Improvements in Statistical Model Checking: Estimation, Optimization, Parallelization

Maxime Parmentier

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

August 2025

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Charles **Pecheur** (Advisor)	UCLouvain/ICTEAM, Belgium
Pr. Peter **Van Roy** (President)	UCLouvain/ICTEAM, Belgium
Pr. Kim **Mens** (Secretary)	UCLouvain/ICTEAM, Belgium
Pr. Pierre-Yves **Schobbens**	UNamur, Belgium
Pr. Maxime **Cordy**	University of Luxembourg, Luxembourg
Pr. Arnd **Hartmanns**	University of Twente, Netherlands

Improvements in Statistical Model Checking: Estimation, Optimization, Parallelization

by Maxime Parmentier

ICTEAM
Université catholique de Louvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work was supported by the F.R.S-FNRS and the UCLouvain.

何でもは知らないわよ。
知ってることだけ。

西尾維新

Nothing in life is to be feared,
it is only to be understood.
Now is the time to understand more,
so that we may fear less.

MARIE CURIE

Manger sur l'herbe, dépêchez vous,
un jour l'herbe mangera sur vous.

JACQUES PRÉVERT

Preamble

By the middle of the 21st century, robots will make all forms of physical work obsolete, we will move around in flying self-driving cars and automatic farms powered by gigantic arrays of solar panels will easily provide enough food to eradicate famines worldwide once and for all.

Or so people (allegedly) thought at the start of the 20th century.

While we are still far from such a utopian goal, technological innovation can still provide hope for the future nonetheless. After all, we already have demining robots that allow us to clean up the leftovers of the madness of war safely [DOK-ING]. The feats of modern electric vehicles can make us dream of a not-so-distant future where the sad reality that more than 20,000 people die each year because of road accidents in Europe alone [Eurostat] will be nothing more than a bad memory. And the International Energy Agency has recently concluded that with the incredible increase in efficiency of solar panels of the last 20 years, solar energy has now generally become the cheapest option when building new power plants [IEA].

However, all those fabulous inventions and all that game-changing infrastructure are complex systems. They are difficult to design, assemble and even run properly. Those cyber-physical systems do not just involve mechanical parts like the revolutionary engines of the past and they do not just rely on the marvels of electronics like the numerous computer programs we use daily: they combine the strengths of specific hardware and software to accomplish multi-layered tasks for which it is often necessary to interact with the environment.

These tasks must be performed successfully, but also safely and at a minimal cost. A demining robot not detecting all mines would be a hazard in itself, the idea of an autonomous bus hitting people on a pedestrian crossing just to guarantee that it will arrive on time is terrifying, and a solar power plant not producing energy because of a bug that prevents the covers of the solar panels to be opened in the morning would result in power outages for entire cities. Ensuring that a newly designed or built cyber-physical system behaves as it should is therefore critical.

The research to create new frameworks and techniques for the verification of complex software and large systems has been progressing for multiple

decades at this point, but many difficulties remain. In particular, because of the always larger size and always greater number of interactive components of cyber-physical systems, most strategies that aim to do better than basic tests or code reviewing and automate the verification process do not scale well [Cla+11]. Alternative strategies exploiting stochastic algorithms have however been developed to (partially) solve that problem, such as the techniques of statistical model checking [LL16].

In this work, we not only propose new theoretical results and suggest improvements for existing algorithms, we also illustrate how the methods of statistical model checking can be enhanced and expanded to new frameworks with multiple case studies. More specifically, the contributions of this thesis are the following:

- *An in-depth theoretical analysis of a large class of estimation algorithms, with a novel result of optimality*

  We generalize the construction of stopping criteria for adaptive stopping algorithms that rely on concentration inequalities. We prove a theoretical bound on the efficiency of those algorithms, and we show that this bound can be reached. We apply those results to generate an adaptive stopping algorithm designed to tackle the rare event problem.

- *An improved version of the Smart Sampling algorithm, a lightweight yet effective method for the optimization of schedulers for Markov decision processes*

  We fix a fundamental issue with the Smart Sampling algorithm which could lead to faulty results. We improve its speed and efficiency by up to multiple hundreds of percents and we enhance the stability of its outputs. We also explore the ideas of scheduler synthesis and genetic algorithms for further improvements.

- *A large-scale implementation of a parallelization strategy to improve the scalability of statistical model checking approaches*

  We detail the theoretical conditions that must be met for such a parallelization of statistical model checking algorithms to be carried out properly and without introducing bias. We explain how we proceeded in practice to perform verification tasks which would have been out of reach otherwise. We describe the results of our experiments and highlight the benefits of parallelization for statistical model checking.

- *A new application of statistical model checking for the framework of temporal networks*

  We show how statistical model checking can be exploited in an original setting by illustrating how classic statistical model checking algorithms can easily be applied for task scheduling. In particular, we propose new methods to analyze the brittleness of temporal networks.

The thesis is organized as follows. Chapter 1 provides an introduction and a broad overview of the field of statistical model checking. In Chapter 2, we list and define all the preliminary notions which are necessary for the understanding of this thesis. The main chapters of the thesis are Chapter 3, Chapter 4, Chapter 5 and Chapter 6. Chapter 3 and Chapter 4 can be grouped together as more theoretical: they are dedicated to estimation and optimization algorithms for statistical model checking, respectively. Chapter 3 delves into the subject of adaptive stopping algorithms, while Chapter 4 focuses on the Smart Sampling optimization algorithm and the various ways to build upon it. Chapter 5 and Chapter 6 are instead centered around the actual implementation of statistical model checking strategies for the verification of cyber-physical systems. Chapter 5 revolves around the concept of parallelization and how it can be exploited to improve the scalability of statistical model checking techniques, and Chapter 6 presents how statistical model checking can be used to study the robustness of temporal networks. Finally, Chapter 7 offers a conclusion for this thesis, and includes additional perspectives for future work.

Bibliographic notes

Conference Publications

1. M. Parmentier, A. Legay, and F. Chenoy. "Optimized Smart Sampling". In: *International Conference on Bridging the Gap between AI and Reality*. Springer. 2023, pp. 171–187.

2. M. Parmentier and A. Legay. "Adaptive Stopping Algorithms Based on Concentration Inequalities". In: *International Conference on Bridging the Gap between AI and Reality*. Springer. 2024, pp. 171–187.

Journal Publications

1. L. Picchiami, M. Parmentier, A. Legay, T. Mancini, and E. Tronci. "Scaling up Statistical Model Checking of Cyber-Physical Systems via algorithm ensemble and parallel simulations over HPC infrastructures". In: *Journal of Systems and Software* (2024), p. 112238.

2. T. Vaquero, S. Chien, J. Agrawal, M. Saint-Guillain, and M. Parmentier. "Property-Based Brittleness Analysis of Temporal Networks". In: *Journal of Aerospace Information Systems* 20.7 (2023), pp. 398–417.

Acknowledgments

Une thèse de doctorat est un processus extrêmement long et qui requiert un investissement hors norme. Le périple commence avec un sentier qui dans mon cas m'était complétement inconnu (la vérification statistique de modèles), dont le tracé a été dessiné par d'autres personnes depuis devenues chercheurs de renom et qui malgré cela n'était très rapidement plus fléché. L'isolement, physique (que la pandémie a amplifié) et mental, l'impression de tourner en rond encore et encore, la nécessité d'orienter l'intégralité de son existence autour de son exploration, et le fait que ce sentier se transforme tôt ou tard en rebord escarpé de montagne en territoire volcanique font qu'à certains moments, on finit par se demander si on a même jamais été capable de marcher.

Mais tout cela signifie que je suis d'autant plus fier, satisfait et heureux d'arriver au bout du chemin.

Néanmoins, malgré tous les efforts que j'ai dû fournir, malgré tous les doutes que je suis parvenu à surmonter, j'ai tant été aidé et soutenu que je pense qu'une partie du mérite et des applaudissement revient légitimement à d'autres personnes, et je souhaite les remercier.

Merci à Axel Legay pour m'avoir offert un cadre fantastique pour la réalisation de ce doctorat. Merci de m'avoir fait découvrir le sujet de la vérification statistique de modèles, sujet qui m'a permis d'exploiter à la fois mon intérêt pour l'informatique et mon adoration pour les mathématiques et la logique.

Merci à Charles Pecheur qui en plus d'avoir gentiment accepté d'être également mon promoteur en début de thèse afin de garantir un bon déroulement administratif, a généreusement pris la relève pour la toute fin de celle-ci.

Merci à Kim Mens et Pierre-Yves Schobbens pour avoir fait partie de mon comité d'accompagnement, merci à Maxime Cordy et Arnd Hartmanns pour avoir accepté de faire partie de mon jury, et merci à Peter Van Roy pour avoir accepté de présider celui-ci.

Merci également à tous les membres du département avec qui j'ai interagi durant ces nombreuses années de thèse. En particulier, merci à Yves Deville pour m'avoir fait rencontrer Axel Legay et pour m'avoir donné l'occasion d'enseigner la calculabilité. Merci à Vanessa Maons pour son accueil, sa disponibilité et sa gentillesse. Merci à Eduard Baranov pour m'avoir parfois servi de mentor. Merci à tous les autres doctorants avec qui j'ai pu discuter et échanger des photos de lamas.

Merci à Charles-Henry pour avoir été mon compagnon de thèse dès le début, pour nos discussions très intéressantes, pour tous nos échanges d'entraide et de compréhension mutuelle, et pour m'avoir finalement montré l'exemple.

Merci à mes amis avec qui je me réunis encore régulièrement pour lancer des dés et raconter des histoires pleines de magie et de dragons, et ce malgré nos vies qui évoluent et mes performances en tant que maître du jeu qui ont parfois tout au plus été passables à cause de ma fatigue ou de mon manque de temps pour préparer les séances. Merci à Joachim pour notre amitié qui a tant de valeur à mes yeux, pour m'avoir précédé et m'avoir dès lors donné envie de t'égaler.

Merci à Comète, dont les ronronnements endormis sont toujours aussi apaisants. Même si tu m'as rendu maintes fois la tâche plus difficile en exigeant sans cesse plus de caresses et en t'allongeant sur mes bras, je te promets que je ne t'en veux pas.

Merci à mon frère Tom, avec qui je continue de garder un lien fraternel inébranlable et plein d'amour malgré nos différentes personnalités et le fait que nous ne nous voyons plus souvent. Sans toi, je suis certain que je serais devenu une personne moins douée, mais aussi quelqu'un d'encore plus replié sur soi-même.

Enfin, merci à mes parents. Je pourrais lister les innombrables façons très concrètes par lesquelles vous m'avez aidé durant toutes ces années de thèse (et durant toutes les années qui ont précédé), mais toutes ces formes de soutien (très appréciées et que je n'oublierai jamais) ne sont pas ce à quoi je pense en écrivant ces mots. Ce que j'ai en tête est bien plus fondamental. Merci, merci, merci.

Contents

Introduction 1

In this chapter, we define the subject of this thesis and provide a broad overview of the literature related to the verification of complex systems with statistical model checking.

1.1 The Verification of Complex Software and Large Systems

Technology is not what it used to be: attaching a pointy rock on top of a stick is not enough anymore to revolutionize our lives. Technological innovation has become so advanced and intricate that it requires entire teams of specialists and years of research and development to come up with new solutions to the challenges of the modern world. These solutions can take many forms, ranging from structural changes of infrastructure to useful accessories.

Specifically, computer systems occupy an increasingly predominant position in our daily lives. Those systems find themselves embedded in strategic applications such as connected homes ([DMC18; ASB20]), medical robots ([Dey+18]), space rovers (see Chapter 6) or autonomous vehicles ([Kim+13; Che+17; Gil+19]). All of those examples can be labeled as cyber-physical systems. Cyber-physical system has become a polyseme over time, and can describe completely unrelated architectures in different contexts [Wol09; BG11; RDK16; Let+17; Zan17]. In general, a **Cyber-Physical Systems (CPS)** could be broadly defined as any system made up of both physical and software components, designed as a whole to achieve one specific goal. It is to be understood as the whole set of a computing unit, its software, the mechanical parts (sensors, actuators, engines, ...) of a system *and their interactions*.

