experiments, we will create an automation tool that will perform a large number of experiments in order to have the most reliable results possible. This tool will allow us to test a wide range of extreme models, each with different input and output values. By doing so, we can ensure that we have a comprehensive understanding of how the smart sampling algorithm performs with extreme models.

By modifying the reduction factor parameter of the Smart Sampling algorithm, we expect to see a time variation to compute the maximum and minimum probabilities to satisfy a given property as for the previous models. However, we also expect to see significant variations in the probabilities themselves. These variations will allow us to identify the strengths and weaknesses of the smart sampling algorithm when dealing with extreme cases.

The results of this section will provide us with a better understanding of the smart sampling algorithm's capabilities and limitations. We will be able to identify the types of extreme models that are the most challenging for the algorithm and develop strategies to address these challenges. This will help us to improve the overall performance and accuracy of the algorithm.

In conclusion, this section is essential to the overall success of the smart sampling algorithm. By creating and testing extreme models, we can identify the algorithm's limitations and develop strategies to overcome them. The automation tool we will develop will allow us to perform a large number of experiments and ensure that we have reliable results. Ultimately, the goal is to improve the algorithm's performance and accuracy.

4.1 The Issue With the Previous Models

One of the problems with the previous models is that the schedulers for the mean probability are easy to find. This means that the probability distribution of the schedulers is not homogeneous and lots of schedulers lead to the same probability, which is the mean probability. Moreover, extreme schedulers, where the maximum or minimum probability is far away from the mean probability, are rare or does not even exist in the previous tested models as shown in the figure bellow. For example, maybe only one scheduler out of a thousand has the maximum probability while all the others are closer to the mean probability.

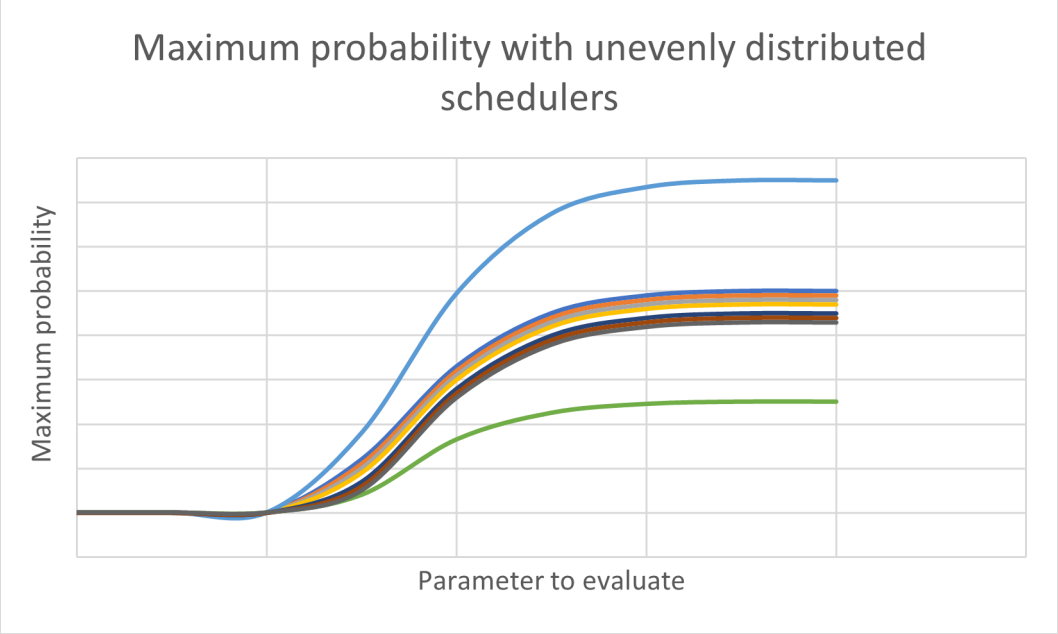

Figure 26: Maximum probability with unevenly distributed schedulers

The impact of the reduction factor parameter variation is very small on these types of models because a scheduler with a mean probability is always likely to be in the set, even if we drop lots of schedulers with a high reduction factor. Therefore, it is necessary to create new models that have a better distribution of the schedulers' probabilities in order to find easier and more often the maximum or minimum probability.

To address this issue, we will create a new model with a better distribution of the scheduler (see figure bellow) thus having a better distribution of the probabilities themselves. The model must be easy to understand and to modify if we want to create sub-models from it.

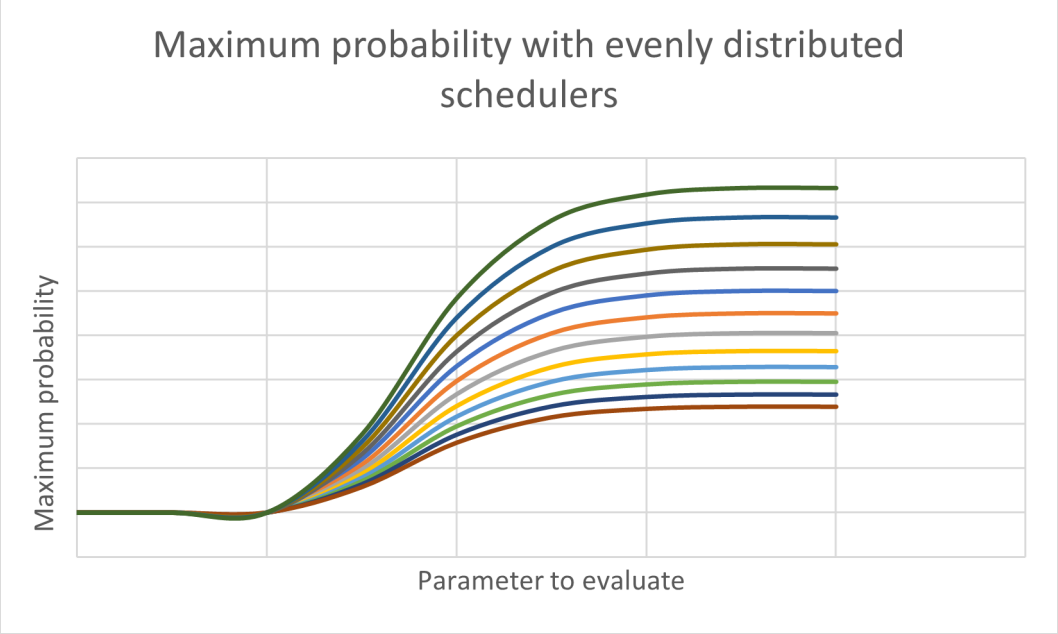

Figure 27: Maximum probability with evenly distributed schedulers

4.2 A New Model

The development of the new model is expected to improve the distribution of schedulers by making it more homogeneous. This means that not all schedulers will be located around the mean probability, but rather there will be more spread in their distribution. In other words, the new model will be able to identify extreme cases more efficiently by generating more schedulers that fall outside the mean probability. By increasing the number of extreme schedulers, the new model will be able to identify schedulers that were previously overlooked by previous models.

We also introduced noise into the new model. By adding noise, the algorithm becomes less predictable, making it harder for it to find the exact same solutions over and over again. This can make the algorithm take more risks and explore new possibilities, potentially leading to more diverse and innovative results. However, it's important to carefully calibrate the amount of noise added to the algorithm to ensure that it doesn't become too erratic and unstable. With the right amount of noise, the algorithm can strike a balance between taking calculated risks and maintaining stability, ultimately leading to better results.

In the new model, we have one initial state which is the state 0 where there are 1000 possible actions, which are to choose the next transition leading to a single state from a pool of 1000 possibilities. Each next state is identified by an

ID going from 1 to 1000. The goal of this model is to satisfy a certain property that requires a parameter, x, to be equal to 1. However, the parameter x can only take two values: 0 or 1. The choice of state can only be done in the initial state thus only one action is possible. This model leads to 1000 Markov Chain because only 1000 different choices can be made. We will call the state chosen after the initial state of the model. The initial state choice is the only possible transition of the model so the state does not change anymore after the first choice.

At each state, the probability of the parameter x being equal to 1 depends on the id of the selected state. The higher the ID of the selected next state, the higher the probability that x will be equal to 1. For example, if the selected state id is 1, the probability that x=1 will be 1/1400, on the other hand, if the selected state id is 1000, the probability that x=1 will be 1000/1400. The probability of the parameter x being equal to 1 is also affected by noise. Specifically, the noise probability makes x equal to 1 in 90% of the cases, while there is a 10% chance of noise occurring. The maximum theoretical probability of the model is then 73,29% which correspond to 10% of noise with a maximum probability of 90% plus 90% of non-noise with a maximum probability of 1000/1400 (0.1 * 0.9 + 0.9 * 1000/1400)

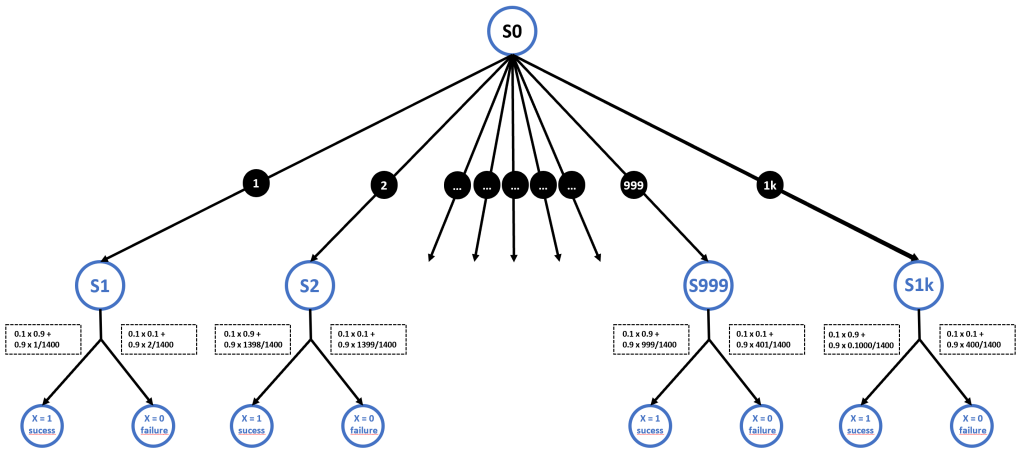

Figure 28: Model 1

We also created a variation of the model explained above with a global artificial extremum. This extremum corresponds to the state 1000 and has a higher maximum probability (97,2%) than the noise (90%) and theoretical one (73,29%). The interest of adding this extremum is to understand how the Smart Sampling algorithm behave when the maximum probability is rare and higher than the noise probability. In particular, we expect to see a different impact on the maximum probability computation when modifying the reduction factor parameter. This impact should ideally be different than with the

initial version of the new model.

To create the model in rml, a python script was very much needed due to the complexity of the model's initial state. With 1000 possible choices, the initial state generates 1000 lines of code that describe the possible next states. This would be impractical to write manually, and therefore a script is necessary. Python is a popular language for scientific computing and has a large library of tools that make it well suited for creating models such as this one. The script will generate the initial state and the possible next states, as well as the probabilities associated with each of those next states. It will also include the logic necessary to ensure that the parameter x is equal to 1 for a given property. With the script, we can easily modify the model and experiment different variations of it like the one with an artificial global extremum.

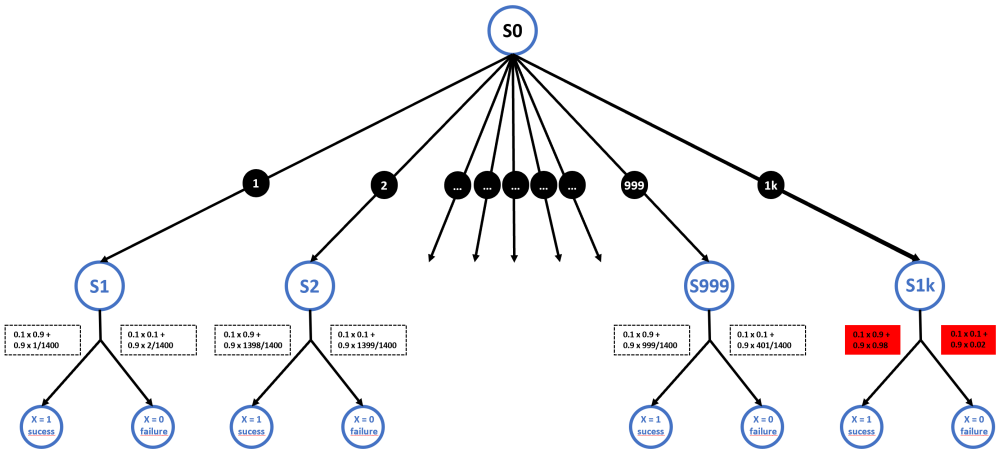

Figure 29: Model 2

4.3 Experimentation Protocol

Automating the experimentation process is crucial when dealing with a large number of test cases. In this case, we have chosen to use a Python script to automate the experimentation process. The script allows us to perform 100 experiments with different values of the reduction factor parameter. We can easily modify the values of the reduction factor and let the script perform the experiments. This saves us a lot of time and effort since we do not have to manually perform each experiment. Additionally, the script generates the results automatically, which makes it easy for us to analyse the data and draw conclusions.