As these new complex systems are multifaceted, multilayered, possibly dematerialized, dynamic and heterogeneous, have spatial features and often interact with their environment to perform critical missions, they have become more and more difficult and costly to design and produce.

Additionally, we expect more from technology than ever: safety, efficiency, security, adaptability... Sometimes, those requirements are even interdependent. Both for the engineers' and the users' sake, it is therefore extremely important to have robust theories and powerful tools to check as early as possible whether a new piece of elaborate software or the plans for a massive new infrastructure meet their design requirements.

So, it should not surprise anyone that the *verification* of complex systems, in particular at *design time*, has been an ongoing topic of research for several decades already.

(Unit) Testing is the most widespread approach to check the correctness of a system [Bro+05] and can occasionally be very useful [AGP19; God20], but it suffers from many limitations. The number of tests are almost always very low with respect to the size of the exploration space, because of time and resource constraints. Designing relevant tests depends on *a priori* knowledge, which can be limited or insufficient, and make it for example difficult to identify cyber security issues [Lan+18]. The outcomes of those tests can be difficult to interpret and do not always provide a path to a solution when a problem is discovered. Furthermore, these techniques do not quantify the degree of confidence which can be associated with their conclusions [Lin+12], which means they have no value for contexts where certificates of validity are required [Dup+22].

The two other practical strategies for the verification of software and systems are code analysis and prototyping. Unfortunately, code analysis does not scale at all and is prone to human errors, while prototyping is almost always prohibitively expensive or even unfeasible.

An alternative is formal methods [CW96; BK08]. Formal methods exploit a (mathematical) representation of a system to derive information about its characteristics and possible realizations. For instance, if an executable model can be built, one can go through every possible specification or execution with simulations and check if all of them meet the system requirements. This is called **model checking (MC)**.

While formal methods have the drawback that they require the generation of a model of the system which needs to be verified, they may appear like the perfect solution at first glance. They can be automatized, offer absolute certainty, and the mathematical frameworks they are built upon allow to prove impactful theoretical results. Moreover, with popular models such as (extensions of) transition systems, they also make it possible to model and analyze the failures of the system, the evolution of its parameters or its interactions with the environment [Cla+18b].

However, because the size of a model grows exponentially with the number of variables which are used for the representation, the so-called **state explosion problem** [Cla+11], complete methods of model checking are simply not exploitable for real-case applications and industry-scale CPSs. This issue is only amplified when considering continuous state and action spaces.

Thankfully, there exists an excellent trade-off between the usability of testing protocols and the completeness of formal methods: **statistical model checking**.

1.2 Statistical Model Checking

By applying the ideas of formal methods to a well-generated sample, it is possible in many situations to provide strong and sufficient statistical insight without having to perform an exhaustive analysis of the system. This is the core idea behind statistical model checking. **Statistical Model Checking (SMC)** is the set of all methods which combine modelization, simulation and statistics for the verification and analysis of (complex) systems. SMC can be regarded as a subset of MC, the subset of techniques that trade the absolute certainty of completeness with the much more easily achievable probabilistic guarantees of stochastic algorithms.

By giving up on an exhaustive search of the exploration space, SMC partially solves the state-explosion problem, although it would be more correct to say that it temporarily avoids it. For large enough models and difficult requirements, SMC still struggles in practice. A lot of the research in the field precisely aims at pushing always further the scalability of SMC methods, for example by minimizing the number of simulations that needs to be performed to obtain the necessary statistical guarantees (see Chapter 3) or by developing new strategies to implement known algorithms (see Chapter 5).

SMC is at least three decades old at this point, and yet the discipline is still growing [SVA05; LDB10; LL14; LL16; Leg+19b; AP18; YS06; Leg+19a; LL20]. The development of new SMC methods specifically designed to verify CPSs has been particularly rich and diverse [CZ11; Zhe+15; PMT20b]. SMC has already been used to provide solutions for a large collection of real-world problems. A few examples are the validation of railway and aviation systems, medical and space components, and even privacy sensitive structures such as an information sharing platform for the healthcare system [Bar+21a; Bas+22a; Pai+20; Col+15; Bas+22b; Bar+21b].

In the next two sections, we describe the main categories of SMC algorithms [Pap+20; PMT20b; AP18; Rei+15], and provide a list of the numerous tools that have been built for the application of SMC [Pap+20; PMT20b; AP18; Bak+17].

1.2.1 Standard Statistical Model Checking Algorithms

SMC methods can be classified in function of three criteria: their system model formalism, their property formalism, and their goal.

Most of the mathematical representations for the systems are state-transition models. In many cases, they can be regarded as upgraded automata. Some of the most common ones are Markov chains and continuous time Markov chains [Bai+03; YS02], generalized semi Markov processes [SVA04; Whi80], generalized stochastic Petri nets [Bal+15], probabilistic timed automata [NPS13],

stochastic hybrid automata and stochastic timed automata [Dav+12], and Markov Decision Processes (MDPs) [LST14; Put14]. We describe in details the models that are relevant to this thesis in Section 2.2 and Section 2.3.

System requirements are generally formalized as properties of a logic whose semantics is built upon the associated class of system models. Most of those logics are modal extensions of propositional logic. Some of the most popular are basic linear temporal logic [Roz11], bounded linear temporal logic [Kam12; DAr+15b], metric temporal logic [Med+18], metric interval temporal logic [AFH96], computational tree logic [Cla+18c], probabilistic computational tree logic [HJ94; Bai98] and continuous stochastic logic [SVA05]. We describe in details the logic that is relevant to this thesis in Section 2.2.

Instead, we dedicate this section to the classification of SMC methods in function of their goal. We differentiate two categories: estimation algorithms and optimization algorithms.

The large majority of the existing SMC methods belong to the first category. It can be broken down further into three subcategories: generic estimation algorithms, probability estimation algorithms, and hypothesis testing.

Generic estimation algorithms are algorithms whose objective is to estimate the parameter of a system or a probability distribution. The foundational algorithm of SMC, that we briefly discuss in Section 2.1.3, is one of those algorithms. However, there exist many alternatives to the elementary Monte Carlo approaches: the confidence interval methods, the (O)AA algorithm [Dag+00; GS04; GS05] and the Bayesian statistical Estimation algorithm (which requires some *a priori* information about the system [ZPC13]) are good examples.

Probability estimation algorithms are possibly the most common and diverse algorithms of SMC. As illustrated in Section 1.2.1, the archetypal SMC method consists in a) build a model of the system, b) formalize the requirement, c) pick some values for the parameters of the SMC algorithm (precision and confidence), d) generate simulations of the systems, e) use those simulations to estimate the probability that the property holds for a random possible execution of the system. Generic estimation algorithms can theoretically be exploited to estimate probabilities as well, as those probabilities can be regarded as the expected values of Bernoulli distributions. However, with the assumption that the quantity to estimate is a probability, it is possible to design much more efficient algorithms, for example by relying on the Chernoff bound for probabilities [Hoe94]. Two examples of such algorithms are the Importance Sampling [JLS16] and the importance Splitting [JLS13] algorithms.

Hypothesis testing algorithms ultimately solve the same problems as generic and probability estimation algorithms, but they rely on the statistical framework of hypothesis testing. In practice, there exist different approaches [Rei+15], some (indirectly) relying on confidence interval methods, some on the Chow-

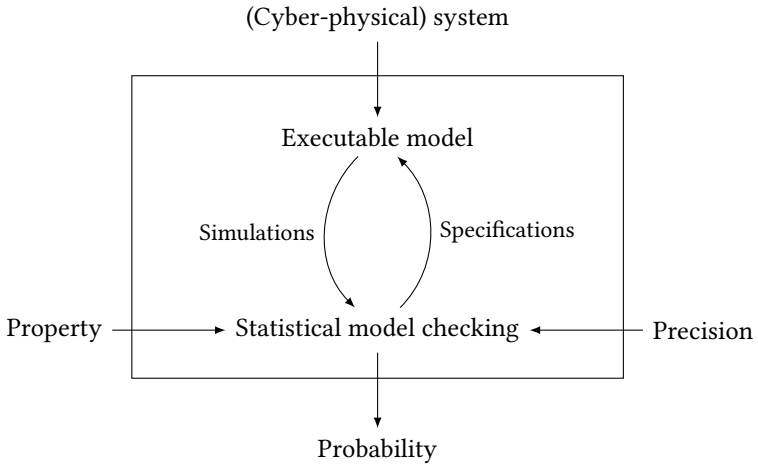


Figure 1.1: A schematic view of a typical statistical model checking strategy for the verification of a system requirement

Robbins test, some on the sequential probability ratio test of Wald, and some on the Single Sampling Strategy [You04].

Last but not least: SMC optimization algorithms. Those algorithms try to determine the best way to control the system so as to minimize or maximize that a requirement is verified. This is obviously much more difficult than simple estimation, and SMC optimization algorithms can only be built on top of efficient estimation algorithms. An example is the Smart Sampling algorithm we analyze and improve in Chapter 4.

1.2.2 Challenges of the Field

The fundametal challenge of SMC is the one that led to its creation: scalability. Even with sampling and statistical algorithms, real-world systems are so large and complex that even large high performance computing infrastructure can struggle to verify them within realistic time frames (see Chapter 5). A lot of the research done in the field tries to deal, directly or indirectly, with that limitation.

However, improving the scalability of existing methods of SMC is not the only way to expand the usability of SMC. Working on specific settings, such as the rare event problem (estimating probabilities very close to 0 or 1), creating new ways to model systems and their requirements, developping new tools for the actual implementation of SMC algorithms and focusing on special forms of properties for a new class of models are all valid strategies.

Figure 1.2: Summary and main contributions of the thesis

Challenge	Contribution	Thesis chapter
Rare event problem	A new estimation algorithm which minimizes the number of simulations	Chapter 3
System optimization	An improved version of Smart Sampling	Chapter 4
Developing better tools	The EAA algorithm and its implementation on a HPC infrastructure	Chapter 5
Enrichment of model formalization capabilities + Verification of specialized properties (safety, robustness, ...)	Application of SMC to the analysis of task brittleness for probabilistic temporal networks	Chapter 6

In this thesis, we have exploited all of those strategies. A summary of our contributions can be found in Figure 1.2.

1.2.3 Existing Tools

While the following list may not be exhaustive, we believe it should include most (if not all) of the tools available to the academic community. It should be noted that some of these are no longer actively maintained or developed.

Those tools fundamentally differ from one another by the formalisms they use to represent the systems and the requirements, but also by the different algorithms they implement.

The most well-known tool of this list is likely UPPAAL-SMC. Its usage is widespread.

In the context of this thesis, we mostly used Plasma Lab as we initially focused on the formalism of Markov decision processes. Moreover, it already included the algorithm that we studied and improved in Chapter 4.