To run the experiments, the script uses the Command Line Interface (CLI) plasma. This allows the script to interact with the Plasma Lab executable and perform the experiments. The script runs multiple experiments with different values of the reduction factor, which allows us to see how the probabilities

computed by the smart sampling algorithm vary with the reduction factor. By running the experiments through the script, we can easily change the reduction factor value and compare the results. This makes it easy for us to determine the impact of the parameter variation on the probabilities and time computed. Overall, using a Python script to automate the experimentation process saves us time and effort, and allows us to quickly and easily generate results that we can analyse and draw conclusions from.

The cli command used to launch an experimentation can be found hereafter:
PATH_TO_PLASMA_CLI.BAT launch
with the options:

- -conf PATH_TO_PLASMA_LAB.CONF
- -m PATH_TO_MODEL:rml
- -r PATH_TO_REQUIREMENT:bltl
- -a smartsampling
- -A "Epsilon"=0.03
- -A "Delta"=0.1
- -A "Budget"=1800
- -A "Maximum"=true
- -A "Reduction Factor"=2

4.4 Results And Analysis

4.4.1 Model 1

The parameter settings for the smart sampling algorithm are critical for its performance. In this case, the delta is set to 0.1, while epsilon is set to 0.03, which represents the relative error that is allowed in the estimation of the probability. Lastly, the budget is set to 1800, which is a relatively low value. This choice is deliberate, as it forces the algorithm to evaluate a smaller set of schedulers and to do so while allocate less budget, fewer simulations to each of these schedulers. This can help to identify the most promising schedulers quickly but may also lead to false optimum schedulers (noise influenced schedulers).

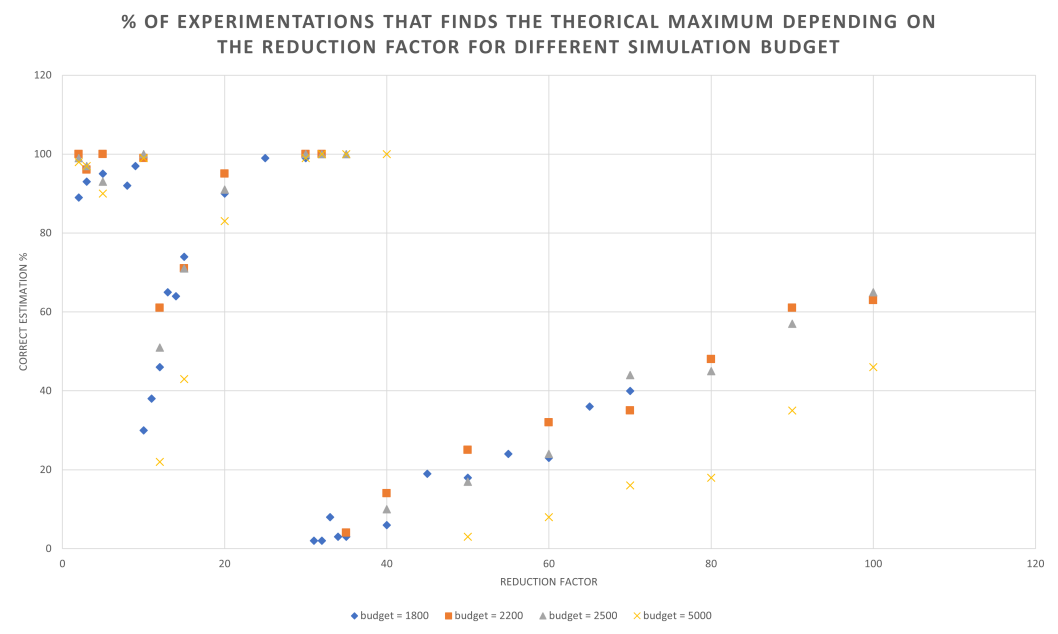

Figure 30: Percentage of experimentation that find the theorical maximum depending on the reduction factor

The results of the experimentation conducted for this new model are presented in the graph shown above. The graph depicts the percentage of experiments that find the theoretical maximum as a function of the reduction factor used in the optimisation algorithm. Interestingly, the graph shows huge drops in the percentage of successful experiments for reduction factors of 10 and 30. we also observe a relatively slow increase in the percentage of successful experiments after these drops. This behaviour is unexpected as we anticipated a constant decrease in the percentage of successful experiments while the reduction factor increase. These two unexpected behaviour prompted us to conduct several additional experiments with slightly different parameters to better understand the phenomenon (see section 4.5).

4.4.2 Model 2

The parameter are the same as for the original experimentation of the basic model: delta = 0.1, epsilon = 0.03 and the budget is 1800.

We can see on the graphic below that the maximum probability computed by the Smart Sampling algorithm for the model containing an artificial global maximum (second model) decrease while the reduction factor parameter increases. This result is different from the one obtained with the basic new model (without extremum). These results can be explained by correctly understanding how the reduction factor works in the algorithm. In fact, the higher the

reduction factor is, the more schedulers are removed at each step. This means that the optimum scheduler has more chance to be set apart compared with a lower reduction factor value (if the optimum scheduler is in the initial set of schedulers). On the other hand, a higher reduction factor also means that the budget allocated to each of the remaining scheduler is greater than with a low value of the reduction factor. This allows the smart sampling algorithm more time to correctly evaluate a single scheduler thus keeping the best ones for the next iteration of the algorithm. But this mechanism is only viable after the first iteration. In fact, the size of the initial set of scheduler is always the same whether the reduction factor value is high or low and the budget allocated to each scheduler is always the same. This is why, with a higher reduction factor, the optimum scheduler may be dropped in the first iteration because the smart sampling algorithm does not have time to evaluate the scheduler correctly. This shows us that the initial set of scheduler is critical when it comes to finding the best scheduler.

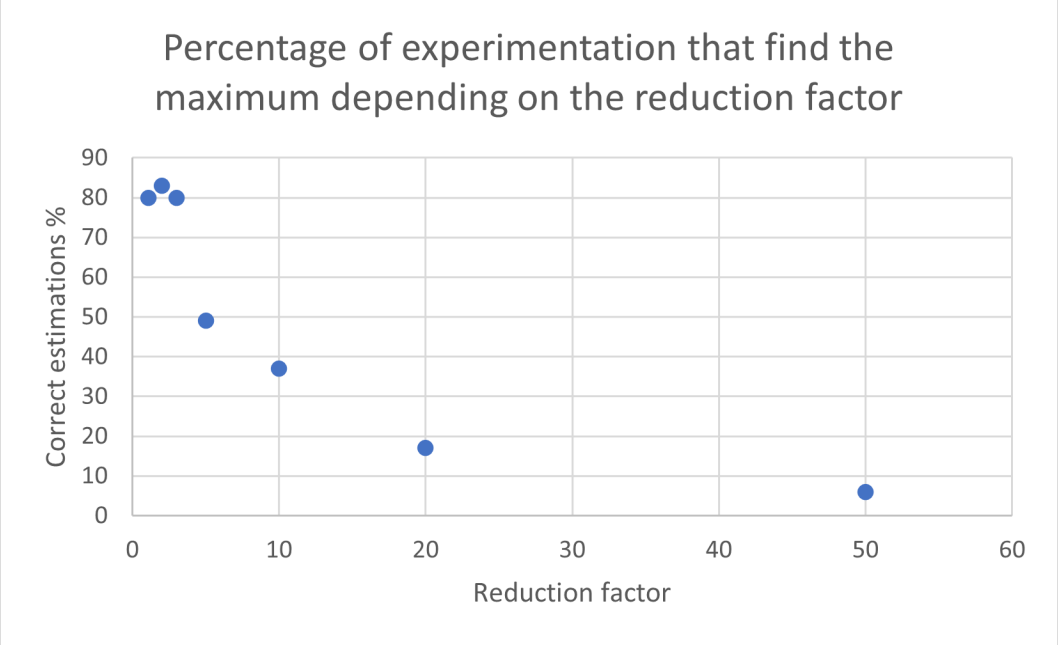

Figure 31: Percentage of experimentation that find the maximum depending on the reduction factor

In Conclusion, the reduction factor is a crucial parameter in the smart sampling algorithm. It defines the speed at which the algorithm terminates. In general, increasing the reduction factor decreases the number of schedulers evaluated, reducing the computational cost thus the time needed to find the optimum. However, it also leads to a decrease in the probability that the algorithm actually estimates a correct (epsilon,delta)-estimation.

4.5 Why Do We Have Drops ?

As shown in the section 4.4.1, the experimentation with the new model without artificial global extremum leads to drops occurring around reduction factors of 10 and 30. In this section, we will conduct various experimentations on this model to understand the origin of these drops. The experimentations are done using the new basic model and the smart sampling algorithm.

The first approach to understand the phenomenon is to make small variations of the parameters and to conduct new experimentation with these variations. The goal is to determine which of the parameters used in the experimentation influence the drops. The first parameter we decided to modify is the simulation budget given to each experimentation. This parameter is crucial for the smart sampling algorithm because the more budget it has, the more time the algorithm passes to evaluate the schedulers.

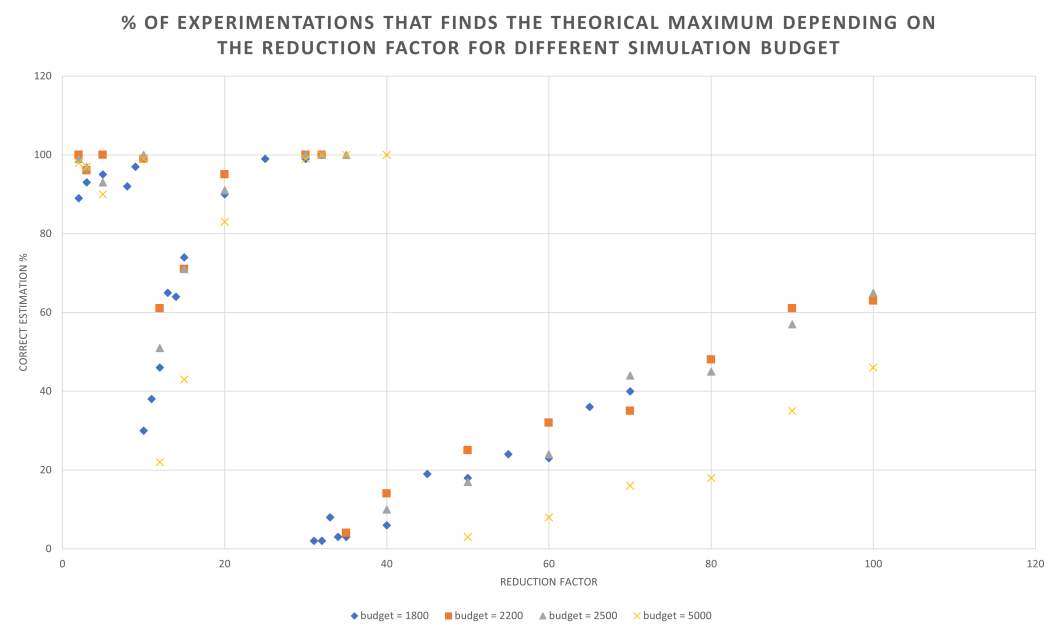

Figure 32: Percentage of experimentations that finds the theorical madimum depending on the reduction factor for different simulation budget

We can see on the graphic above that with more simulation budget, the drops occurs later (at higher reduction factors). These results tell us that the simulation budget influences the drops and maybe the directly linked to them.

To better understand the budget management of the smart sampling algorithm, we must dive into the algorithm code. The code is available at appendices A. The first thing we observe in line 285 is that the simulation budget is

more of a per-iteration budget than a total simulation budget. In fact, at every iteration, the algorithm use the whole budget to evaluate the schedulers. The simulation budget for the whole process is thus the simulation budget (budget at each step) times the number of steps.

The second observation due to the code meticulous analysis is that the budget allocated to every scheduler is sometimes replaced by the chernoff bound and that the algorithm stops when the budget per scheduler is lower than the Chernoff bound (line 324). The Chernoff bound is a mathematical theorem used in probability theory and statistics. It provides a way to estimate the probability that a sum of independent random variables deviates significantly from its expected value. More specifically, the Chernoff bound gives an upper bound on the probability that the sum of independent random variables exceeds a certain threshold. This bond is used in the smart sampling algorithm to overcome the budget allocated to every scheduler when this one is estimated to be too large. When this happens, the budget per scheduler becomes the chernoff bound instead of the simulation budget divided by the number of schedulers which decrease the simulation budget of the iteration. One way to better understand this feature is to analyse a step-by-step iteration of the Smart Sampling algorithm.

```
iteration 0
-> # schedulers: 1000
-> # simulation per scheduler: 2
-> simulation budget at iteration: 2000
------------------------------------------------------------------------
iteration 1
-> # schedulers: 333
-> # simulation per scheduler: 6
-> simulation budget at iteration: 1998
-> next chernoff bound: 3867
------------------------------------------------------------------------
iteration 2
-> # schedulers: 111
-> # simulation per scheduler: 18
-> simulation budget at iteration: 1998
-> next chernoff bound: 3258
------------------------------------------------------------------------
iteration 3
-> # schedulers: 37
-> # simulation per scheduler: 54
-> simulation budget at iteration: 1998
-> next chernoff bound: 2634
------------------------------------------------------------------------
iteration 4
-> # schedulers: 12
-> # simulation per scheduler: 166
-> simulation budget at iteration: 1992
-> next chernoff bound: 2028
------------------------------------------------------------------------
iteration 5
-> # schedulers: 4
-> # simulation per scheduler: 500
-> simulation budget at iteration: 2000
-> next chernoff bound: 1280
------------------------------------------------------------------------
```

Figure 33: Smart Sampling iterations

The figure above shows an execution of the Smart Sampling algorithm with the new model. The parameter used for this simulation are: $\epsilon = 0.03$, $\delta = 0.1$, budget = 2000 and reduction factor = 2. We can see that the last iteration is the fifth one because the next Chernoff bound is lower than the next simulation per scheduler (2000 because 2000 / (int(4 / 3))). This stopping condition halts the algorithm with 4 schedulers left with a budget of 500 allocated to each of them. This relatively high number of schedulers at the last iteration allows less computational resources for the evaluation of the remaining schedulers which can lead to bad assessments of these schedulers.

```
iteration 0
-> # schedulers: 1000
-> # simulation per scheduler: 2
-> simulation budget at iteration: 2000
------------------------------------------------------------------------
iteration 1
-> # schedulers: 333
-> # simulation per scheduler: 6
-> simulation budget at iteration: 1998
-> chernoff bound: 4478
------------------------------------------------------------------------
iteration 2
-> # schedulers: 111
-> # simulation per scheduler: 18
-> simulation budget at iteration: 1998
-> chernoff bound: 3867
------------------------------------------------------------------------
iteration 3
-> # schedulers: 37
-> # simulation per scheduler: 54
-> simulation budget at iteration: 1998
-> chernoff bound: 3258
------------------------------------------------------------------------
iteration 4
-> # schedulers: 12
-> # simulation per scheduler: 166
-> simulation budget at iteration: 1992
-> chernoff bound: 2634
------------------------------------------------------------------------
iteration 5
-> # schedulers: 4
-> # simulation per scheduler: 500
-> simulation budget at iteration: 2000
-> chernoff bound: 2028
------------------------------------------------------------------------
iteration 6
-> # schedulers: 1
-> # simulation per scheduler: 1280
-> simulation budget at iteration: 1280
-> chernoff bound: 1280
```

Figure 34: Smart Sampling iterations without stopping conditions

The figure above shows an execution of the Smart Sampling algorithm without the stopping condition described above. We can see that at the last iteration, the simulation budget is overcome by the chernoff bound, the simulation budget is 1200 instead of 2000 which can result in a bad evaluation of the remaining scheduler.

The early stop caused by the condition and the poor evaluation of the remaining scheduler when the chernoff bound replace the simulation budget are explicable causes of the drops discovered with the new model. In fact, if the last(s) scheduler(s) is(are) not sufficiently evaluated, the probability computed could be influenced by the noise probability resulting in a bad evaluation of the last(s) scheduler(s).

4.6 Modification Of The Smart Sampling Algorithm

to overcome the Chernoff bound issues that leads to the drops problem, we can modify the Smart Sampling algorithm code. The first modification is to change the stopping condition. Instead of terminate the execution of the Smart Sampling algorithm when the budget per schedulers is lower than the Chernoff bound, we could stop the algorithm once there is only one scheduler left. This modification will force the algorithm to keep the best overall scheduler while maximise the budget allocated to him. The goal of the second modification is to keep the budget at each iteration constant. To achieve it, we must divide the budget equally between each remaining scheduler while not being influenced by the Chernoff Bound.

```
iteration 0
-> # schedulers: 1000
-> # simulation per scheduler: 2
-> simulation budget at iteration: 2000
------------------------------------------------------------------------
iteration 1
-> # schedulers: 333
-> # simulation per scheduler: 6
-> simulation budget at iteration: 1998
------------------------------------------------------------------------
iteration 2
-> # schedulers: 111
-> # simulation per scheduler: 18
-> simulation budget at iteration: 1998
------------------------------------------------------------------------
iteration 3
-> # schedulers: 37
-> # simulation per scheduler: 54
-> simulation budget at iteration: 1998
------------------------------------------------------------------------
iteration 4
-> # schedulers: 12
-> # simulation per scheduler: 166
-> simulation budget at iteration: 1992
------------------------------------------------------------------------
iteration 5
-> # schedulers: 4
-> # simulation per scheduler: 500
-> simulation budget at iteration: 2000
------------------------------------------------------------------------
iteration 6
-> # schedulers: 1
-> # simulation per scheduler: 2000
-> simulation budget at iteration: 2000
------------------------------------------------------------------------
```

Figure 35: Smart Sampling iterations

4.7 Modified Smart Sampling Algorithm Results

In this section, we present the experimentation and analysis of the modified smart sampling algorithm version. The goal of the experimentation was to evaluate the algorithm's performance in terms of correct estimation, focusing on the reduction factor and different budget values. The graph presented in Figure bellow illustrates the percentage of correct estimation depending on the reduction factor for multiple budgets. The model used for this experimentation is the new model without the artificial global maximum. The parameters set

for the experimentation were epsilon = 0.03 and delta = 0.1.

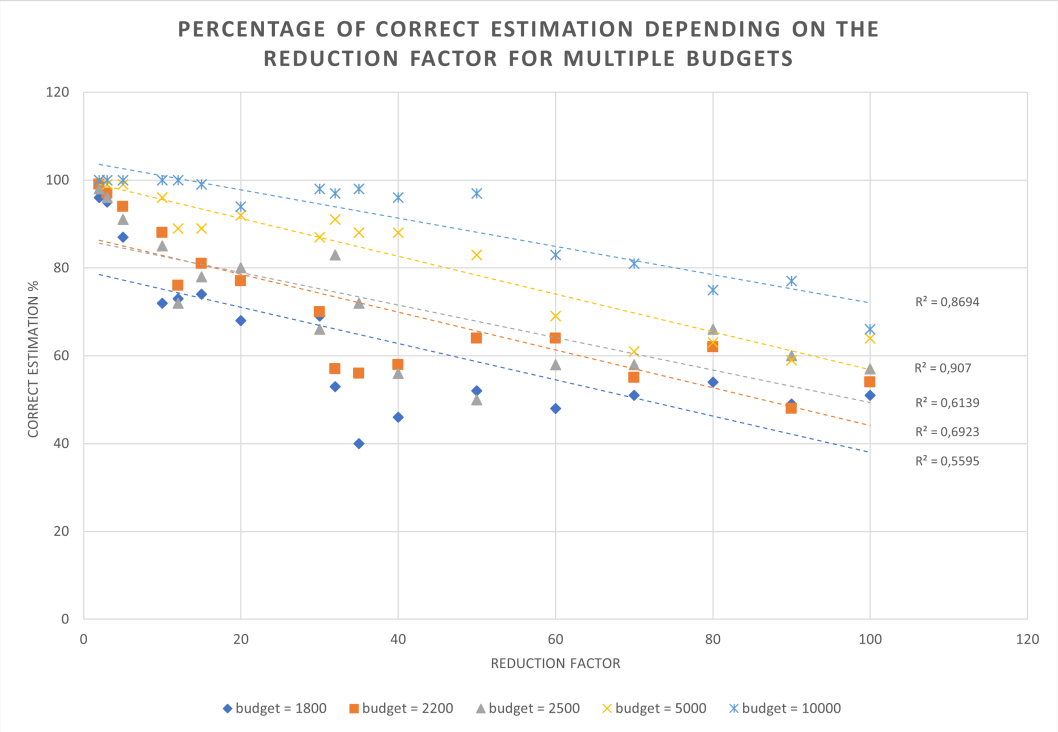

Figure 36: Smart Sampling Alternative version with multiple budgets

To obtain the data points for the graph, we conducted 100 experiments and computed the percentage of values within the theoretical probability value, with a margin of plus or minus epsilon (0.03). The dotted lines in the graph represent a linear interpolation of the data points, providing a visual representation of the trends observed.

From the graph, it is evident that the higher the budget, the higher the percentage of correct estimations. This finding shows that allocating more resources to the algorithm leads to improved accuracy of the estimations. We also observe a significant improvement in the correctness of the estimation when the budget is increased by 5.5 times (from 1800 to 10000) and employing a high reduction factor. In fact, the percentage of correct estimations nearly doubled compared to the previous budget value. This indicates that using a high reduction factor combined with a high budget leads to acceptable results compared to low reduction factors combined with a low budget.

In addition to considering the estimation performance and efficiency of the smart sampling algorithm, it is crucial to examine the impact of the budget on the time needed to complete an execution. This section focuses on assessing

the relationship between the budget allocated and the execution time, with the aim of determining the optimal budget for achieving a balance between estimation accuracy and computational efficiency.

By evaluating the time require to conduct 100 experimentation, we anticipated that increasing the budget would lead to a proportional increase in the execution time. This expectation is based on the assumption that a larger budget would necessitate more computations and evaluations, resulting in a longer runtime. Our analysis (sefigure bellow) revealed that the computation time slightly increased with the budget, albeit not as much as initially expected

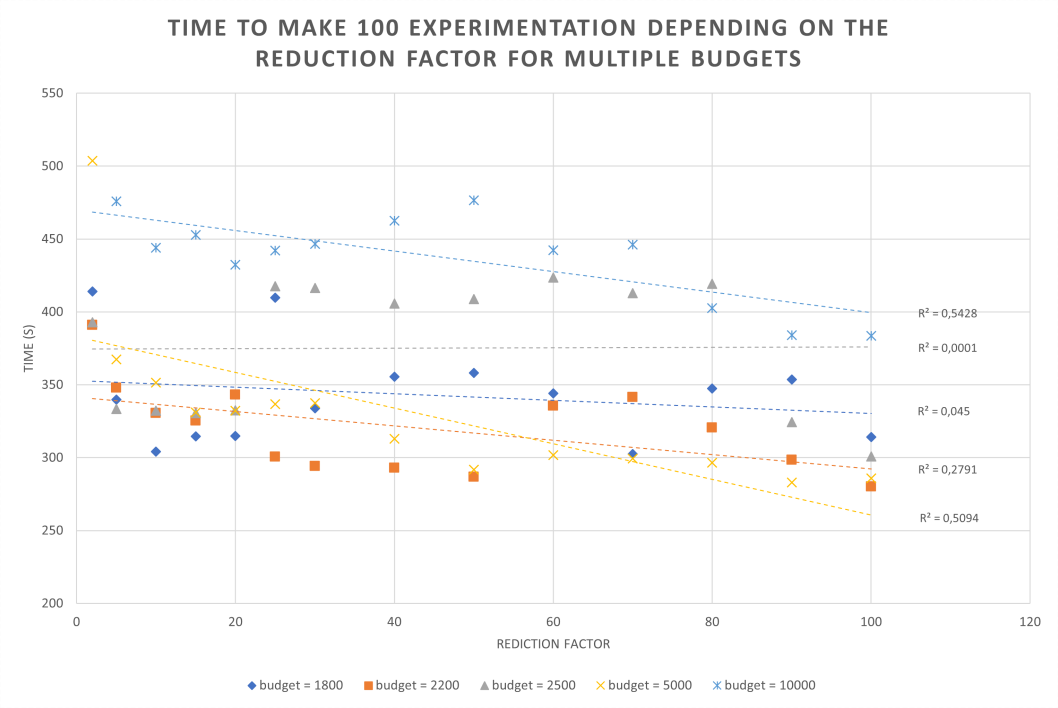

Figure 37: Time to make 100 experimentation depending on the reduction factor for multiple budgets

Interestingly, we observed a significant variation in the computation time across different budget values. Surprisingly, it took less time to complete the execution for budgets of 2200 and 5000 compared to the baseline budget of 1800. However, for budgets of 2500 and 10000, the computation time was relatively higher. This variation indicates that the budget is not the only parameter influencing the execution time of the smart sampling algorithm.

Despite the increased computation time associated with higher budgets, it is important to note that the overall increase remains within acceptable limits. The additional computational overhead introduced by larger budgets does not

significantly limit the feasibility of further budget increases. This observation is encouraging, as it implies that the algorithm can effectively handle increased resources without compromising its practical applicability.

Overall, the experimentation results demonstrate the influence of budget allocation and reduction factor on the correctness of the estimation. Increasing the budget positively impacts the algorithm's performance, particularly when aiming for higher reduction factors. Moreover, the computation time of the smart sampling algorithm shows a moderate increase with an increase in the budget. However, the relationship between budget and computation time is not linear, as shown by the variation observed across different budget values. Importantly, the increased computation time associated with larger budgets remains within acceptable limits, indicating that further budget increases can be considered without significantly hindering the algorithm's efficiency.