Name	Formalisms	Summary	Reference(s)
Approximate Probabilistic Model Checker	Probabilistic transition systems, discrete time Markov chains + linear temporal logic	A relatively simple tool that implements a Monte-Carlo based method to approximate the probability that a property is satisfied. It relies on the Chernoff bound.	[Hér+04]
Bayesian Tool	Stateflow / Simulink models (discrete time hybrid systems) + bounded linear temporal logic	Implements both hypothesis testing and basic estimation capabilities for the probability that a property is verified for a stochastic hybrid system. It relies on Bayesian statistics.	[Jha+09; ZPC13]
COSMOS	Discrete event stochastic processes (includes continuous time Markov chains) + hybrid automata stochastic language	Implements algorithms to estimate the probability that a property holds. It implements both *a priori* sample size estimation techniques and stopping algorithms. It relies on the Chernoff bound and the Chow and Robbins stopping condition.	[Bal+15]

Name	Formalisms	Summary	Reference(s)
GreatSPN	Generalized stochastic Petri nets + computational tree logic	Main module: implements algorithms to verify a computational tree logic property holds in a generalized stochastic Petri net. It relies on numeric symbolic algorithms. Secondary module: verification of continuous stochastic logic properties in continuous time Markov chains.	[SVA05; DHS08; ABD14]
MARCIE	Generalized stochastic Petri nets and continuous time Markov chains + continuous stochastic logic, continuous stochastic reward logic, and probabilistic linear temporal logic	Implements basic estimation algorithms. It relies on the Chernoff bound, but tries to take into account the variance of the simulations to speed up the verification process.	[HRS13]
Modest	Various enhanced forms of automata, Markov decision processes, labelled transition systems, discrete time and continuous time Markov chains, interactive Markov chains and Markov automata + Modest language, JANI language	Consists of multiple tools: classic model checkers, statistical model checkers, conversion tools, and graphical representation tools. It relies on a large variety of theories and techniques.	[BHH12; TU; HH14]

Name	Formalisms	Summary	Reference(s)
Plasma Lab	Continuous time Markov chains and Markov decision processes + bounded linear temporal logic	Iimplements many different algorithms, including basic estimation algorithms for probabilities based on the Chernoff bound, hypothesis testing algorithms, the importance splitting algorithm and the smart sampling algorithm.	[JLS12a; Boy+13]
PRISM	Discrete and continuous time Markov chains, Markov decision processes + probabilistic computational tree logic, continuous stochastic logic, and linear temporal logic	Implements many different algorithms, including various confidence interval methods, the Approximate Probabilistic Model Checking method and the sequential probability ratio test of Wald.	[KNP11b; Oxf]
SBIP	Discrete and continuous time Markov chains, generalized semi Markov processess + probabilistic bounded linear temporal logic and metric temporal logic	Implements techniques for both qualitative and quantitative properties. Qualitative properties are checked with basic sampling techniques or the sequential probability ratio test of Wald, while the quantitative properties are verified by relying on the Chernoff bound.	[Nou+18; Alp]

Name	Formalisms	Summary	Reference(s)
UPPAAL-SMC	A large class of timed and hybrid systems, including stochastic hybrid automata and stochastic timed automata + variants of metric (interval) temporal logic	UPPAAL(-SMC) is a huge toolbox which implements classic Monte Carlo algorithms based on the Chernoff bound and sequential probability ratio test strategies. In addition, it offers some really nice visualization tools.	[Ben+96; Dav+15]
(P)VeStA and (Multi)VeStA	Discrete and continuous time Markov chains, PMAUDE (an executable algebraic specification language) models + continuous stochastic logic, probabilistic computational tree logic, and quantitative temporal expressions for PMAUDE models	Implements classic hypothesis testing, a variant of the sequential probability ratio test of Wald, and an algorithm derived from the Chow-Robbins method.	[AM11; SV13]
Ymer	Discrete and continuous time Markov chains. + continuous stochastic logic and probabilistic computational tree logic	Implements basic estimation algorithms and the sequential probability ratio test of Wald.	[You05]

Background

2

In this chapter, we define all the standard notions and remind the reader of some well-known results which are needed for the understanding of this thesis.

2.1 Statistics and Probabilities

A **random variable** X is a measurable function from a probability space $(\Omega, \mathcal{F}, P)$, with Ω being the set of possible outcomes, to a measurable space $\mathcal{M}$. For most practical applications, real-valued random variables are used, i.e. $\mathcal{M} = \mathbb{R}$. A (real-valued) random variable X is entirely characterized by its probability distribution, which can be expressed with the cumulative distribution function of X: $\mathrm{CDF}_X(x) = P(X \leq x)$. The expected value of X is $\mu_X = E(X) = \int_\Omega X \, dP$, the variance of X is $V_X = V(X) = E[(X - E(X))^2]$, and the standard deviation of X is $\sigma_X = \sigma(X) = \sqrt{V(X)}$.

Random variables can be used to formalize any random experiment, such as the generation of a possible specialization or execution of a system model in the context of SMC. In practice, the random variable X whose parameter needs to be estimated can be seen as any quantity associated with the system. The realizations of this random variable are individual values that the quantity X can take with different possible specifications or executions of the system, and one can obtain such realizations by running (independent) simulations of the system. Especially for SMC, the parameter of interest p is generally taken to be the expected value $E(X)$ of the random variable. In particular, it can correspond to the probability that a given property which needs to be verified holds true for a random execution of the system. In some cases, that quantity can correspond to the probability that a specific requirement of the system is verified during one of its random possible executions, in which case the corresponding random variable is bounded in $[0; 1]$ [SVA05]. But it can represent more concrete physical quantities as well, such as the fuel consumption of an engine [ZPC10].

A rule or an algorithm to compute an estimation $\hat{p}$ of a parameter p of the distribution of a random variable X based on realizations (observations) of X is called an **estimator**. Such an estimator is said to be unbiased if $E(\hat{p}) = p$.

For example, if $x_1, ..., x_n$ are n observations of X, the sample mean $\frac{1}{n}\sum_{i=1}^{n} x_i$ is an unbiased estimator for $E(X)$, but the (uncorrected) sample variance (sometimes called population variance) $\frac{1}{n}\sum_{i=1}^{n}(x_i - (\frac{1}{n}\sum_{i=1}^{n} x_i))^2$ is a biaised estimator for $V(X)$.

For a precision $\epsilon > 0$ and and a confidence (level) $\delta > 0$, an estimation $\hat{p}$ is a **(ϵ,δ)-estimation** if $P((1-\epsilon)p \leq \hat{p} \leq (1+\epsilon)p) \geq 1-\delta$, which is equivalent to $P(|\hat{p} - p| \geq \epsilon p) \leq \delta$. An ($\epsilon$,$\delta$)-estimation is an estimation that comes with a probabilistic bound on the relative error of the estimation.

Similarly, an estimator is an **(ϵ, δ)-estimator** if for all $\epsilon > 0$ and $\delta > 0$, there always exists a sample size $N_{(\epsilon,\delta)}$ such that the estimation produced from any sample of that size is an (ϵ,δ)-estimation.

The notion of (ϵ,δ)-estimation is key for SMC algorithms. Indeed, since those algorithms are stochastic, their outputs must come with statistical guarantees: the precision ϵ of their output (how close the estimation should be to the true value) and the confidence associated with their output (how likely it should be that the true value is within the precision of the estimation). SMC algorithms always ask for those two parameters, either explicitly or indirectly.

2.1.1 Central Limit Theorem

For statistical model checking methods, the sampling process can generally be assumed to be independent and to be always performed following a constant distribution: the simulations are executed independently of each other and the model's behaviour does not change from one simulation to the next (nondeterminism of the model put aside).

Mathematically, this corresponds to the setting where we have access to a large number of random variables $X_1, ..., X_n$ which are **independent and identically distributed (iid)**: they all have the same probability distribution and they are mutually independent.

The assumption that a sequence of random variables $X_1, X_2, ...$ is iid is the key hypothesis of the **central limit theorem**, a fundamental theorem in statistics. The most common version of the central limit theorem states that if the random variables have a finite expected value μ and a nonzero variance σ^2, then $\frac{\sqrt{n}}{\sigma}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - E\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right)\right) = \frac{\sqrt{n}}{\sigma}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right)$ converges in distribution to the standard normal distribution $\mathcal{N}(0,1)$:

$$\frac{\sqrt{n}}{\sigma}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right) \xrightarrow{d} \mathcal{N}(0,1)$$

A random variable $Z \sim \mathcal{N}(\mu, \sigma^2)$, that is, following a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with expected value μ and variance σ^2, has the following cumulative

distribution function:

$$\mathrm{CDF}_Z(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-(t-\mu)^2}{2\sigma^2}\right)} \mathrm{dt}$$

If $\mu = 0$ and $\sigma^2 = 1$, the distribution is called the standard normal distribution and its cumulative distribution function is noted Φ. Related to the cumulative distribution function of the standard normal distribution is the error function, which is defined as $\mathrm{erf}(x) = P(-x \leq Z \leq x)$ with $Z \sim \mathcal{N}(0, 1/2)$. Explicitely:

$$\mathrm{erf}(x) = \int_{-x}^{x} \frac{1}{\sqrt{\pi}} e^{-t^2} \mathrm{dt} = 2 \int_{0}^{x} \frac{1}{\sqrt{\pi}} e^{-t^2} \mathrm{dt}$$

Finally, Φ and erf are related by the following formulas:

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

$$\mathrm{erf}(x) = 2\Phi(x\sqrt{2}) - 1$$

2.1.2 Concentration Inequalities

The central limit theorem is not the only statistical result that can help understand the behavior of the sum of iid random variables. In particular, **concentration inequalities** are very useful theorems that provide a bound on the probability that a random variable deviates from a specific value. There exists a multitude of well-known concentration inequalities which can be applied to large classes of random variables, among which are Markov's inequality, Chebyshev's inequality, and the Chernoff bound.

A particularly important collection of concentration inequalities for sampling and estimation algorithms (and thus SMC) are those that are specific to random variables $S_n = X_1 + ... + X_n$ which can be decomposed as the sum of n independent random variables, such as the random variables that follow a binomial distribution. The Hoeffding and the Bernstein inequalities belong to that group, among others such as Azuma's inequality, McDiamid's inequality or Benett's inequality.

If the random variables $X_1, ..., X_n$ are iid, additional bounds can be derived, for instance from the central limit theorem. Moreover, many classic concentration inequalities can be simplified and enhanced under the assumption that the random variables $X_1, ..., X_n$ are iid.

Below is a nonexhaustive list of some of the most well-known and useful concentration inequalities. Most of them are regarded as relatively elementary results in statistics [BLB03; Was04; CB21].

Let X be a random variable.

- Markov's inequality (X must be almost surely nonnegative):

$$\forall \epsilon > 0 : P(X \geq \epsilon) \leq \frac{E(X)}{\epsilon}$$

- Chebyshev's inequality ($E(X)$ and $V(X)$ must be finite):

$$\forall \epsilon > 0 : P(|X - E(X)| \geq \epsilon) \leq \frac{V(X)}{\epsilon^2}$$

- Chernoff bound (the moment generating function of X, $M_X(t) = E(e^{tX})$, must be well-defined and finite):

$$\forall \epsilon > 0 : P(X \geq \epsilon) \leq \frac{E(e^{tX})}{e^{t\epsilon}}$$

Now let $X = S_n = X_1 + ... + X_n$ be the sum of n independent random variables, with each X_i almost surely bounded within $[a_i, b_i]$. Let $R_i = b_i - a_i$ be the range of X_i. Let be $R = \max_{1 \leq i \leq n} (R_i)$.

- Hoeffding's inequality:

$$\forall \epsilon > 0 : P(|S_n - E(S_n)| \geq \epsilon) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

- Bernstein's inequality:

$$\forall \epsilon > 0 : P(|S_n - E(S_n)| \geq \epsilon) \leq 2 \exp\left(-\frac{\epsilon^2}{2V(S_n) + (2/3)\epsilon R}\right)$$

- Both Azuma's Inequality and McDiamid's inequality are equivalent to Hoeffding's inequality under those assumptions:

$$\forall \epsilon > 0 : P(|S_n - E(S_n)| \geq \epsilon) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

- Benett's inequality (with $f(x) = (1 + x)\log(1 + x) - x$):

$$\forall \epsilon > 0 : P(|S_n - E(S_n)| \geq \epsilon) \leq 2 \exp\left(-\frac{V(S_n)}{R^2} f\left(\frac{\epsilon R}{V(S_n)}\right)\right)$$

2.1.3 Sample Size Minimization

The most elementary but also the most fundamental algorithm of SMC is the Monte Carlo method, which consists in simulating a system a large number of times and taking the average of the results to perform an estimation. However, if statistical guarantees are required for that estimation, how to decide how many simulations need to be completed?

Algorithm 1
Generic Monte Carlo Method for SMC

Input: executable model M, estimator $\hat{x}$, precision ϵ, confidence δ
Output: (ϵ, δ)-estimation of x

1: $N \leftarrow$ compute_sample_size(ϵ, δ)
2: $t \leftarrow 0$
3: sample $\leftarrow \{\}$
4: **while** $t \leq N$ **do**
5: $\quad t \leftarrow t + 1$
6: $\quad$ sample $\leftarrow$ sample $\cup$ new_simulation
7: **return:** $\hat{x}$(sample)

Designing experiments and protocols which minimize the size of a required sample is a very old problem in statistics [SM14]. The sampling process is a common feature shared by most, if not all, stochastic algorithms.