5 Smart Sampling Algorithm Improvements

In this section of the master thesis, we aim to enhance the effectiveness of the algorithm by introducing specific modifications to its behaviours. The goal is to achieve better estimations and ultimately improve the algorithm's performance. The proposed modifications will focus on two key aspects: improving the existing behaviour of the algorithm and introducing new schedulers. These changes come from the understanding that the initial population and its evaluation significantly influence the algorithm's subsequent execution.

The first type of modification involves enhancing the current behaviours of the algorithm, particularly in terms of evaluating the schedulers. The second type of modification centres around devising techniques for scheduler generation during the algorithm execution.

5.1 More Budget At First Iteration

In this section, we describe the first modification implemented in the smart sampling algorithm to improve its effectiveness in generating accurate estimations. The modification involves allocating a larger budget for the initial evaluation of the schedulers. By increasing the budget at this crucial stage, we aim to enhance the quality of the initial set of schedulers and reduce the influence of noise to ultimately improve the overall estimation performance of the algorithm.

To evaluate the effectiveness of this modification, we conducted experiments

using the new model without artificial global extremum and the corrected version of the smart sampling algorithm. The experimentation settings included a fixed budget of 1800, an epsilon value of 0.03, and a delta value of 0.1. These parameters ensured consistency across the experiments and allowed for a fair comparison of the results.

In our experimentation, we introduced variations in the budget allocation for the initial evaluation of the schedulers. Specifically, we tested different budget multipliers, including the initial budget of 1800, twice the initial budget (3600), three times the initial budget (5400), five times the initial budget (9000), and ten times the initial budget (18000). This range of budgets provided a comprehensive analysis of the impact of increased resources on the quality of the initial set of schedulers.

The results obtained from the experiments were analysed and visualised in the form of a graph presented below. In the graph, the dotted lines represent a linear interpolation of the data points, allowing for a clear visualisation of trends.

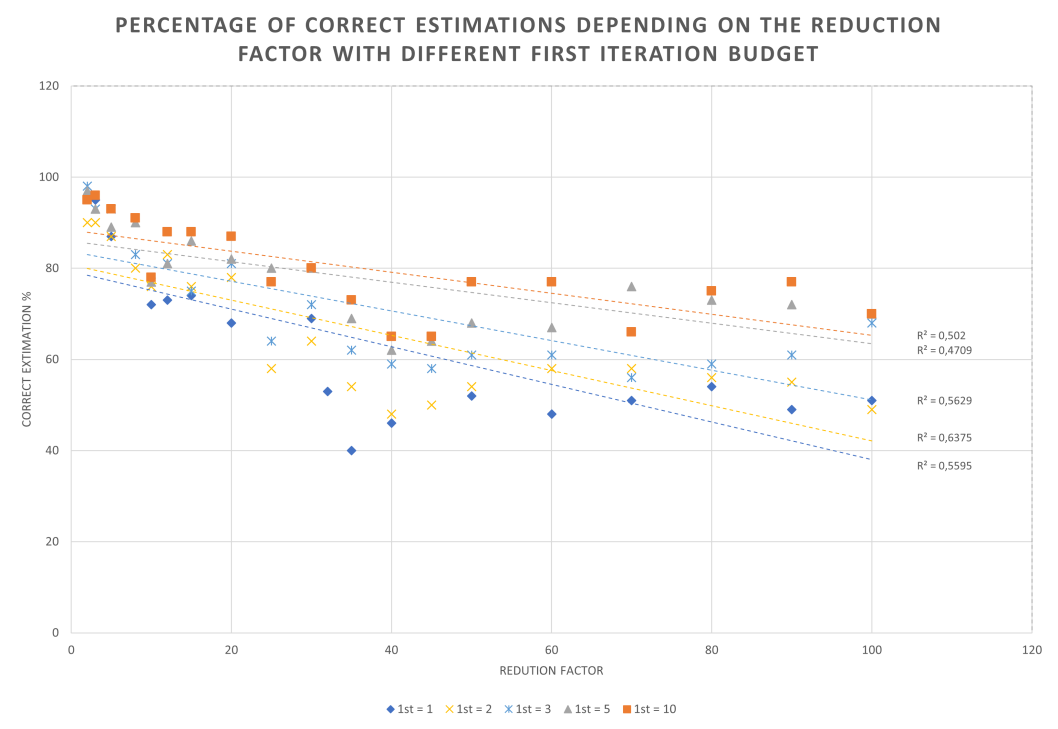

Figure 38: Percentage of correct estimations depending on the reduction factor with different first iteration budget

The analysis of the experimental data revealed valuable insights into the relationship between budget allocation and the quality of the final estimations

of the smart sampling algorithm. Through our experimentation, we observed that increasing the budget for the initial evaluation had a positive impact on the estimation performance of the algorithm.

As the budget multiplier increased, we noticed a corresponding improvement in the algorithm's ability to generate accurate estimations. The linear interpolation of the data points effectively captured this increasing trend, providing a visual representation that clearly demonstrated the correlation between budget allocation and estimation quality.

These findings emphasise the critical importance of allocating sufficient resources for the initial evaluation stage of the smart sampling algorithm. By providing a larger budget at the outset, we enhance the algorithm's ability to explore the solution space more thoroughly and identify better-quality solutions. The increased budget allows for a more comprehensive and accurate sampling of the initial population.

Beside the performance improvement, it is crucial to consider the effect of the first iteration budget on the algorithm's runtime. This section focuses on assessing the relationship between the first iteration budget and the execution time, with the aim of determining the optimal initial budget for achieving a balance between estimation accuracy and computational efficiency.

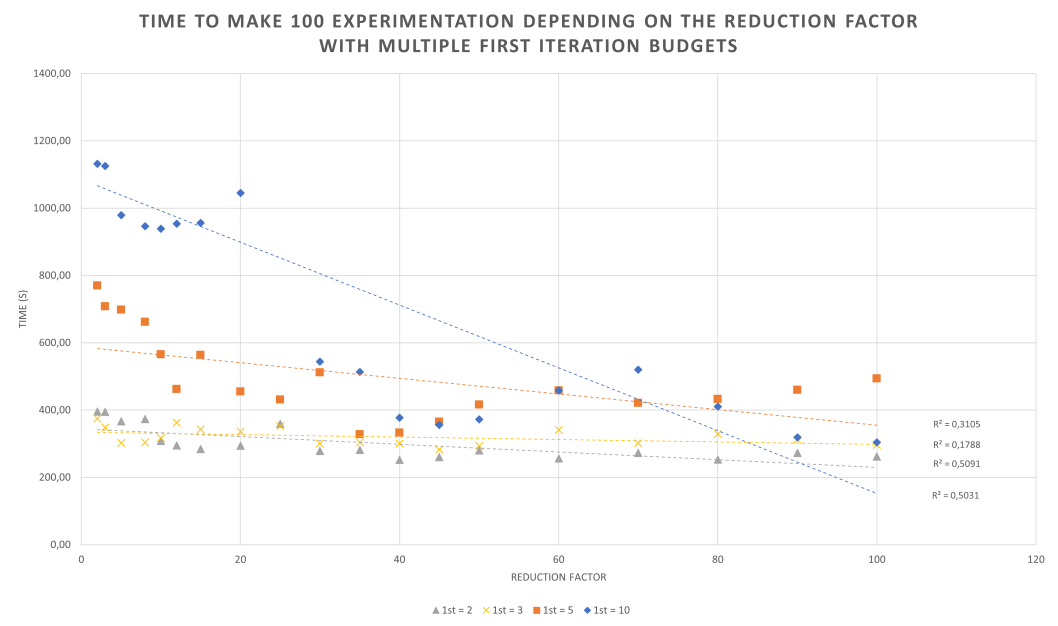

Figure 39: Time to make 100 experimentation depending on the reduction factor with multiple first iteration budgets

We hypothesised that increasing the first iteration budget would result in a proportional increase in the execution time. This expectation was based on the assumption that a higher initial budget would require more extensive computations and evaluations, leading to a longer runtime. By computing the time required to conduct 100 experiments, we aimed to gain insights into the relationship between the first iteration budget and execution time.

Our analysis revealed that, as expected, the execution time of the smart sampling algorithm increased with a larger first iteration budget (see figure above). This finding indicates that allocating more resources to the initial evaluation stage prolongs the execution process. The increase in execution time can be attributed to the additional computations and evaluations performed to explore the first scheduler space in greater detail. Interestingly, we also observed that the difference in computation time between different first iteration budgets was less significant when a high reduction factor was employed.

Based on these findings, it can be inferred that utilising a high reduction factor in conjunction with a high first iteration budget may be a favourable approach. Although the computation time still increases with a larger first iteration budget, the difference in execution time compared to lower first iteration budget scenarios becomes relatively smaller. Using a high reduction factor allows increasing the first iteration budget without impacting the total computation time thus benefiting of the positive impact of a greater first iteration budget on the correctness of the estimations.

5.2 Random Reintroductions

In this section, we present the second modification implemented in the smart sampling algorithm, which involves introducing randomly generated schedulers to the scheduler set at multiple steps of the algorithm. This modification aims to enhance the algorithm's exploration of the solution space and promote diversity in the sampled population.

As for the first modification, the experimentation of this modification was conducted using the new model without artificial global extrema and the corrected version of the smart sampling algorithm. The parameters used for the experimentation included a fixed budget of 1800, an epsilon value of 0.03, and a delta value of 0.1. These parameters ensured consistency across the experiments and facilitated a fair comparison of the results.

For this modification, the number of reintroductions of schedulers depended on the reduction factor. The reduction factor played a crucial role in determining the frequency of reintroduction. As the reduction factor increased,

indicating a higher level of reduction in population size, the number of reintroductions also increased. This approach aimed to counterbalance the reduction due to the reduction factor and maintain an adequate level of diversity in the sampled population.

The results of the experimentation were analysed and visualised through a graph displayed below. In the graph, the dotted lines represented a linear interpolation of the data points, offering a clear visualisation of the trends observed.

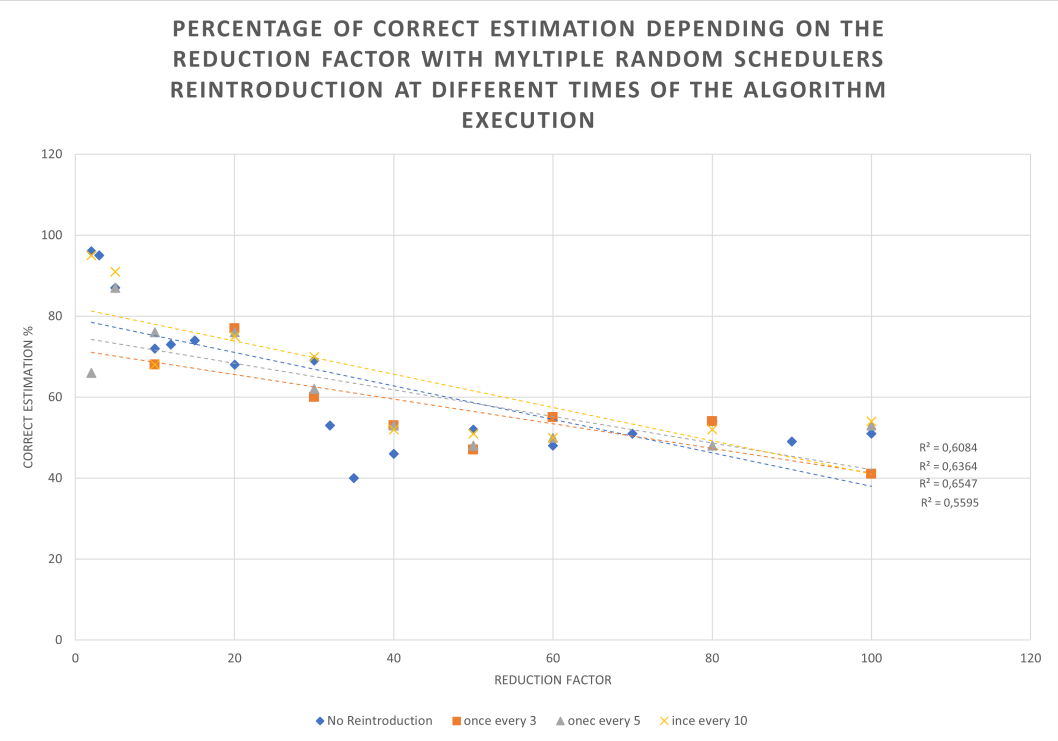

Figure 40: Percentage of correct estimation depending on the reduction factor with myltiple random schedulers reintroduction at different times of the algorithm execution

Our analysis revealed that the random reintroduction of schedulers had no significant impact on the efficiency of the probability computed. Despite introducing random solutions to the scheduler set, the algorithm's estimation performance remained consistent. The reintroduction frequency had no discernible effect on the efficiency of the probability computed. Reintroducing schedulers every 3, 5, or 10 iterations yielded similar estimation results, indicating that the choice of frequency did not significantly impact the algorithm's estimation efficiency

In conclusion, we introduced two modifications aimed at improving the ef-

fectiveness of the smart sampling algorithm. The first improvement involved allocating more budget at the first iteration, which led to promising results with only a marginal increase in execution duration. This modification enhanced the algorithm's ability to correctly evaluate the important first set of schedulers, leading to more efficient process. On the other hand, the second modification, known as random reintroduction, did not significantly impact the performance of the algorithm. Although it was expected to introduce variation and potentially enhance the scheduler exploration process, the experimental results did not demonstrate any substantial improvement. Nonetheless, this finding provides valuable insights into the algorithm's behaviour and suggests that alternative approaches or further refinements may be necessary to achieve significant performance gains like intelligent reintroduction of schedulers...

6 Use Cases Validation

The goal of this section is to validate the results obtained in Section 4, which focused on the modification of the smart sampling behaviour, especially the iteration behaviour, as well as the findings from Section 5, which presented two improvements to the smart sampling algorithm. In order to validate these results and assess the practical implications of the proposed modifications, we conducted a series of experiments on two real-world use cases already presented in earlier, the wireless LAN protocol ant the Gossip protocol. Although these two models are more complex than the one creates for in section 3, we expect to see the same results as the ones presented in the previous sections.

For the two experimentations conducted in this section, several parameters were carefully selected to ensure reliable and meaningful results. The value chosen for these parameters were as follows: $\delta = 0.1$, $\epsilon = 0.01$, budget $= 15000$ while performing 50 experimentation for each reduction factor. These parameter values were chosen based on the experimentation made in [18].

To ensure reasonable experimentation duration without compromising the validity of the results, slight modifications were made to the original experimentation parameters. Specifically, the value of δ was adjusted from 0.01 to 0.1. This adjustment allowed for a lower budget of 15,000 instead of the original 100,000. By reducing the budget, we aimed to speed up the experimentation process without sacrificing too much about the reliability of the results. In addition to optimising the experimentation parameters, we also made improvements to the test algorithm itself to further expedite the experimental process.

The theoretical probability values of the WLAN use case can be found on this website [29] while the theoretical probability value for the Gossip use case was determined experimentally as no exact values were given at the RPISM

webpage of the protocol [30] . These theoretical probabilities serve as reference points against which the empirical results obtained from the experimentations can be evaluated and analysed. By comparing the experimental results with the corresponding theoretical probabilities, we aim to assess the accuracy and validity of the modifications and improvement mode to the Smart Sampling algorithm.

For these experimentations, we firstly tested the two uses case models using the original smart sampling algorithm. This served as a baseline to assess the algorithm's further modification and improvements. Subsequently, the modified version of the smart sampling algorithm was utilised to retest the two use case models. This goal of modified section is to remove the drop phoenomen discovered in section 4. Lastly, we tested the two Smart Sampling algorithm improvement (first iteration budget and random reintroduction) on the two uses cases models.

By conducting experiments on real-world use cases and implementing these modifications, we aim to validate the effectiveness and practical viability of the proposed smart sampling behaviour modifications and improvements. The results obtained from this section will provide valuable insights into the performance and efficiency gains achieved through these modifications, as well as their applicability to real-world scenarios.

6.1 Experimentation Protocol

In order to expedite the experimentation process and reduce the overall duration, we introduced a modification to the algorithm presented in Section 4.3. As a reminder, the algorithm involves conducting a certain number of experiments for each reduction factor and store the result in a .csv file. To further enhance efficiency, we introduce a multithreading approach, allowing for simultaneous execution of multiple experiments (Appendix B).

By leveraging multithreading, we can take advantage of the computational capabilities of modern processors and perform concurrent experiments. This parallelisation technique enables us to distribute the workload across multiple threads, effectively reducing the time required to complete a set of experiments. The modified algorithm takes advantage of multithreading by dividing the experimentation workload for a particular reduction factor into multiple threads. Each thread independently performs an experiment with the given reduction factor, effectively allowing multiple experiments to run concurrently.

The introduction of multithreading in the experimentation process has proven to be highly beneficial in terms of reducing the overall duration. In

our experimental setup, we observed a remarkable improvement in execution time, with a reduction of approximately 80% compared to the previous sequential implementation. This improvement significantly enhances the scalability and efficiency of the tester, especially when dealing with lots of experimentations.

6.2 WLAN And Gossip Experimentation Analysis

In this section, we present the experimental results obtained from the evaluation of the WLAN and the Gossip model using the original Smart Sampling algorithm, the modified Smart Sampling algorithm and the improved Smart Sampling algorithm.

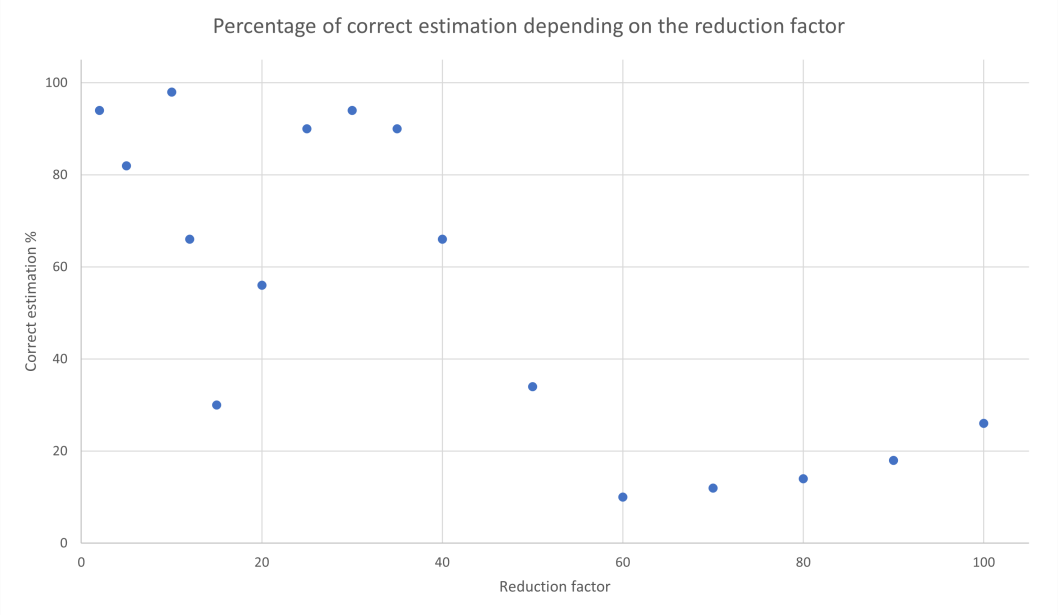

Figure 41: WLAN model with original Smart Sampling Algorithm

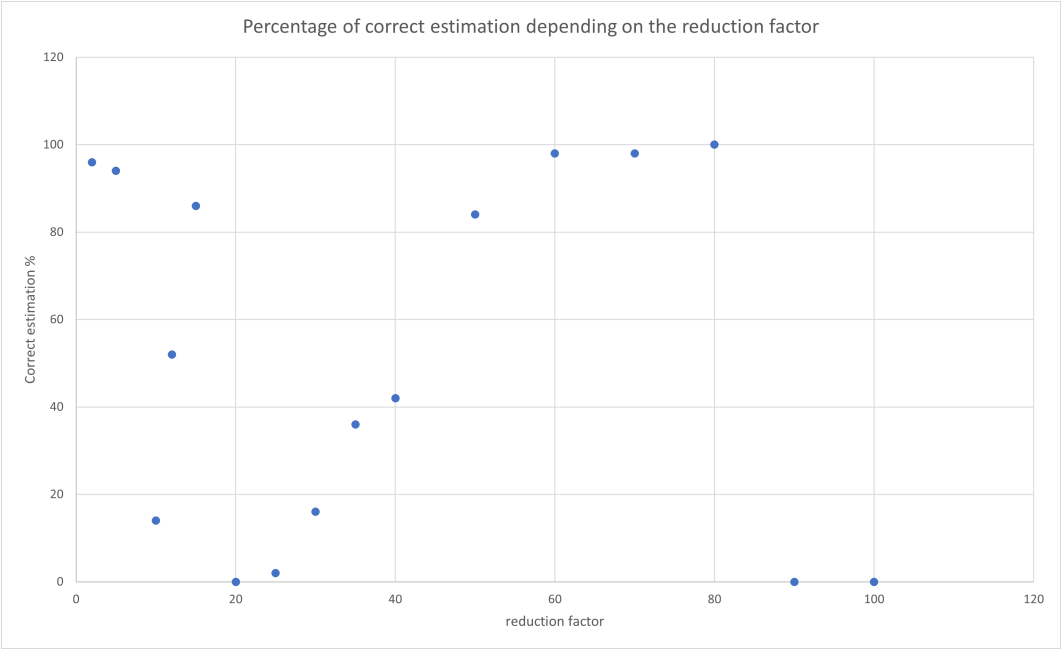

Figure 42: Gossip model with original Smart Sampling Algorithm

The graphs in figures 41 and 42 illustrates the experimentation results achieved through the utilization of the original smart sampling algorithm using the WLAN and Gossip models respectively. It showcases the percentage of experimentation that successfully identified the theoretical maximum as a function of the reduction factor. Notably, the graph reveals certain drops in the percentage of successful experimentation for specific reduction factors.

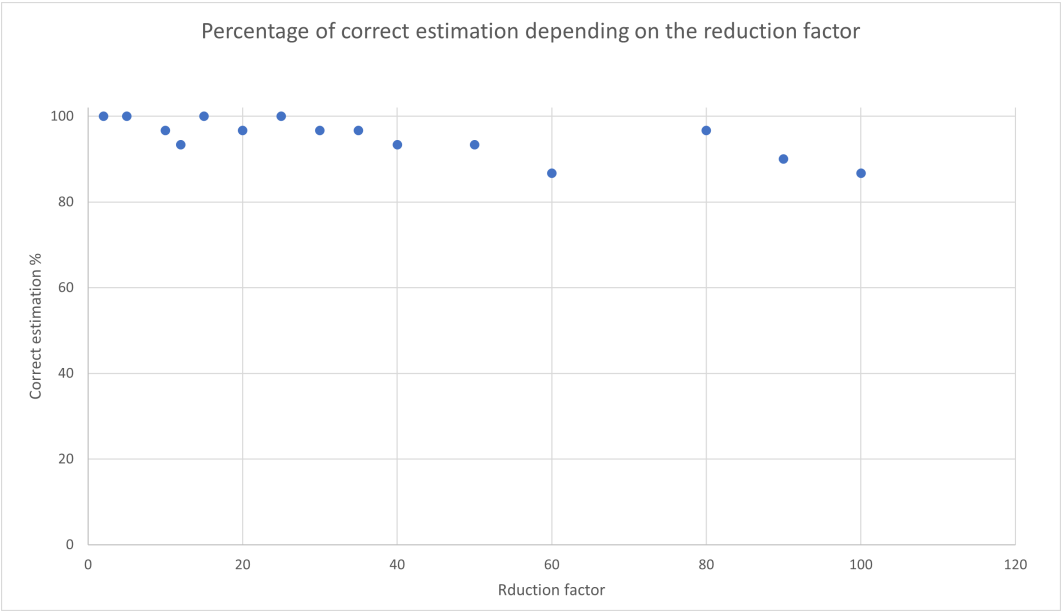

Figure 43: WLAN model with modified Smart Sampling Algorithm

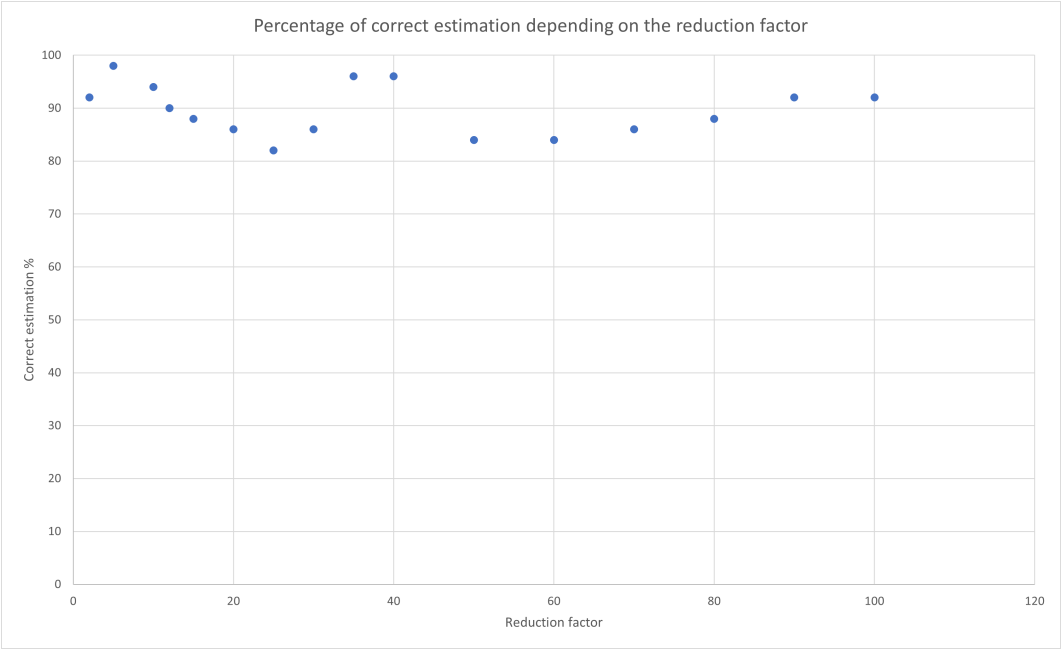

Figure 44: WLAN model with modified Smart Sampling Algorithm

To address the observed drops, the smart sampling algorithm was modified, incorporating specific improvements aimed at mitigating this issue. The two graphs above represents the experimentation results obtained using the modified Smart Sampling algorithm using the WLAN and Gossip model. Here, we observe a significant improvement, as the drops that were evident in the first graph have disappeared. This outcome aligns with our expectations, demonstrating the effectiveness of the modifications in improving the algorithm.