It is to be noted that for the verification of real-world systems, the sampling process and the simulations of the system represent the quasi totality of the computational work needed. This phenomenon is well illustrated in Chapter 5.

Neglecting the importance of the sampling process can thus have dire consequences for a SMC algorithm. If the system under scrutiny is so complex that any single simulation asks for extensive computations, or if the verification process must be performed "online", or if the resources allocated for the verification are purposefully limited (for instance within an embedded system), not minimizing the size of the sample which needs to be exploited by the algorithm can mean that a perfectly valid and efficient algorithm will not be viable in practice. Therefore, many techniques have already been developed to improve the sampling strategies in the context of SMC [BHP12; JLS12b; DAr+15b].

In practice, most SMC algorithms try to guess *a priori* the size of the necessary sample. Unfortunately, either for the sake of simplicity or so that they can be as universal as possible, SMC algorithms generally exploit very general theoretical results to compute that sample size, which leads to overshooting

[AP18; CZ11]. Moreover, independently of whether the sample is completely generated at the start or in multiple steps, its generation is made agnostically, without exploiting any knowledge related to or derived from the system.

For example, when estimating probabilities, a classic method to compute a sample size *a priori* is to use the Chernoff-Hoeffding bound [Oka59a; DAr+15a]: to produce an (ϵ, δ)-estimation of an unknown probability p, the sample size

$$N = \frac{\ln(2) - \ln(\delta)}{2\epsilon^2}$$

is always sufficient, independently of p. This strategy is for instance implemented in PRISM as the APMC (Approximate Probabilistic Model Checking) method [Hér+04] and used in PLASMA (Platform for Learning and Advanced Statistical Model checking Algorithms) [LST16b] for the Smart Sampling algorithm [DAr+15b].

Of course, this sample size might not be optimal, as we demonstrate in Chapter 3.

2.2 Formalisms

To apply SMC for the verification of CPSs, we need two formalisms: one for the models of the systems, and one to encode the properties to verify.

2.2.1 Markov Decision Processes

As CPSs are immensely varied, a lot of different frameworks have been invented to represent them: discrete time Markov chains and continuous time Markov chains [Bai+03; YS02], generalized semi Markov processes [SVA04; Whi80], generalized stochastic Petri nets [Bal+15], population models [Lui+17], probabilistic timed automata [NPS13], stochastic hybrid automata and stochastic timed automata [Dav+12], and, last but not least, Markov Decision Processes (MDPs) [LST14; Put14]. Many share similarities since they are often designed to tackle the common specificities of CPSs (heterogeneous, nondeterministic, interacting with their environment, dynamic...), but there exists no universal solution to model those complex systems. In the fourth chapter of this thesis (Chapter 4), we focus on Markov Decision Processes.

Markov Decision Processes allow to model both controllable nondeterminism and uncontrollable nondeterminism, which is necessary to modelize situations where an agent-based system operates in a dynamical environment with uncertainty. A **Markov Decision Process (MDP)** is a tuple $(S, s_0, A, \{P_a\}_{a \in A})$ where S is a finite set of states, s_0 is the initial state, A is a finite set of (labelled) actions and $\{P_a\}_{a \in A}$ is a set of probabilities over pairs of states such as for all $a \in A$, there exists a unique state $s \in S$ for which $P_a(s, s')$ can be

greater than zero (that is, each action has a single source state but multiple possible target states). In some cases, a reward function which is a function $R : A \times S \times S \to \mathbb{R}$ is also included in the definition.

A MDP can be seen as a directed graph whose edges start from a single vertex but can point to multiple vertices, with a probability associated with each of those possible destinations. In any state (vertex) s, a state s' is reachable if there exists an action (edge) a from s to s' with $P_a(s, s') > 0$. A (finite) path is a (finite) sequence of states $(s^1, s^2, ...)$ such that s^{i+1} is reachable from s^i for all $i \in \mathbb{N}_0$. A (finite) trace is a (finite) path such that $s^1 = s_0$. We use the notations T^n for the set of all finite traces of length n, and T^* (or T^∞) for the set of infinite traces. For each simulation of a MDP, a trace is produced.

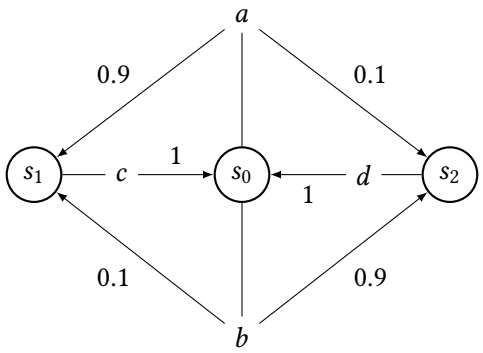


Figure 2.1: A simple MDP with 3 states s_0, s_1, s_2 and 4 actions a, b, c, d

To resolve the controllable non-determinism of a MDP, i.e. the necessity to select the actions which will be chosen during one simulation, one can refine a MDP into a Markov chain with the choice of a scheduler. In practice, a scheduler can be interpreted as a possible interaction with the environment (a possible choice of a user for instance) or as a chosen strategy for the execution of the system to achieve a specific goal. A **scheduler**, also called policy, is a function which indicates at any point during the generation of a trace which is the action that needs to be considered to determine the next state of the trace.

The most basic kind of schedulers are memoryless schedulers whose choices only depend on the current state, i.e. functions of the form $\pi : S \to A$. For reachability problems, memoryless schedulers suffice, but they form a strict subset of history-dependent schedulers. Finite-memory schedulers take as inputs finite paths of a predetermined maximum length n, i.e. are functions of the form $\pi : (s^i)_{1 \leq i \leq n} \to A$. Infinite-memory schedulers take as inputs finite traces with no predetermined maximum length, i.e. are functions of the form $\pi : T^* \to A$. Schedulers that do not truly resolve the controllable non-determinism are also possible and are called probabilistic schedulers, i.e.

functions of the form $\pi : S \to Dist(A)$, where $Dist(A)$ is the set of probability distributions over A. A standard probabilistic scheduler is one that choose the uniform distribution over all the available actions in each state. In this thesis, unless specified otherwise, scheduler is used as a shortcut for memoryless scheduler.

The choice of a scheduler turns a MDP into a Markov Chain. A **Markov chain (MC)** is a Markov decision process with exactly one available action for each state. Once a MDP is specified as a Markov chain, we can automatically define the associated probability space $(\Omega, \mathcal{F}, \mu)$. $\Omega = T^\infty$ is the set of all infinite traces of the Markov chain, $\mathcal{F}$ is the σ-algebra generated by subsets Ω of the form $\Omega_{(s^1,...,s^n)} = \{(t^1, ..., t^n, ...) \in \Omega \mid t_1 = s_1, ..., t_n = s_n\}$, and $\mu : \mathcal{F} \to [0, 1]$ is the distribution over $\mathcal{F}$ which verifies for any finite trace $(t_0, ..., t_n) \in T^n$: $\mu(\Omega_{(t_0,...,t_n)}) = \prod_{i=0}^{n-1} P_{a_{t_i}}(t_i, t_{i+1})$, with a_{t_i} being the unique action available in the Markov chain at state t_i. Such a distribution μ_π can be built for each scheduler π of a MDP, and for any (finite) trace T of the MDP the probability $\mu_\pi(T)$ is the probability that a simulation of the MDP under the policy π will produce the trace T.

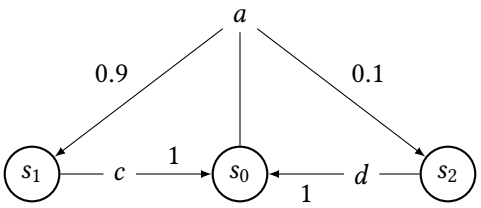


Figure 2.2: MC obtained after specifying the MDP of Figure 2.1 with a scheduler π such that $\pi(s_0) = a$

2.2.2 Temporal Logic

To formalize the properties of cyber-physical systems whose models are MDPs or similar objects, some form of logic must be used. There are many different options: basic linear temporal logic [Roz11], bounded linear temporal logic [Kam12; DAr+15b], signal temporal logic [MN04], signal spatio-temporal logic [Lui+17], metric temporal logic [Med+18], metric interval temporal logic [AFH96], computational tree logic [Cla+18c], probabilistic computational tree logic [HJ94; Bai98], continuous stochastic logic [SVA05], quantitative temporal expressions [AMS06], CSL^{TA} [DHS08], mobile stochastic logic [De +07], ...

In this thesis we focus on **Bounded Linear Temporal Logic (BLTL)**. BLTL is a variant of Linear Temporal Logic (LTL), which is itself a modal extension of propositional logic. The addition of only one modal operator is sufficient to fully define BLTL.

A minimal BNF grammar for the syntax of BLTL is:

$$\phi, \psi ::= \bot \mid \alpha \mid \neg\phi \mid \phi \wedge \psi \mid \phi\, U^k\, \psi$$

α denotes an atomic proposition. In the context of statistical model checking, those can be simply defined as a list of logical variables $\{\alpha_s\}_{s\in S}$ with α_s true if and only if the current state is s, but more complex atomic proposition can be defined, for example with fluents. $\neg$ and $\wedge$ are the basic operators of propositional logic for the negation and the conjunction. U^k (with $k \in \mathbb{N}$) is the (bounded) until operator.

The semantics of BLTL operators are defined in a similar way to what is done for linear temporal logic. In the context of SMC, the models for the formulas of BLTL are usually taken as pairs (T, z) with T being a (infinite) trace of a Markov Chain and z being a starting index for the sequence T. The semantics of the $\neg$ and $\wedge$ operators are the same as with propositional logic, while the semantic of the U^k operator formalizes the intuitive idea of "true if the first formula is true until the second one becomes true, in a time interval of length k". More formally, for a trace T: $(T, z) \models \phi\, U^k\, \psi$ iff there exists $j \in \mathbb{N}$ with $z \leq j \leq z + k$ such that $(T, i) \models \phi$ for all $z \leq i \leq j$ and $(T, i) \models \psi$ for all $j < i \leq k$. Just as the $\phi \vee \psi$ operator can be defined as $\neg(\neg\phi \wedge \neg\psi)$ within propositional logic, additional operators can be defined for (bounded) linear temporal logic. The most common ones are $X\,\phi$, $F^k\,\phi$ and $G^k\,\phi$, the next, the finally/eventually and the globally/always operators, respectively defined as $\top U^1 \phi$, $\top U^k \phi$ and $\neg(\top U^k \neg\phi)$.

Given a BLTL formula ϕ associated with a MDP, a **score** can be given to any scheduler π by considering the quantity $\int_\Omega \mathbb{1}_{T\models\phi}\, d\mu_\pi$, i.e. the exact probability that the property of the system formalized by the formula ϕ is verified for a random possible execution of the MDP under the policy π. For small MDPs, one can compute the exact score of all schedulers. However, in practice, that quantity can often only be estimated, which is usually done with a basic Monte-Carlo algorithm.

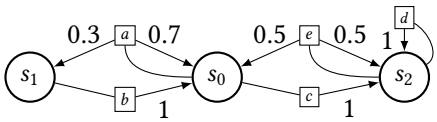


Figure 2.3: A MDP with 3 states and 5 actions. The schedulers that choose the action c in state s_0 have a score of 0 for the formula $X(\neg\alpha_{s_0}) \Rightarrow (\alpha_{s_0} U^3 \alpha_{s_1})$, while any scheduler which picks action a has a score of $1 - (0.7)^3 = 0.657$. No memoryless scheduler can produce a trace that satisfies $G^8(F^4\alpha_{s_1} \wedge F^4\alpha_{s_2})$, but it is possible with a history-dependent scheduler.

2.3 Temporal Networks

Temporal networks (TN) are networks (graphs) whose links' availability is dependent on time. They are used in a wide array of contexts, from communication network analysis [EMS04; KPV14] to epidemic modeling [Ste+11]. They can help solve problems related to scheduling [San+16b], temporal constraint reasoning [DMP91], environmental uncertainty [VG02] or resource management [WF00; Lab03; Sid+16; Kum+18].

A **Simple Temporal Network (STN)** is a pair (T, C) with T a set of real-valued variables t_i named time points and C a set of constraints $c(t_i, t_j)$ that encode bounds of the form $l_{i,j} \leq (t_j - t_i) \leq u_{i,j}$ with $l_{i,j} \leq u_{i,j}$ (and, in practice, $0 \leq l_{i,j}$). For a STN, a schedule is simply an assignment to all time point variables t_i, i.e. a function from T to $\mathbb{R}$.

A **Simple Temporal Network with Uncertainty (STNU)** is a pair (T, C). $T = T_E \cup T_C$ is a set of *executable* time points (T_E) and *contingent* points (T_C). The executable time points are controllable, while the contingent time points are determined by the environment. $C = C_E \cup C_C$ is a set of *requirement* constraints (C_E) and *contingent* constraints (C_C). Requirement constraints are decided by an agent, while contingent constraints are not fully known (in advance) and cannot be controlled. The contingent constraints link executable time points to contingent time points, representing uncertainty with respect to the time difference between the points: the exact value of $l_{i,j} \leq (t_j - t_i) \leq u_{i,j}$ is unknown prior to execution but is bounded. For a STNU, a schedule is an assignment to all executable time point variables $t_i \in T_E$, i.e. a function from T_E to $\mathbb{R}$.

A **Probabilistic Simple Temporal Network (PSTN)** is a STNU for which a probability distribution is associated to each contingent constraints: all $c \in C_C$ are represented as $(t_j - t_i) = X_{i,j}$ where $X_{i,j}$ is a random variable (which typically follows a normal distribution).

Lastly, **Simple Temporal Networks with Resources (STNR)** are a recent extension to STNs whose goal is to incorporate resource management into the formalism. Those resources can be consumable resources (e.g., energy, fuel) [WF00; Lab03; Sid+16; Kum+18] or unary resources (e.g., instruments). There is no commonly accepted definition for STNR yet. For consumable resources, it has been attempted to represent them as auxiliary networks [WF00; Lab03; Sid+16; Kum+18] or to develop Time Resource Networks (TRN) as an extension of the framework that also covers PSTNs [Sid+16]. For unary resources, the work from [HLV02] and [Com+19] focus on extending STNUs to include elements such as instruments or agent assignments.

For a concrete example of a temporal network (PSTN), see Figure 6.1 in Chapter 6.

Estimation Algorithms | 3

In this chapter, we explore a specific class of estimation algorithms for SMC: (adaptive) stopping algorithms. We detail how they can offer a remarkable solution to the problem of sample minimization, which has great implications for the applicability of SMC strategies. We present a unifying novel theorem which generalizes the construction of existing stopping algorithms. We use that theorem to generate a new stopping algorithm designed to tackle the rare event problem. Additionally, we discuss some practical optimizations which can be implemented to further improve the efficiency of stopping algorithms. Finally, we prove a brand-new result which provides a bound for the effectiveness of stopping algorithms and we show how that bound can be reached.

The content of this chapter is related to the conference publication titled "Adaptive Stopping Algorithms Based on Concentration Inequalities":
M. Parmentier and A. Legay. "Adaptive Stopping Algorithms Based on Concentration Inequalities". In: *International Conference on Bridging the Gap between AI and Reality*. Springer. 2024, pp. 171–187.

3.1 Introduction

Estimation algorithms form the building blocks of SMC. Even for SMC methods whose goal is not simply to estimate a key performance indicator of a system or the parameter of an underlying probability distribution, but that are designed for hypothesis testing or optimization, it is generally required to aggregate the information accumulated through numerous simulations with an estimation most often computed with a classic Monte Carlo algorithm. Moreover, in many cases, this subroutine must be performed thousands, if not millions of times. As soon as the systems under verification are not trivial, this corresponds to the most costly step(s) of practically all SMC strategies since the computational cost of even a few simulations of the system far exceeds the cost of the computations of the overarching SMC algorithm.

Therefore, making estimation algorithms as efficient as possible is key to improve the actual usability and the scalability of SMC for real-world cyber-physical systems. The best way to do so is to minimize the number of system simulations which have to be performed for each estimation.

Algorithm 2
Generic Adaptive Stopping Algorithm

Input: random variable X,
precision ϵ and confidence level δ
Output: (ϵ, δ)-estimation of $\mu = E(X)$

```
1: t ← 1
2: sample ← {x_1}
3: μ̂_1 ← x_1
4: while ¬ stopping criterion do
5:     t ← t + 1
6:     sample ← sample ∪ {x_t}
7:     μ̂_t ← empirical_mean(sample)
8:     update_stopping_criterion(sample, μ̂_t, ϵ, δ)
9: return: μ̂_t
```

Unfortunately, basic Monte Carlo algorithms are particularly ill-fitted to reach that goal, as they determine the sample size they need for the estimation *a priori* and without exploiting any information about the system along the way, often making them overshoot their target, sometimes by several orders of magnitude.

An alternative is (Adaptive) Stopping Algorithms (SAs). Those approximation algorithms do not fix the number of required samples in advance, but instead monitor their own progress and gradually ask for new samples until enough knowledge has been accumulated to compute an approximation of the quantity of interest with the sought statistical properties.

Some SAs have even been proven to be optimal, in that they are probabilistically guaranteed to stop after a number of samples which is always within a constant factor from the minimum number of samples theoretically needed to compute the requested approximation.

Moreover, SAs can be applied in an extremely large range of situations since they only modify the sampling process and not how the samples are (or must be) generated. They can be combined with other more complex algorithms, often providing them with a free boost in performance. For example, SAs have been used to scale up machine learning algorithms [BS07] or to offer an efficient solution to the multi-armed bandit problem [AMS07a; EMM02]. Of course, they have been extensively studied for their applications for SMC as well [DW+00; DGW02; MSA08; Dag+00].

In general, a SA is built upon a stopping criterion, also called stopping rule. The various SAs available (some of them are described in Section 3.2) mainly differ with respect to the statistical result exploited to derive their

stopping criterion. The difficulty of building new SAs lies in finding and demonstrating the validity of such a stopping criterion. In its most generic form (see Algorithm 2), a SA is a single loop that iteratively checks if the stopping criterion has been reached, produces one individual sample, and then update the estimation and the stopping criterion.

The stopping criterion depends on the precision ϵ and the confidence level δ chosen for the estimation, but also on the already accumulated sample at any step. This means that, if built properly, the stopping criterion can continuously take into account the knowledge which has been obtained about the system. Typically, the stopping criterion will for instance stay unverified for longer if the elements of the sample that have been produced up to a certain point have a high variance.

3.2 Background

We now give a brief description of the state of the art for SAs in the context of SMC, with a focus on the two adaptive stopping algorithms which led to our generalization.

In general, statistical results similar to the central limit theorem and concentration inequalities are a good source of theorems that can lead to meaningful stopping criteria. For instance, the AdaSelect algorithm [DW+00] is a SA developed by Domingo, Watanabe, et al. for bounded random variables with expected value $\mu \neq 0$. It is a straightforward SA whose stopping criterion is of the form $|\hat{\mu}_t| < c_t$, where $c_t = k\sqrt{\frac{\log(t(t+1)/\delta)}{2t}}(1 + 1/\epsilon)$ are parameters of the algorithm with $k > 0$ depending on the range of the random variable. The expressions for these parameters are derived from Hoeffding's inequality.

The EBStop algorithm [MSA08] was introduced in 2008 by Mnih, Szepesvári, and Audibert as an improvement of the AdaSelect algorithm. Similarly to its predecessor, EBStop builds a sequence of parameters c_t as its stopping criterion. However, by exploiting the (empirical version of) Bernstein's inequality [AMS07a] instead of Hoeffding's inequality, the variance of the random variable X takes part in the computation of such parameters. Especially when the variance is significantly smaller than the range R of the random variable, these parameters decrease much faster, therefore putting an end to the sampling process much sooner. For any fixed sequence $(d_t)_{t\in\mathbb{N}}$ such that $\sum_{t=1}^{\infty} d_t \leq \delta$, the parameters c_t can be taken as $c_t = \hat{\sigma}_t\sqrt{\frac{2\log(3/d_t)}{t}} + \frac{3R\log(3/d_t)}{t}$, with $\hat{\sigma}_t$ being the sample standard deviation at step t.

The EBStop algorithm adds another improvement to the most basic version of the AdaSelect algorithm. The stopping criterion is divided into a lower stopping criterion $(1+\epsilon)\max\{0, \max_{1\leq s\leq t}(|\hat{\mu}_s| - c_s)\}$ and an upper stopping criterion $(1+\epsilon)\min_{1\leq s\leq t}(|\hat{\mu}_s| - c_s)$, so that X must not be supposed non-

negative. The most upgraded version of the algorithm, called EBGStop, also adopts a geometric sampling strategy, accumulating samples at an exponential rate β rather than with a linear rate. The right choice of β and additional countermeasures [MSA08; AMS07b] allow for a speed-up of the algorithm without running the risk of taking more samples than needed.

If $R > 0$ is the (finite) range of the random variable X, Mnih, Szepesvári, and Audibert proved that the EBStop algorithm will require a sample size smaller than a multiple of $\max\left\{\frac{\sigma^2}{\epsilon^2,\mu^2}, \frac{R}{\epsilon|\mu|}\right\} \times \left(\log\frac{1}{\delta} + \log\frac{R}{\epsilon|\mu|}\right)$, which guarantees its (near-)optimality in a similar way to is main competitor: the AA algorithm.

The AA algorithm [Dag+00] is a SA in three steps, introduced in 2000 by Dagum et al. It was built upon the theory of supermartingales and a generalized version of the zero-one estimator theorem. It can only be applied to random variable X that is bounded and non-negative. AA is significantly more complex than EBStop. Actually, AA can be seen as a relatively basic Monte-Carlo algorithm for its second and third steps, with the size of the required sample being computed indirectly with a SA during the first step.

AA first computes a $\left(\max\left\{\sqrt{\epsilon}, \frac{1}{2}\right\}, \frac{\delta}{3}\right)$-approximation $\tilde{\mu}$ of μ, which is only used to compute a Monte-Carlo estimation $\tilde{\sigma}^2$ of σ^2, which is in turn exploited to derive a value $N = \max\left\{\tilde{\sigma}^2, \epsilon\tilde{\mu}\right\} \frac{\log(\frac{1}{\delta})}{\epsilon^2\tilde{\mu}^2}$ for the final sample size which guarantees that the final Monte-Carlo estimation $\hat{\mu}$ is an (ϵ, δ)-approximation of μ.

While the hypothesis that X is non-negative makes AA a bit more restrictive than EBGStop, it was proven to be fully optimal for that class of random variables. Namely, for any bounded and non-negative random variable X, Dagum et al. showed that with probability $1 - \delta$, AA will require a sample size that is at most a multiple of $\max\left\{\sigma^2, \epsilon\mu\right\} \times \frac{1}{\epsilon^2\mu^2}\log(\frac{2}{\delta})$, while also proving that any SA designed for that class of random variables will always require with probability $1 - \delta$ a sample size of at least a multiple of that exact quantity.

EBStop has been effectively exploited to solve subset selection problems [Bus+13], policy search problems [HI09a], and improve reinforcement learning strategies [HI09b]. AA has for example been used to compute the permanent of matrices [CRS02; Für00; FK05; Lia+07], find perfect matchings of bipartite graphs [Hub06] and to study viral marketing [Bor+14; NTD16; TXS14; TSX15].

3.3 Concentration Inequality Functions

As a preparation for the next section, we introduce the definition of concentration inequality function to link together the notions of concentration

inequalities and (ϵ, δ)-estimations. As mentioned in Section 2.1.1, in the context of SMC, the sampling process can generally be assumed to be independent and to follow a constant distribution. Therefore, it makes sense to speak of a concentration inequality that holds for a single real-valued random variable X by implying that the concentration inequality can be applied to any sequence $X_1, ..., X_n$ of random variables iid to X.