Then we analysed the experimental results obtained from the evaluation of the first Smart Sampling improvement, which involves allocating more budget at the first iteration. The graph below showcases the experimentation results, illustrating the percentage of experimentation that successfully identified the theoretical maximum as a function of the reduction factor using multiple first iteration reintroduction multiplier (1, 2, 3, 5 and 10). For the WLAN experimentation, we had to increase the value of the precision we use to assess tho correctness of the computed probability. This modification was necessary due to the high correctness of the estimation computed with the modified Smart Sampling algorithm. A higher precision ensure a lower correct estimation percentage and allow further improvement of the algorithm to be visible on the graphic.

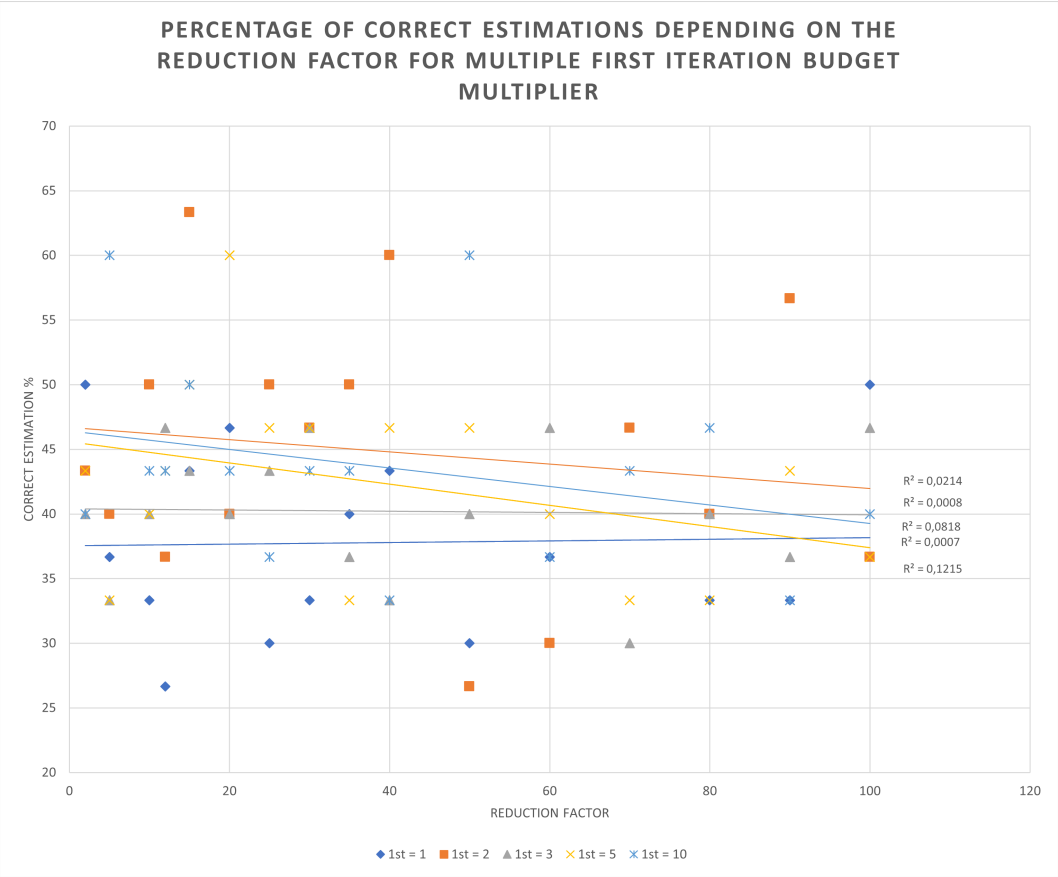

Figure 45: WLAN model with first budget iteration improvement

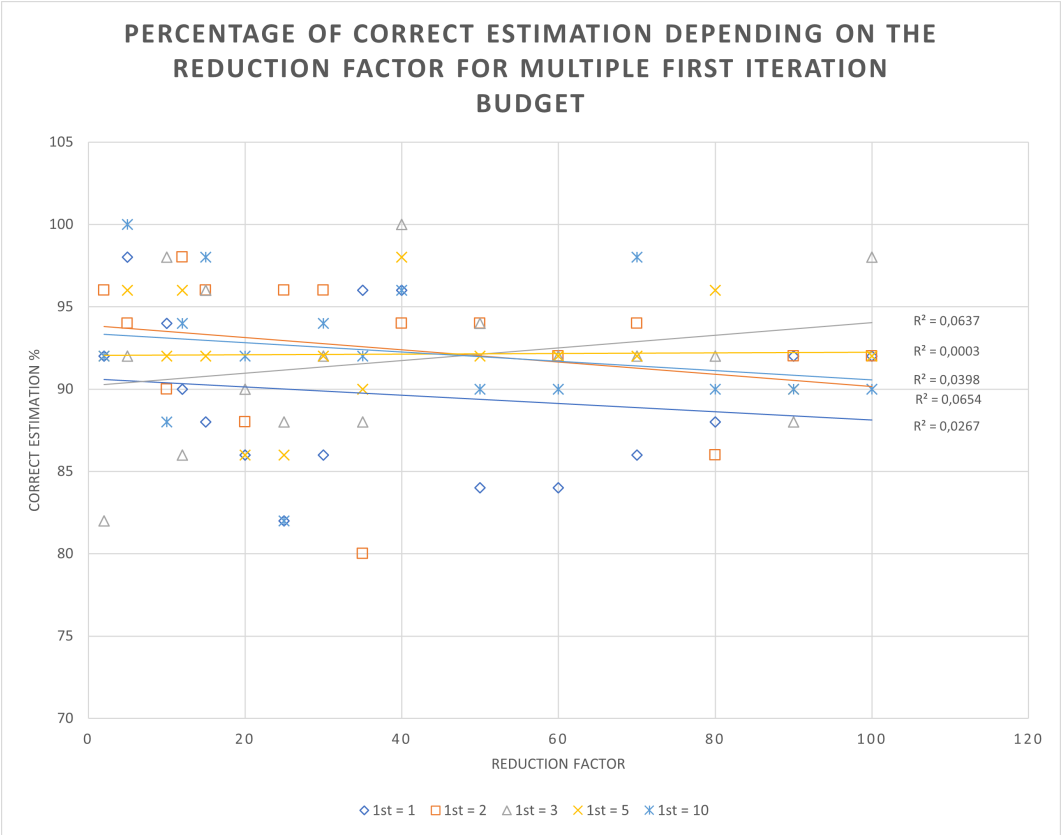

Figure 46: Gossip model with first budget iteration improvement

Upon careful examination of the graphics, we observe that each experimentation with an increased budget at the first iteration (1, 2, 3, 5, and 10 times more budget) consistently yields better results compared to the original experimentation without any additional budget allocation at the first iteration. This finding supports the results of the section 5. However, it is important to note that we do not observe a clear trend or pattern among the different experimentation cases (2, 3, 5, and 10 times more budget). The results vary across different reduction factors, and no specific experimentation consistently outperforms the others across the entire range of reduction factors.

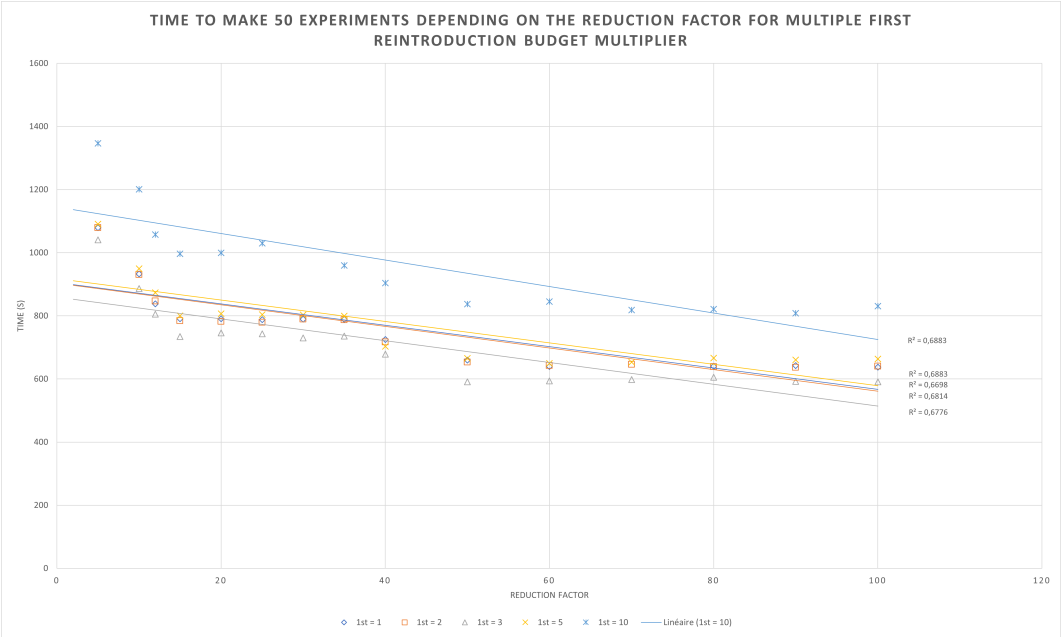

Figure 47: WLAN model time with first budget iteration improvement

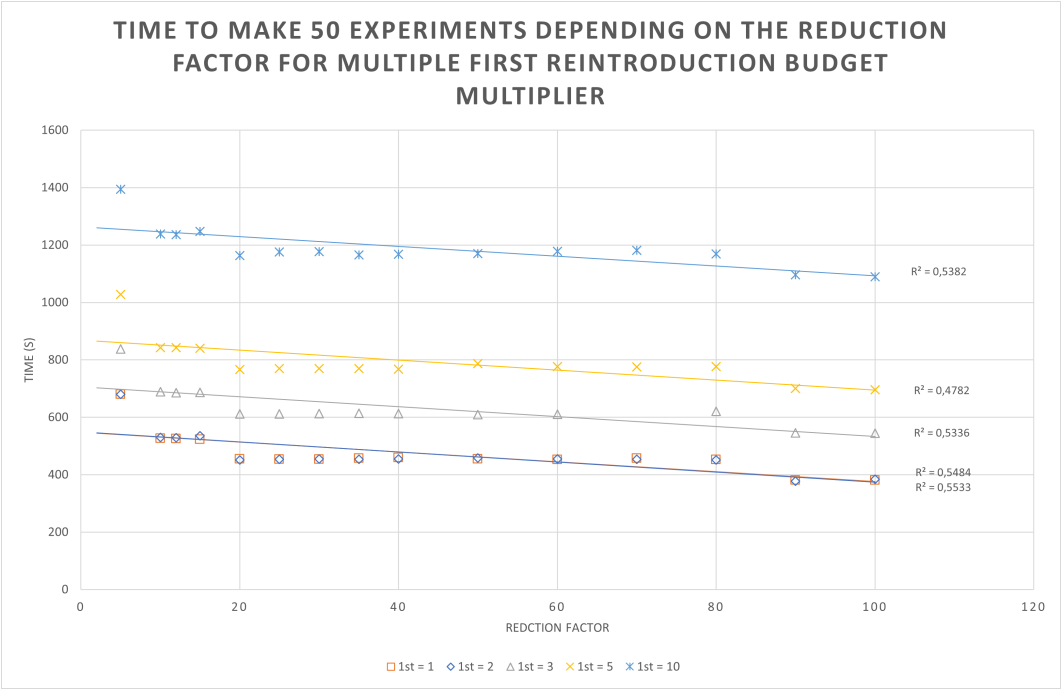

Figure 48: Gossip model time with first budget iteration improvement

It is also important to analyse the execution duration of the first Smart Sampling algorithm improvement. The graphics above illustrate the relationship between the reduction factors and the corresponding 50 execution duration

for each budget multiplier (1, 2, 3, 5, and 10).