Definition 3.1 (Concentration Inequality Function)**.** *Let $X_1, ..., X_n$ be real-valued random variables that are iid to a random variable X.*
A function $F_X : \mathbb{N}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}^+$ such that, for all $n \in \mathbb{N}_0$ and for all $\epsilon > 0$:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \leq F_X(n, \epsilon) \tag{3.1}$$

is called a concentration inequality function *for X.*

Proposition 3.1 links the notions of concentration inequality function and (ϵ, δ)-estimation. Proposition 3.1 shows that if F_X is a concentration inequality function for X, then any realization of $\frac{1}{n}\sum_{i=1}^{n} X_i$ is an (ϵ, δ)-estimation of $E(X)$ with $\delta = F_X(n, n\epsilon)$.

Proposition 3.1. *Let X be a random variable of expected value $\mu = E(X)$. Let $X_1, ..., X_n$ be random variables iid to X. Let F_X be a concentration inequality function for X. Then, for all $n \in \mathbb{N}_0$ and for all $\epsilon > 0$:*

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \leq F_X(n, n\epsilon)$$

Proof. Equation (3.1) with $\epsilon^* = n\epsilon$ gives:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq n\epsilon\right) \leq F_X(n, n\epsilon)$$

However, since $X_1, ..., X_n$ are iid to X:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq n\epsilon\right) =$$

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \frac{1}{n}\sum_{i=1}^{n} E(X_i)\right| \geq \epsilon\right) =$$

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \frac{n\mu}{n}\right| \geq \epsilon\right)$$

□

3.4 The GSA Theorem

AdaSelect and EBStop are both based on concentration inequalities and the construction of their stopping criterion is similar. The intuition that led to our generalization of that process is the following. A concentration inequality that is valid for a random variable X (or a specific class of random variables) gives a bound to the probability that a sample mean deviates from the expected value of X by a quantity larger than $\epsilon > 0$ (Proposition 3.1). This means that for any ϵ, someone performing a Monte-Carlo estimation of $\mu = E(X)$ can directly know how likely it is that the estimation does not possess the required precision, *i.e.* the confidence level of the estimation, in function of the size of the sample which was used. Therefore, to obtain an (ϵ, δ)-approximation of μ, one can successively produce (new and independent) larger and larger samples, hoping that the values of the bounds eventually become smaller than the chosen δ. With Hoeffding's inequality and Bernstein's inequality, both of the corresponding concentration inequality functions, $F(n, \epsilon) = 2\exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n}(b_i-a_i)^2}\right)$ and $F(n, \epsilon) = 2\exp\left(-\frac{\epsilon^2}{2V(S_n)+(2/3)\epsilon R}\right)$, are such that $F(n, n\epsilon)$ is (strictly) decreasing with respect to n and such that $\lim_{n\to\infty} F(n, n\epsilon) = 0$ (for any fixed ϵ). Therefore, with those two concentration inequality functions, for any δ, this process of repeating Monte Carlo estimations with larger and larger samples will always eventually terminate.

Of course, such a strategy is not only impractical, it is extremely inefficient. Therefore, instead of blindly trying larger and larger sample sizes, we consider the inverse problem of identifying a sequence of decreasing precision levels "ϵ_t" with the property that the series of the sequence $F(t, t\epsilon_t)$ reaches δ (for the sake of readability and to avoid confusion, we will use the symbol c_t instead of ϵ_t thereafter). Once found, that sequence ensures that for any cumulative sampling with sample mean μ_t, the event $\mathcal{E} = \{|\mu_t - \mu| \leq c_t \text{ for all } t \in \mathbb{N}_0\}$ happens with a probability $1 - \delta$. The key and difficult step is then to design a stopping criterion based on those parameters c_t, one that will only put an end to the sampling process if the current sample mean μ_t is an (ϵ, δ)-estimation of μ (as soon as it is the case, and not before), given that $\mathcal{E}$ holds. This is the exact problem that our main result solves: Theorem 3.1 shows how to systematically turn a decreasing sequence of precision levels $\{c_t\}_{t\in\mathbb{N}_0}$ with the right properties into a valid stopping criterion for a given concentration inequality function.

Theorem 3.1. *Let X be a real-valued random variable with $\mu = E(X) \neq 0$, and let F_X be a concentration inequality function for X. Let be $\epsilon > 0$ and $\delta > 0$.*

If there exist a sequence $(c_t)_{t\in\mathbb{N}_0}$ that is decreasing, with $\lim_{t\to\infty} c_t = 0$, and

such that for all $t \in \mathbb{N}_0$*:*

$$F(t, tc_t) = \frac{\delta}{t(t+1)}$$

Then the Generalized Stopping Algorithm, $GSA(X, \epsilon, \delta)$*, outputs an* (ϵ, δ)*-estimation of* μ*, and does so with a number of steps* n *that is minimal with probability* $1 - \delta$*.*

Algorithm 3 Generalized Stopping Algorithm

Input:
random variable X, precision ϵ and confidence level δ
Output:
$\hat{\mu}$, an (ϵ, δ)-estimation of $\mu = E(X)$

$t \leftarrow 1$
sample $\leftarrow \{x_1\}$
$\hat{\mu}_1 \leftarrow x_1$
while $\neg \left(|\hat{\mu}_t| \geq c_t(\frac{1}{\epsilon} + 1)\right)$ **do**
    $t \leftarrow t + 1$
    sample $\leftarrow$ sample $\cup \{x_t\}$
    $\hat{\mu}_t \leftarrow \frac{1}{t} \sum_{i=1}^{t} x_i$
    update c_t
return: $\hat{\mu}_t$

Proof. To shorten the proof and lighten the notations, we make the unneeded assumption that X is nonnegative. The general case is only a matter of dealing with additional computational considerations because of absolute values.

We have to bound the sum of the two following probabilities by δ:

1. The probability that the stopping criterion is reached at a step t while μ_t is not at a distance of at most $\mu\epsilon$ of μ, *i.e.* the probability of a false positive.

2. The probability that the stopping criterion is **not** reached at a step t while μ_t **is** at a distance of at most $\mu\epsilon$ of μ, *i.e.* the probability of a false negative.

Let n be the smallest integer such that:

$$c_{n-1} > \mu\epsilon \quad \text{and} \quad c_n \leq \mu\epsilon$$

($c_{n-1} > |\mu|\epsilon$ and $c_n \leq |\mu|\epsilon$ in the general case)

We want to show that the algorithm ends at step n with high probability. Note that the integer n has to exist since the sequence $(c_t)_{t\in\mathbb{N}_0}$ is decreasing, $\lim_{t\to\infty} c_t = 0$, and $\mu \neq 0$. Keep in mind that those two inequalities imply that $\frac{c_{n-1}}{\epsilon} > \mu$ and $\frac{c_n}{\epsilon} \leq \mu$.

1. First, let us show that the probability of a false postive, P_{f^+}, is smaller than δ. First, note that if P_t is the probability that the stopping criterion is reached at step t, we have:

$$P_{f^+} \leq \sum_{1\leq t<n} P_t$$

By the definition of the stopping criterion of the algorithm:

$$P_t \leq P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq c_t\left(\frac{1}{\epsilon}+1\right)\right)$$

However:

$$P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq c_t\left(\frac{1}{\epsilon}+1\right)\right) = P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq c_t\left(\frac{1}{\epsilon}\right)+c_t\right)$$

Since $c_1 \geq ... \geq c_t \geq ... \geq c_{n-1} \geq \mu\epsilon$:

$$P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq c_t\left(\frac{1}{\epsilon}+1\right)\right) \leq P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq \mu\epsilon\left(\frac{1}{\epsilon}\right)+c_t\right)$$

Therefore:

$$P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq c_t\left(\frac{1}{\epsilon}+1\right)\right) \leq P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq \mu+c_t\right)$$

In conclusion:

$$P\left(\frac{1}{t}\sum_{i=1}^{t} X_i \geq c_t\left(\frac{1}{\epsilon}+1\right)\right) \leq P\left(\frac{1}{t}\sum_{i=1}^{t} X_i - \mu \geq c_t\right)$$

Now, remember that F_X is a concentration inequality function for X (Proposition 3.1):

$$\begin{aligned} P_{f^+} &\leq \sum_{1\leq t<n} P_t \leq \sum_{1\leq t<n} P\left(\frac{1}{t}\sum_{i=1}^{t} X_i - \mu > c_t\right) \\ &\leq \sum_{1\leq t<n} F(t, tc_t) \leq \sum_{1\leq t<n} \frac{\delta}{t(t+1)} \end{aligned}$$

Finally:

$$P_{f^+} \leq \sum_{1 \leq t < n} \frac{\delta}{t(t+1)} \leq \sum_{1 \leq t} \frac{\delta}{t(t+1)} \leq \delta \left(1 - \frac{1}{n}\right)$$

2. Now, let us show that the probability of a false negative, P_{f^-}, is smaller than δ. Because of the definition of the stopping criterion, we have:

$$P_{f_-} \leq P\left(\frac{S_n}{n} \leq c_n \left(\frac{1}{\epsilon} + 1\right)\right)$$

However:

$$P\left(\frac{S_n}{n} \leq c_n \left(\frac{1}{\epsilon} + 1\right)\right) \leq P\left(\frac{S_n}{n} \leq \frac{c_n}{\epsilon} + c_n\right)$$
$$\leq P\left(\frac{S_n}{n} \leq \mu + c_n\right) \leq P\left(\frac{S_n}{n} - \mu \leq c_n\right)$$

Since F_X is a concentration inequality function for X:

$$P_{f_-} \leq P\left(\frac{S_n}{n} - \mu \leq c_n\right) \leq F(n, nc_n) \leq \frac{\delta}{n(n+1)}$$

3. In conclusion, if $P_{f_-^+}$ is the probability to have a false positive or a false negative:

$$P_{f_-^+} \leq P_{f^+} + P_{f_-} \leq \delta \left(1 - \frac{1}{n}\right) + \frac{\delta}{n(n+1)} \leq \delta$$

□

From the proof of the second claim of Theorem 3.1 directly follows Corollary 3.1, which is the generalization of the corresponding results bounding the size of the samples produced by AdaSelect and EBStop mentioned in Section 3.2.

Corollary 3.1. *With the setting of Theorem 3.1, the GSA algorithm requires a sample size of at most n with probability $1 - \delta$, with n being the smallest integer such that $c_n \leq \mu\epsilon$.*

Given a concentration inequality function, it is relatively easy to find a sequence $(c_t)_{t \in \mathbb{N}_0}$ which verifies the conditions of Theorem 3.1 to generate a specialized version of the GSA algorithm. As hinted by Proposition 3.1, it generally suffices to invert (with respect to ϵ) the function $F(t, t\epsilon)$ around its image $\delta/(t(t+1))$, for all $t \in \mathbb{N}_0$. For instance, with Hoeffding's inequality, this is a straightforward computation which can even be completed symbolically: with R the range of X, $\exp\left(-2(tc_t)^2/tR^2\right) = \delta/(2t(t+1))$. If we isolate c_t, we obtain the formula $c_t = R\sqrt{((1/2t)\ln((t(t+1))/\delta))}$, which is exactly the formula for the parameters of the stopping criterion of AdaSelect.

When an explicit expression for the parameters c_t can be derived, Corollary 3.1 can additionally be used to provide an explicit bound on the size of the generated samples, similar to those given for AdaSelect and EBStop in Section 3.2. However, having access to an explicit formula for those parameters is not absolutely needed to make use of the GSA algorithm, for they can always be computed numerically if needed (*e.g.* this would be necessary to find the parameters c_t associated with Benett's inequality). Moreover, the computation of those parameters can be performed in parallel to the generation of the simulations. Furthermore, once they have been determined (up to a certain point), they can be stored and reused afterwards (even with other values for the precision ϵ, as they only depend on δ and the properties of X).

Lastly, depending on the concentration inequality function, it might happen that the functions $F(t, t\epsilon)$ are only decreasing bijections (with respect to ϵ, for a fixed t) when restricted to a small enough interval around 0, and only for large enough t. In that case, the sequence of parameters c_t might need to start at an index $t_* > 1$ rather than with index 1. However, this purely technical problem can only arise when considering very large ϵ and computing estimations which require very small sample sizes. Initializing the SA with a sample of size t_* (rather than 1) and starting with the t_*-th iteration of the main loop is the easiest technical solution to that problem. Its only downside is to make the algorithm slightly overshoot in the case of the most trivial estimations.

3.5 Additional Optimizations

Based on the literature and our own experience with implementing multiple SAs in different contexts, we can suggest two general optimizations which can easily be applied to the algorithms produced by Theorem 3.1.

3.5.1 Geometric Sampling and Bilateral Testing

The basic Generalized Stopping Algorithm can be improved and optimized in the same way that the basic AdaSelect and the basic EBStop algorithms can be upgraded to the geometric AdaSelect and the EBGStop algorithms. The underlying optimization is based on the idea that checking if the stopping criterion has been reached at every step of the execution of the algorithm is rarely useful, and serves no purpose as long as that stopping criterion is far from being verified. A geometric progression in how the sample is grown can instead be used, with $\lceil g^t \rceil$ elements added at step t, for some $g > 1$. On one hand, choosing a large geometric parameter g can significantly speed up the algorithm, especially when large samples are needed and simulation times are low. Moreover, it can also significantly lower the number of parameters c_t

which needs to be computed. On the other hand, it can have the undesirable consequence of forcing the algorithm to generate slightly too many samples, by an order of the multiplicative factor g.

Additionally, a bilateral reformulation of the stopping criterion of the Generalized Stopping Algorithm, similar to the bilateral reformulation of the stopping criterion of EBStop can be implemented. With $\mathrm{LB}_t = \max(0, \max_{1 \leq s \leq t}(|\mu_s| - c_s))$ and $\mathrm{UB}_t = \min_{1 \leq s \leq t}(|\mu_s| + c_s)$, the stopping criterion can be rewritten as $(1+\epsilon)\mathrm{LB}_t \geq (1-\epsilon)\mathrm{UB}_t$ (see [MSA08] for details). By forcing an absolute values into the stopping criterion, this can make it easier to deal with random variables that are not necessarily nonnegative.

3.5.2 Welford's online algorithm

Every time the parameters c_t of the stopping criterion of the Generalized Stopping Algorithm involve the (empirical) variance, successive computations of both the mean and the variance (or standard deviation) of a growing sample must be performed. In the context of the verification of cyber-physical systems, the sample size can reach the thousands or the millions, which can make those almost identical computations from one step to the next become quite time-consuming in the long run.

Thankfully, there exists algorithms to update the mean and the variance of a sample $\{x_t\}_{1 \leq t \leq n}$ when a new element x_{n+1} is added. In the case of the sample mean $\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, an elementary solution is to use the formula:

$$\bar{x}_{n+1} = \frac{n\bar{x}_n + x_{n+1}}{n+1} = \bar{x}_n + \frac{x_{n+1} - \bar{x}_n}{n+1}$$

The equivalent for the update of the biased and unbiased estimators for the variance, $\mathcal{V}_n = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2$ and $V_n = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2$, is slightly less known. Moreover, one must actually be cautious in how the computations are performed as to avoid any risk of catastrophic cancellation.

The standard implementation of the classic solution is shown with Algorithm 4, with M_n defined as $\sum_{i=1}^{n}(x_i - \overline{x}_n)^2$. This algorithm was discovered by Welford [Wel62; Knu14] in 1962 and it has been widely analyzed since then [CGL83; Lin74].

Algorithm 4 Welford's algorithm

Input: currrent_data, new_value
Output: new_data

1: (t, μ, M) = current_data
2: $t = t + 1$
3: Δ = new_value $- \mu$
4: $\mu = \mu + \Delta / t$
5: Δ_2 = new_value $- \mu$
6: $M = M + \Delta * \Delta_2$
7: **return:** (t, μ, M)

At any point, the variance and the sample variance can be computed safely with the formulas M/t and $M/(t-1)$.

3.6 Application of the GSA Theorem

To analyze the applicability and the performance of the Generalized Stopping Algorithm, we decided to focus on the setting of estimating "extreme" probabilities, *i.e.* probabilities very close to 0 or 1. This is called the rare event problem.

As mentioned in Chapter 2, a common SMC strategy to check whether a system verifies one of its requirements is to estimate the probability p that the property will hold true for a random execution or specification of the system.

Since only the success or the failure of the system for each simulation matters, the complexity of the (model of) the system does not matter for any estimation algorithm whose operation is independent of the way each of those simulations is performed and deemed a success or a failure. The verification task is thus reduced to the estimation of the expected value of a random variable X which follows a Bernoulli distribution. More precisely, to facilitate the comparison with existing methods, we focused on the task of estimating $p = \mu = E(X)$ with $X \simeq \mathcal{B}(p)$, with absolute precision ϵ and a confidence level of δ.

The rare event problem has already been widely studied over the years. Importance sampling [Kah50; KM53] and importance splitting [JLS13; KH51; KM53; RR55] are the two classic approaches. Importance sampling corresponds to the modification of the distribution of the random variable whose expected value must be estimated, *i.e.* the introduction of a bias towards successful or failing simulations that is compensated later on. Importance splitting corresponds to the reformulation of the property which needs to be verified into several more likely properties, so that the probability of the rare

event can eventually be estimated as the product of their intermediate probabilities.

As SAs are nothing but statistical algorithms that minimize the sample size without trying to optimize how the samples are produced or how the parameters are estimated, one should not expect what we expose below to outperform those advanced strategies by itself. However, specifically because SAs are agnostic with respect to those processes, they could actually be combined with importance sampling and importance splitting to improve their effectiveness. This has not been implemented yet.

3.6.1 A Stopping Algorithm for Bernoulli Distributions

We compared a basic Monte Carlo approach and a standard sample size estimation method used in multiple SMC tools with AdaSelect, EBStop and three variants of GSA: GSA_H, GSA_B and GSA_C. Those three algorithms were respectively obtained from Hoeffding's inequality, Bernstein's inequality and a specialized version of the Chernoff bound.

The Monte Carlo algorithm MC uses Chebyshev's inequality to determine *a priori* the sample size it needs to compute an estimation of p with an absolute precision ϵ and a confidence level δ. For a sample of size n, Chebyshev's inequality ensures that $P(|S_n/n - E(S_n/n)| \geq \epsilon) \leq \frac{V(S_n/n)}{\epsilon^2}$. Since X follows a Bernoulli distribution, $V(X) \leq 1/4$. Therefore, as $V(S_n/n) \leq 1/4n$, a naive guess for the minimal sample size that is required for the estimation is the smallest integer n such that $1/4n\epsilon^2 \leq \delta$, *i.e.* such that $1/4\delta\epsilon^2 \leq n$.

For a less naive point of comparison, we considered the sample size estimation formula that is derived from a concentration inequality which was designed for Bernoulli distributions [Oka59b]: $n \geq \ln\left(\frac{2}{\delta}\right)/(2\epsilon^2)$. It is for instance implemented in PRISM as the APMC (Approximate Probabilistic Model Checking) method [Hér+04] and used in PLASMA (Platform for Learning and Advanced Statistical Model checking Algorithms) [LST16b] for the Smart Sampling algorithm [DAr+15b; PLC23].

GSA_H, GSA_B are theoretically equivalent to AdaSelect and EBStop. To find a true challenger, we therefore needed to start with a concentration inequality that holds for the random variables which follow a Bernoulli distribution specifically, ideally one which depends on the variance. It turns out that the Chernoff bound can be specialized and simplified for that class of simple random variables, in multiple ways [CL06; DAr+15b; Oka59b]. In particular, we used the following version [CL06] to build GSA_C: if $X_1, ..., X_n$ are iid to X with $0 \leq |X| \leq 1$, then for any $0 \leq k \leq 2\sigma_n$: $P\left(|S_n - E(S_n)| \geq k\sigma_n\right) \leq 2\exp(-k^2/4)$ (with $S_n = X_1 + ... + X_n$) and $\sigma_n = \sigma(S_n)$). For any $\epsilon \leq 2(\sigma_n)^2$, this gives us the concentration inequality:

$$P\left(|S_n - E(S_n)| \geq \epsilon\right) \leq 2\exp(-\epsilon^2/4(\sigma_n)^2)$$

Theorem 3.1 applied with the associated concentration inequality function produces a stopping algorithm GSA_C whose parameters c_t are of the form $(2\sigma_t/t)\sqrt{\ln(t(t+1)/\delta)}$, *i.e.* $(2\sigma(X_t)/\sqrt{t})\sqrt{\ln(t(t+1)/\delta)}$. This shows that GSA_C can take advantage of a low variance like GSA_B and EBStop, but that its parameters decrease at a faster rate.

Finally, we added small optimizations to the implementations of the SAs. First, as the goal is to estimate extreme probabilities, we force the algorithms to generate simulations at the start until they can initialize with a sample containing at least one success and one failure. Second, we estimate in parallel $E(X)$ and $E(1-X)$ to deal with probabilities close to 0 just as well as with probabilities close to 1 without making any assumption about the extreme nature of p. This is justified by the fact that, if $\hat{p}$ is an (ϵ, δ)-estimation of p, since $p \leq 1$, we automatically have $P(|p - \hat{p}| < \epsilon) \geq 1 - \delta$.

3.6.2 Results

Table 3.1, Table 3.2 and Table 3.3 show the order of magnitude of the (average) sample sizes for six (ϵ, δ) configurations with ϵ, δ and p ranging from 10^{-3} to 10^{-6} (and down to 10^{-9} for δ). As the results for GSA_H and GSA_B were very similar to those of AdaSelect and EBStop, we do not report them.

The SAs scale proportionally to $1/\epsilon$ and proportionally to $\ln(1/\delta)$. This is consistent with the theoretical bounds for AdaSelect and EBStop (see Section 3.2) and the structure of the stopping criterion of Theorem 3.1. All SAs require much smaller samples than the Monte Carlo method. AdaSelect clearly suffers from not being able to exploit the small variances. GSA_C outperforms EBStop in all cases, except when the probability to estimate (and thus the variance) is relatively large with respect to ϵ. When it matters the most however, with very small ϵ, δ and p, the reduction in sample size that GSA_C offers is substantial (−43%). The parallelization of the two algorithms appears to be the optimal strategy, an idea that we develop in Chapter 5.

Table 3.1: Average sample sizes for MC, AdaSelect (GSA_H), EBStop (GSA_B) and GSA_C

$\delta = 0.001$, $p = 0.001$

ϵ	MC	APMC	AdaSelect	EBStop	GSA_C
10^{-3}	$2.5 \cdot 10^8$	$3.8 \cdot 10^6$	$2 \cdot 10^7$	$2.3 \cdot 10^5$	$1.2 \cdot 10^5$
10^{-4}	$2.5 \cdot 10^{10}$	$3.8 \cdot 10^8$	$2.5 \cdot 10^9$	$1 \cdot 10^7$	$1.6 \cdot 10^7$
10^{-5}	$2.5 \cdot 10^{12}$	$3.8 \cdot 10^{10}$	$> 10^{10}$	$1 \cdot 10^9$	$1.7 \cdot 10^9$

Table 3.2: Average sample sizes for MC, AdaSelect (GSA_H), EBStop (GSA_B) and GSA_C

$\epsilon = 0.001$, $p = 0.001$

δ	MC	APMC	AdaSelect	EBStop	GSA_C
10^{-3}	$2.5 \cdot 10^8$	$3.8 \cdot 10^6$	$2 \cdot 10^7$	$2.3 \cdot 10^5$	$1.2 \cdot 10^5$
10^{-6}	$2.5 \cdot 10^{11}$	$7.2 \cdot 10^6$	$2.4 \cdot 10^7$	$2.7 \cdot 10^5$	$1.4 \cdot 10^5$
10^{-9}	$2.5 \cdot 10^{14}$	$1 \cdot 10^7$	$2.8 \cdot 10^7$	$3 \cdot 10^5$	$1.9 \cdot 10^5$

Table 3.3: Average sample sizes for MC, AdaSelect (GSA_H), EBStop (GSA_B) and GSA_C

$x = \epsilon = \delta = p$

x	MC	APMC	AdaSelect	EBStop	GSA_C
10^{-3}	$2.5 \cdot 10^8$	$3.8 \cdot 10^6$	$2 \cdot 10^7$	$2.3 \cdot 10^5$	$1.2 \cdot 10^5$
10^{-4}	$2.5 \cdot 10^{11}$	$5 \cdot 10^8$	$2.6 \cdot 10^9$	$2.5 \cdot 10^6$	$1.5 \cdot 10^6$
10^{-5}	$2.5 \cdot 10^{14}$	$6.1 \cdot 10^{10}$	$> 10^{10}$	$2.3 \cdot 10^7$	$1.7 \cdot 10^7$
10^{-6}	$2.5 \cdot 10^{17}$	$7.3 \cdot 10^{12}$	$> 10^{11}$	$3.5 \cdot 10^8$	$2 \cdot 10^8$

3.7 A Bound for the Effectiveness of Stopping Algorithms

When looking at the rate at which the parameters c_t for the stopping criterion of AdaSelect, EBStop (and GSA_C) decay, one can observe that they all do so proportionally to $\sqrt{\frac{\ln(t)}{t}}$. Is there a way to do better, that is, to find a concentration inequality such that the SA produced by Theorem 3.1 will require fundamentally smaller sample sizes for at least some kind of interesting class of random variables?