We observe that, in most cases, the first iteration budget multiplier has minimal impact on the execution duration. Beside a more important deviation in the Gossip executions times, the duration remains relatively consistent across different reduction factors, regardless of the first iteration budget multiplier used. This finding aligns with the results discussed in Section 5, where we investigated the impact of the first iteration budget modification on the algorithm's effectiveness and efficiency. However, it is worth noting that a slight impact on the execution duration can be observed when using a first iteration budget multiplier of 10. In this case, there is a small increase in the execution duration compared to the other multipliers. Nonetheless, this increase is relatively modest and does not significantly affect the overall performance of the algorithm.

Overall, for the first iteration budget improvement, the initial observation conducted in section 5 remains consistent: increasing the budget at the first iteration leads to improved results compared to the original experimentation without such an adjustment. Evermore, the consistency in execution duration across various reduction factors, suggests that the algorithm's efficiency is not heavily influenced by the first iteration budget.

Lastly, let's analyse the result of the second Smart Sampling algorithm improvement which is to introduce random schedulers at different steps of the algorithm execution. The graph bellow illustrates the percentage of correct estimation for different reduction factors when applying this modification.

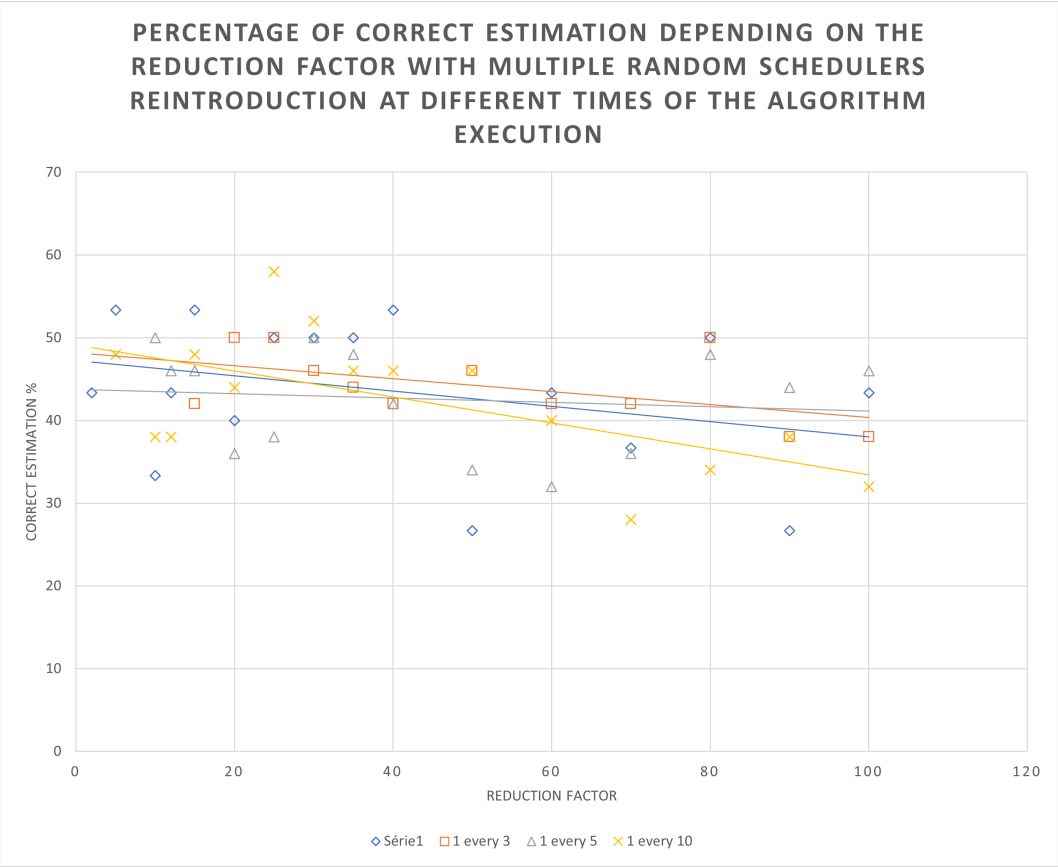

Figure 49: WLAN model with random reintroduction improvement

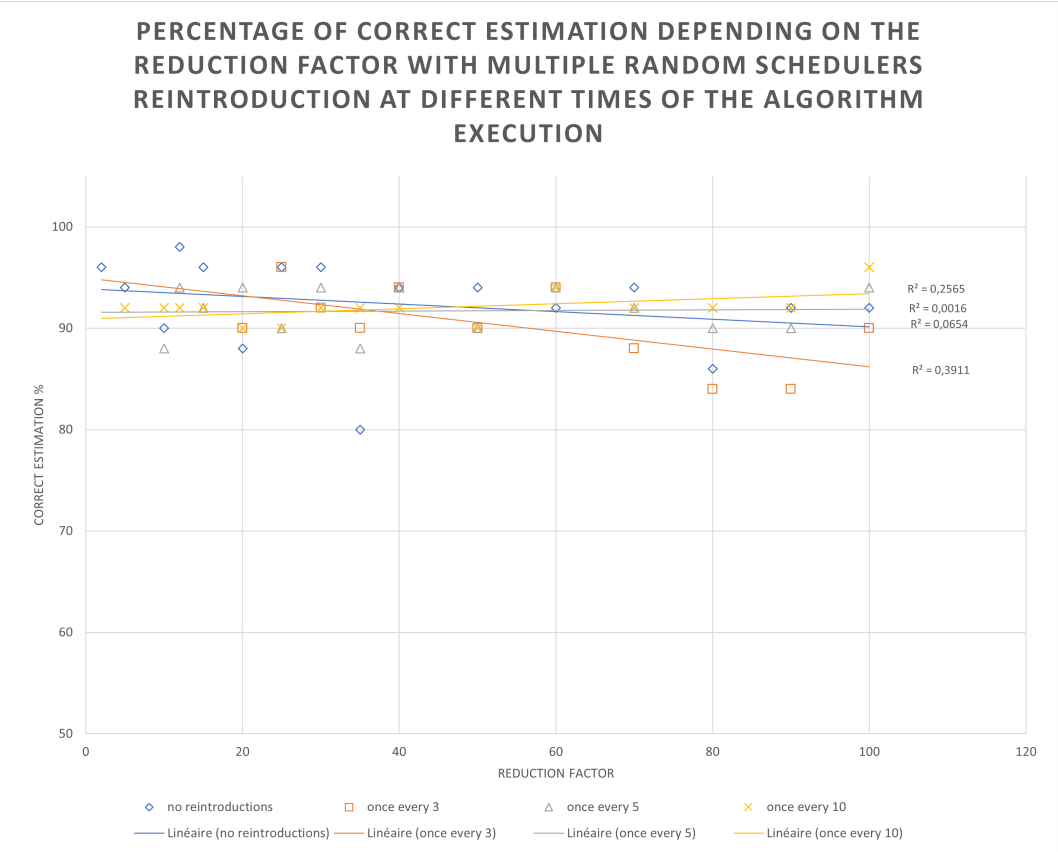

Figure 50: Gossip model with random reintroduction improvement

Upon examining the graphics ,We see that the introduction of multiple random schedulers at different points in the algorithm's execution does not have a significant impact on the estimation results. The percentages of correct estimation remain consistent across the various reduction factors, regardless of the timing of the scheduler reintroduction. This finding aligns with the results obtained in Section 5, where we investigated the behaviour of the second improvement of the Smart Sampling algorithm.

In conclusion, this section presented the evaluation of the modified Smart Sampling algorithm and its improvements using two use case models, WLAN and Gossip. We observed that the modified Smart Sampling algorithm successfully addressed the issue of drops encountered in the original algorithm. The elimination of these drops is a significant improvement as it ensures more consistent and reliable estimation results across different reduction factors. Furthermore, the first improvement, which involved allocating more budget at the first iteration, yielded the expected impact on the algorithm's ability to make correct estimations. The experimental results showed that increasing the budget of the initial iteration resulted in better estimation performance, as evidenced by higher percentages of correct estimations. Importantly, the

increase in computation duration due to this modification remained relatively low, indicating a favourable trade-off between accuracy and efficiency. On the other hand, the second improvement, which introduced random scheduler reintroduction, did not have a discernible impact on the estimation results. The experimental findings indicated that this modification had no significant effect on the algorithm's ability to make correct estimations. This result aligns with our earlier discussions, suggesting that the reintroduction of random schedulers at different times during the algorithm's execution does not significantly influence the correctness of the estimations.

7 Conclusion And Future Work

In conclusion, this master thesis undertook a comprehensive analysis and improvement of algorithms for statistical model checking, with a particular focus on the Smart Sampling algorithm. The goal was to gain insights into the algorithm's behaviour, identify areas for improvement, and enhance its effectiveness in estimating probabilities.

The initial phase of the research involved reproducing two existing experiments from a prior study [18] while varying the reduction factor parameter. The objective was to examine the impact of the reduction factor in the computed probabilities as well as the execution time. The findings revealed that modifying the reduction factor had no significant effect on the probabilities computed by the algorithm. However, it was evident that the execution time was influenced by this parameter.

Building upon these insights, a new model was created during the second part of this thesis to address the limitations encountered in the previous use cases. This new model exhibited a drop problem, wherein the algorithm's estimation accuracy experienced a sudden decrease for certain reduction factors. To overcome this issue, modifications were made to the behaviour of the Smart Sampling algorithm

Through meticulous analysis and experimentation, the modified Smart Sampling algorithm successfully eliminated the drop problem. The algorithm's performance was notably improved, resulting in more consistent and reliable estimation results across different reduction factors. This achievement demonstrates the efficacy of the proposed modifications in enhancing the algorithm's estimation capabilities

The third part of this thesis delved into the pursuit of further improvements to the Smart Sampling algorithm. Recognising the significance of the

initial iteration in shaping the algorithm's behaviour, we sought to enhance its performance by introducing modifications. Two specific improvements were developed and rigorously tested.

The first improvement centred around allocating more budget at the first iteration. This modification was motivated by the realisation that the initial step plays a pivotal role in the algorithm's overall performance. By reducing noise and refining the estimation process from the outset, we aimed to achieve more accurate probability computations. The results of this improvement were highly promising, demonstrating a clear positive correlation between increased budget allocation in the first iteration and the algorithm's ability to produce precise estimations. Importantly, the associated increase in computation time was minimal, making this modification a valuable enhancement to the Smart Sampling algorithm

The second improvement involved the reintroduction of schedulers at different stages of the algorithm's execution. The objective behind this modification was to enhance the exploration of the solution space, allowing for a more comprehensive search for optimal estimations. However, the results of this modification did not yield the desired outcomes. We discovered that introducing random schedulers at different steps had no discernible effect on the algorithm's capability to generate accurate estimations.

To validate the effectiveness of the modifications and improvements made to the Smart Sampling algorithm, we conducted extensive testing on the initial use cases of WLAN and Gossip models. The aim was to assess whether the observed enhancements would consistently yield improved results across different scenarios.

Remarkably, the results obtained from testing the modified algorithm and the two introduced improvements to the WLAN and Gossip models were consistently aligned with our expectations. The modifications successfully addressed the limitations encountered in the initial use cases, demonstrating their effectiveness in mitigating the drop problem and improving estimation accuracy. This outcome serves as a strong validation of the proposed improvements and modifications, as their positive impact transcended the specific context of the initial model.

In conclusion, this master thesis has not only contributed to the understanding and improvement of the Smart Sampling algorithm but has also successfully validated the proposed modifications and enhancements through comprehensive testing. By addressing the limitations encountered in the initial use cases and consistently delivering improved results across different scenarios, the modifications have demonstrated their effectiveness in enhancing estimation accu-

racy and mitigating challenges such as the drop problem. This validation solidifies the value and applicability of the modified Smart Sampling algorithm in statistical model checking and opens avenues for further research and applications in diverse domains.

As with any research endeavour, this thesis opens up new avenues for future exploration. While the modifications introduced in this study have yielded positive outcomes, there is still ample room for further investigation. Future research could explore alternative modifications to the Smart Sampling algorithm like for example an intelligent generation of scheduler bases on genetic algorithms, delve into the impact of different parameters, or investigate the algorithm's efficacy in diverse domains and real-world scenarios. By continually refining and expanding our understanding of statistical model checking algorithms, we can pave the way for enhanced verification techniques and contribute to the advancement of reliable and secure systems

Appendices

A Smart Sampling algorithm

```
181     private void runProba(int st, int req) throws PlasmaSimulatorException, PlasmaCheckerException, PlasmaStopOrder {
182         long start1 = System.currentTimeMillis();
183         long start2;
184         long time;
185         iterations[st][req] = 0;
186 
187         int scheduler, nbTokeep, batchUpdate;
188         int nbIterMax = (int) (Math.log(simBudget)/Math.log(factor));
189 
190         listener.publishResults(nodeURI, getResults());
191         Random schedulerRng = new Random();
192 
193         // First iteration ------------------------------------------------------------------------------------------
194         // Roughly estimate the maximum (resp. minimum) probability ps
195         int totalToDo = (int)(Math.sqrt(simBudget));
196         while (probability[st][req]==0 && totalToDo <= simBudget) {
197             start2 = System.currentTimeMillis();
198             iterations[st][req]++;
199             nbSchedulers = simBudget/totalToDo;
200             probability[st][req] = maximum ? 0 : 1;
201             logger.info("Iteration " + iterations[st][req] + ": " + totalToDo + " simulations; " + nbSchedulers + " schedulers;");
202             nbIterMax++;
203             batchUpdate = Math.max((int) (nbSchedulers*0.01),1);
204 
205             //Iterate over scheduler
206             for(scheduler=0; scheduler<nbSchedulers; scheduler++){
207                 // Initialize a new scheduler/seed
208                 long seed = schedulerRng.nextLong();
209 
210                 // Check the scheduler
211                 double proba = checkSchedulerProba(totalToDo, requirements.get(req), seed);
212 
213                 if (!maximum && proba == 0) { // The minimum probability is 0.
214                     probability[st][req] = 0;
215                     break;
216                 }
217 
218                 // Update the maximum probability
219                 if((maximum && proba > probability[st][req]) || (!maximum && proba < probability[st][req])) {
220                     probability[st][req] = proba;
221                 }
222 
223                 // Update progress
224                 if(scheduler%batchUpdate == 0){
225                     double percent = ((double)nbTotalSimulations[st][req])/(nbIterMax*simBudget+schedBudget);
226                     double remaining = (System.currentTimeMillis()-start1)/percent-(System.currentTimeMillis()-start1);
227                     listener.notifyProgress((int)(100*percent));
228                     listener.notifyTimeRemaining((long)remaining);
229                 }
230             }
231             time = System.currentTimeMillis() - start2;
232             logger.info("Iteration " + iterations[st][req] + ": " + simBudget + " simulations in " + time + " ms ");
233             listener.publishResults(nodeURI, getResults());
234             totalToDo = totalToDo*2;
235         }
236         if (probability[st][req]==0)
237             return;
```

Figure 51: Smart Sampling 1

```
239         // Iteration 2: initialize scheduler seeds ----------------------------------------------------------------------------------
240         start2 = System.currentTimeMillis();
241         iterations[st][req]++;
242         totalToDo = (int)(1/probability[st][req]);
243         nbSchedulers = schedBudget/totalToDo;
244         nbTokeep = Math.min(Math.max(nbSchedulers/factor,1), simBudget);
245
246         logger.info("Iteration " + iterations[st][req] + " : " + totalToDo + " simulations; " + nbSchedulers + " schedulers;");
247         batchUpdate = Math.max((int) (nbSchedulers*0.01),1);
248
249         TreeMap<Double, List<Long>> bestSchedulers = new TreeMap<~>();
250         int bestSchedulersSize = 0;
251         for(scheduler=0; scheduler<nbSchedulers; scheduler++){...}
269         // Keep the best schedulers according to the factor
270         Map.Entry<Double,List<Long>> best = maximum ? bestSchedulers.lastEntry() : bestSchedulers.firstEntry();
271         probability[st][req] = best.getKey();
272
273         nbSchedulers = nbTokeep;
274         long[] schedulerSeed = new long[nbSchedulers];
275         sortBestSchedulers(bestSchedulers,schedulerSeed);
276
277         // Notify new results
278         time = System.currentTimeMillis() - start2;
279         logger.info("Iteration " + iterations[st][req] + " : " + schedBudget + " simulations in " + time + " ms ");
280         listener.publishResults(nodeURI, getResults());
281
```

Figure 52: Smart Sampling 2

```
    // Iteration 3 to n ------------------------------------------------------------------------------------------------
    // Compute the probability of each of the schedulers. Keep the greater half (resp. the lower half)
    int chernoffBound = (int) Math.ceil(-Math.log(1-Math.exp(Math.log(1-delta)/nbSchedulers))/(2* Math.pow(epsilon, 2)));
    totalToDo = Math.min(simBudget/nbSchedulers,chernoffBound);
    while(totalToDo < chernoffBound) {
        start2 = System.currentTimeMillis();
        iterations[st][req]++;

        logger.info("Iteration " + iterations[st][req] + " : " + totalToDo + " simulations; " + nbSchedulers + " schedulers;");
        batchUpdate = Math.max((int) (nbSchedulers*0.01),1);

        bestSchedulers = new TreeMap<Double, List<Long>>();
        bestSchedulersSize = 0;
        nbTokeep = Math.max(nbSchedulers/factor, 1);
        for(scheduler=0; scheduler<nbSchedulers; scheduler++){
            nbTotalSimulations[st][req] += totalToDo;

            // Check the scheduler
            double val = checkSchedulerProba(totalToDo, requirements.get(req), schedulerSeed[scheduler]);

            // Update the map
            bestSchedulersSize = updateBestSchedulers(val, schedulerSeed[scheduler], bestSchedulers, bestSchedulersSize, nbTokeep);

            // Update progress
            if(scheduler%batchUpdate == 0){
                double percent = ((double)nbTotalSimulations[st][req])/(nbIterMax*simBudget+schedBudget);
                double remaining = (System.currentTimeMillis()-start1)/percent-(System.currentTimeMillis()-start1);
                listener.notifyProgress((int)(100*percent));
                listener.notifyTimeRemaining((long)remaining);
            }
        }
        // Keep the best schedulers according to the factor
        best = maximum ? bestSchedulers.lastEntry() : bestSchedulers.firstEntry();
        probability[st][req] = best.getKey();
        logger.info("minimum probability: " + bestSchedulers.firstEntry().getKey());
        logger.info("maximum probability: " + bestSchedulers.lastEntry().getKey());

        nbSchedulers = nbTokeep;
        schedulerSeed = new long[nbSchedulers];
        sortBestSchedulers(bestSchedulers,schedulerSeed);

        chernoffBound = (int)   Math.ceil(-Math.log(1-Math.exp(Math.log(1-delta)/nbSchedulers))/(2* Math.pow(epsilon, 2)));
        totalToDo = Math.min(simBudget/nbSchedulers,chernoffBound);

        // Notify new results
        time = System.currentTimeMillis() - start2;
        logger.info("Iteration " + iterations[st][req] + " : " + simBudget + " simulations in " + time + " ms ");

        listener.publishResults(nodeURI, getResults());
    }
}
```

Figure 53: Smart Sampling 3

B Multi-threads tester

```python
import os, csv, time
import concurrent.futures
from subprocess import Popen, PIPE, STDOUT

def launch_plasmacli(rf):
    p = Popen(r"""plasmacli.bat launch --conf configs\plasmalab.conf -m wlan_5.rml:rml -r collisions.bltl:bltl -a smartsampling
    -A"Epsilon"=0.1 -A"Delta"=0.03 -A"Budget"=1800 -A"Maximum"=true -A"Reduction Factor"=""" +
              rf, shell=True, stdin=PIPE, stdout=PIPE, stderr=STDOUT)
    proba = p.communicate()[0].decode().split('|')[7].replace('.', ',')
    return proba

os.chdir(r"C:\Users\Firmin\OneDrive\Unif\TFE\PlasmaLab\Executable\plasmalab-1.4.4")
result = []
lst = ['2', '5', '10', '20', '30', '40' , '50', '70', '80', '100']
for rf in ['2', '5', '10', '15', '20', '25', '30', '40' , '50', '60', '70', '80', '100']:
    rf_data = []
    f = open(r"C:\Users\Firmin\OneDrive\Unif\TFE\PlasmaLab\Executable\plasmalab-1.4.4\result" + rf + ".csv", 'w')
    writer= csv.writer(f, delimiter=' ',lineterminator='\n',)
    t1 = time.time()


    # multithreading
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = [executor.submit(launch_plasmacli, rf) for _ in range(1)]
        concurrent.futures.wait(futures)

        for future in futures:
            proba = future.result()

            rf_data.append(proba)  # 7
            writer.writerow([proba])  # 7


    comp_time = time.time() - t1
    print('-> rf:', rf , 'time:', comp_time)
    f.close()

    rf_data.insert(0, str(comp_time).replace('.', ','))
    result.append(rf_data)


with open(r"C:\Users\Firmin\OneDrive\Unif\TFE\PlasmaLab\Executable\plasmalab-1.4.4\result.csv", 'w', newline='') as file:
    writer = csv.writer(file, delimiter=';')
    for row in zip(*result):
        row = [elem.strip() for elem in row]
        writer.writerow(row)
```

Figure 54: Multi-threads tester

References

[1] Thamilselvam B K, Subrahmanyam Kalyanasundaram, Shubham Parmar, and M. Rao. *Statistical Model Checking for Traffic Models*, pages 17–33. 11 2021.

[2] Benedek Horváth, Bence Graics, Ákos Hajdu, Zoltan Micskei, Vince Molnár, István Ráth, Luigi Andolfato, Ivan Gomes, and Robert Karban. Model checking as a service: towards pragmatic hidden formal methods. pages 1–5, 10 2020.

[3] Dirk Beyer and Andreas Podelski. *Software Model Checking: 20 Years and Beyond*, pages 554–582. 12 2022.

[4] Edmund M. Clarke and Qinsi Wang. 25 years of model checking. In *Ershov Memorial Conference*, 2014.

[5] Armin Biere. Tutorial on model checking: Modelling and verification in computer science. In *Algebraic Biology*, 2008.

[6] Dirk Beyer and Andreas Podelski. *Software Model Checking: 20 Years and Beyond*, pages 554–582. 12 2022.

[7] Bin Yu, Zhenhua Duan, and Cong Tian. Bounded model checking of traffic light control system. *Electronic Notes in Theoretical Computer Science*, 309, 12 2014.

[8] Edmund Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. 01 2012.

[9] S. Demri, F. Laroussinie, and Ph. Schnoebelen. A parametric analysis of the state-explosion problem in model checking. *Journal of Computer and System Sciences*, 72(4):547–575, 2006.

[10] Kim Larsen and Axel Legay. Statistical model checking past, present, and future. pages 135–142, 10 2014.

[11] Gul A. Agha and Karl Palmskog. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28:1 – 39, 2018.

[12] Angela Pappagallo, Annalisa Massini, and Enrico Tronci. Monte carlo based statistical model checking of cyber-physical systems: A review. *Inf.*, 11:588, 2020.

[13] Axel Legay, Anna Lukina, Louis-Marie Traonouez, Junxing Yang, Scott A. Smolka, and Radu Grosu. Statistical model checking. In *Computing and Software Science*, 2019.

[14] Edmund Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. pages 1–12, 10 2011.

[15] Souymodip Chakraborty, Joost-Pieter Katoen, Falak Sher, and Martin Strelec. Modelling and statistical model checking of a microgrid. *International Journal on Software Tools for Technology Transfer*, 17, 01 2014.

[16] Fang Li, Frederike Jörg, Xinyu Li, and Talitha Feenstra. A promising approach to optimizing sequential treatment decisions for depression: Markov decision process. *PharmacoEconomics*, 40, 09 2022.

[17] Dan Calderone and S. Sastry. Markov decision process routing games. pages 273–279, 04 2017.

[18] Pedro D'Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart sampling for lightweight verification of markov decision processes. *International Journal on Software Tools for Technology Transfer*, 17, 09 2014.

[19] Olivier Spanjaard and Paul Weng. Markov decision processes with functional rewards. In Sheela Ramanna, Pawan Lingras, Chattrakul Sombattheera, and Aneesh Krishna, editors, *Multi-disciplinary Trends in Artificial Intelligence*, pages 269–280, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[20] Amel Omer and Dereje Woldegebreal. *Review of Markov Chain and Its Applications in Telecommunication Systems*, pages 366–385. 05 2022.

[21] Galin L. Jones and Qian Qin. Markov chain monte carlo in practice. *Annual review of public health*, 2021.

[22] Christian Colombo and Gordon Pace. *Linear Temporal Logic*, pages 109–122. 07 2022.

[23] Cyrille Jegourel, Axel Legay, and Sean Sedwards. A platform for high performance statistical model checking – plasma. pages 498–503, 03 2012.

[24] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. Plasma-lab: A flexible, distributable statistical model checking library. volume 8054, pages 160–164, 08 2013.

[25] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Plasma lab: A modular statistical model checking platform. volume 9952, pages 77–93, 10 2016.

[26] Jeanne Jones, Erol Kalkan, Chris Stephens, and Peter Ng. Prism software: Processing and review interface for strong-motion data. *Seismological Research Letters*, 03 2017.

[27] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In H. Hermanns and R. Segala, editors, *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.

[28] M. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in prism. *ACM SIGMETRICS Performance Evaluation Review*, 36(3):17–22, 2008.

[29] Prism model checker - wireless lan. http://www.prismmodelchecker.org/casestudies/wlan.php.

[30] Prism model checker - gossip. http://www.prismmodelchecker.org/casestudies/gossip.php.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**