To answer that question, we complete the link between SAs and concentration inequalities by proving a result which can be seen as the reciprocal of Theorem 3.1.

Theorem 3.2 (Inverse GSA Theorem). *Let $\mathcal{A}$ be an estimation algorithm for the expected value of random variables belonging to a class of real-valued random variables $\mathbb{X}$. Let us suppose that for a precision $\epsilon > 0$ and a confidence level $\delta > 0$, for any random variable $X \in \mathbb{X}$, the algorithm $\mathcal{A}$ will generate a (minimal) sample of size at most $N_{\mathcal{A}}(\epsilon, \delta)$ with probability $1 - \delta$ to produce an (ϵ, δ)-estimation $\hat{\mu}$ of $\mu = E(X)$.*

Then $F(n,\epsilon) = N_{\mathcal{A}}^{-1}(\epsilon/n,\cdot)(n)$*, with* $N_{\mathcal{A}}^{-1}(\epsilon/n,\cdot)(n)$ *defined as any* $\delta > 0$ *such that* $N_{\mathcal{A}}(\epsilon/n,\delta) = n$*, is a valid concentration inequality function for all* $X \in \mathbb{X}$*.*

Proof. First, let us give some precisions related to the defition of $F(n,\epsilon) = N_{\mathcal{A}}^{-1}(\epsilon/n,\cdot)(n)$. We have to explain why, or rather in which cases, this function is well-defined.

For a class of non-trivial random variables $\mathbb{X}$ and a non-trivial algorithm $\mathcal{A}$, one should expect $N_{\mathcal{A}}(\epsilon,\delta)$ to always and endlessly increase as δ decreases for any fixed ϵ (the algorithm $\mathcal{A}$ should require larger and larger samples as δ gets smaller and smaller). Moreover, if for any fixed ϵ, the function $N_{\mathcal{A}}(\epsilon,\delta)$ is indeed decreasing with respect to δ and unbounded, then for any $n \in \mathbb{N}$ with $n > 1$, $N_{\mathcal{A}}^{-1}(\epsilon,\cdot)(\{n\})$ is either empty (in which case we consider $N_{\mathcal{A}}^{-1}(\epsilon,\cdot)(\{n^\diamond\})$ with $n^\diamond \in \mathbb{N}$ the largest natural number such that $n^\diamond \leq n$ and $N_{\mathcal{A}}^{-1}(\epsilon,\cdot)(\{n^\diamond\}) \neq \emptyset$) or an interval. That interval is of the form $]a,b[$, $[a,b[$, $]a,b]$ or $[a,b]$ with $0 < a < b$. In all four cases, to avoid of the use of the axiom of choice, we can take $N_{\mathcal{A}}^{-1}(\epsilon,\cdot)(n)$ as $(a+b)/2$. Note that this implies that the function $N_{\mathcal{A}}^{-1}(\epsilon,\cdot)$ hence defined is decreasing.

We now start the proof. Let us fix $X \in \mathbb{X}$. For any fixed $n^* \in \mathbb{N}_0$ and $\epsilon^* > 0$, if $X_1, ..., X_n$ are random variables which are iid to X, we have to show that:

$$P\left(\left|\sum_{i=1}^{n^*} X_i - E\left(\sum_{i=1}^{n^*} X_i\right)\right| \geq \epsilon^*\right) \leq N_{\mathcal{A}}^{-1}(\epsilon^*/n^*,\cdot)(n^*)$$

We know that for all $\epsilon > 0$, for all $\delta > 0$, if $X_1, ..., X_{N_{\mathcal{A}}(\epsilon,\delta)}$ are iid to X:

$$P\left(\left|\frac{1}{N_{\mathcal{A}}(\epsilon,\delta)}\sum_{i=1}^{N_{\mathcal{A}}(\epsilon,\delta)} X_i - \mu\right| \geq \epsilon\right) \leq \delta$$

Since $X_1, ..., X_{N_{\mathcal{A}}(\epsilon,\delta)}$ are iid to X:

$$P\left(\left|\frac{1}{N_{\mathcal{A}}(\epsilon,\delta)}\sum_{i=1}^{N_{\mathcal{A}}(\epsilon,\delta)} X_i - E\left(\frac{1}{N_{\mathcal{A}}(\epsilon,\delta)}\sum_{i=1}^{N_{\mathcal{A}}(\epsilon,\delta)} X_i\right)\right| \geq \epsilon\right) \leq \delta$$

In particular, for $\epsilon = \frac{\epsilon^*}{n^*}$:

$$P\left(\left|\frac{1}{N_{\mathcal{A}}(\frac{\epsilon^*}{n^*},\delta)}\sum_{i=1}^{N_{\mathcal{A}}(\frac{\epsilon^*}{n^*},\delta)} X_i - E\left(\frac{1}{N_{\mathcal{A}}(\frac{\epsilon^*}{n^*},\delta)}\sum_{i=1}^{N_{\mathcal{A}}(\frac{\epsilon^*}{n^*},\delta)} X_i\right)\right| \geq \frac{\epsilon^*}{n^*}\right) \leq \delta$$

If $\delta = N_{\mathcal{A}}^{-1}(\epsilon^*/n^*,\cdot)(n^*)$, δ is such that $N_{\mathcal{A}}(\epsilon^*/n^*,\delta) = n^*$ (technically, it could happen that $N_{\mathcal{A}}(\epsilon^*/n^*,\delta) = n^\diamond$ with $n^\diamond \in \mathbb{N}$ the largest natural number such

that $n^\diamond \leq n^*$ and $N_{\mathcal{A}}^{-1}(\epsilon/n^*, \cdot)(\{n^\diamond\}) \neq \emptyset$, in which case consider the same computation for $\epsilon = \epsilon^*/n^\diamond$). Therefore:

$$P\left(\left|\frac{1}{n^*}\sum_{i=1}^{n^*} X_i - E\left(\frac{1}{n^*}\sum_{i=1}^{n^*} X_i\right)\right| \geq \frac{\epsilon^*}{n^*}\right) \leq N_{\mathcal{A}}^{-1}(\epsilon^*/n^*, \cdot)(n^*)$$

Which implies:

$$P\left(\left|\sum_{i=1}^{n^*} X_i - E\left(\sum_{i=1}^{n^*} X_i\right)\right| \geq \epsilon^*\right) \leq N_{\mathcal{A}}^{-1}(\epsilon^*/n^*, \cdot)(n^*)$$

□

In conclusion, not only it is possible to turn concentration inequalities into SAs with Theorem 3.1, it is also possible to turn a SA into a concentration inequality for the class of random variables it can be applied to. This means that if we have a limit on how good concentration inequalities can be, we can deduce a bound on the effectiveness of SAs.

To determine a bound for the speed at which concentration inequality functions can decay, it is useful to remember that the goal of this work is to verify cyber-physical systems with SMC. In such a context, we can rely on the fact that the realizations of the random variables correspond to independent simulations of a fixed system.

Therefore, we can rely on the central limit theorem (see Section 2.1.1) to get an exact description of the asymptotic behavior of the sum of iid random variables.

Theorem 3.3 (Bound for the decay of concentration inequality functions)**.** *Let $X_1, X_2, X_3, \ldots$ be random variables iid to a random variable X with (finite) expected value $\mu = E(X)$ and (finite) variance $\sigma^2 = V(X) \neq 0$. Let F_X be a concentration inequality function for X.*
Then, for any $\epsilon > 0$, $F(n, n\epsilon)$ cannot decay faster than $erfc\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)$ with respect to n, with $erfc(x) = 1 - \operatorname{erf}(x)$ complementary error function.

More precisely, this means that the sequence $\left(\frac{erfc\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)}{F(n,n\epsilon)}\right)_{n\in\mathbb{N}_0}$ is bounded and that it is impossible to have:

$$\lim_{n\to\infty} \frac{erfc\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)}{F(n, n\epsilon)} = +\infty$$

Proof. The central limit theorem tells us that as n grows, the random variable $\left(\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right)$ can be approximated with a random variable $Z \sim \mathcal{N}(0, n\sigma^2)$. As n becomes larger and larger, for all $\epsilon > 0$:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \simeq P\left(|Z| \geq \epsilon\right)$$

That is:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \simeq \int_{-\infty}^{-\epsilon} \frac{1}{\sqrt{2\pi n\sigma^2}} e^{\left(\frac{-t^2}{2n\sigma^2}\right)} \mathrm{dt} + \int_{\epsilon}^{\infty} \frac{1}{\sqrt{2\pi n\sigma^2}} e^{\left(\frac{-t^2}{2n\sigma^2}\right)} \mathrm{dt}$$

Since the integrand is an even function:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \simeq 2\int_{-\infty}^{-\epsilon} \frac{1}{\sqrt{2\pi n\sigma^2}} e^{\left(\frac{-t^2}{2n\sigma^2}\right)} \mathrm{dt}$$

The integral on the right is the cumulative distributive function of Z. A few substitution show that:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \simeq 1 + \mathrm{erf}\left(\frac{-\epsilon}{\sigma\sqrt{2n}}\right)$$

Since the erf function is odd:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \simeq 1 - \mathrm{erf}\left(\frac{\epsilon}{\sigma\sqrt{2n}}\right)$$

With the complementary error function, defined as $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$, this becomes:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq \epsilon\right) \simeq \mathrm{erfc}\left(\frac{\epsilon}{\sigma\sqrt{2n}}\right)$$

Equivalently:

$$P\left(\left|\sum_{i=1}^{n} X_i - E\left(\sum_{i=1}^{n} X_i\right)\right| \geq n\epsilon\right) \simeq \mathrm{erfc}\left(\frac{n\epsilon}{\sigma\sqrt{2n}}\right)$$

Therefore:

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \simeq \mathrm{erfc}\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right) \tag{3.2}$$

By Proposition 3.1, we have:

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \leq F(n, n\epsilon)$$

So $F(n, n\epsilon)$ cannot decay faster than $\operatorname{erfc}\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)$ with respect to n. □

Note that Equation (3.2) tells us what we should expect for the asymptotic behavior of an optimal concentration inequality function in terms of the sample size n and the precision ϵ as well. Additionally, it highlights and quantifies how a smaller variance can be exploited to minimize the sample size of an estimation.

To conclude, we need to make the formula $\operatorname{erfc}\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)$ slightly more explicit. Thankfully, the complementary error function is a function whose asymptotic behavior has been studied extensively [AS65; Tem10]. One can show that the following holds:

$$\lim_{x\to\infty} \frac{\operatorname{erfc}(x)}{e^{-x^2}/x\sqrt{\pi}} = 1$$

Therefore, for any fixed $\epsilon > 0$ and $\sigma > 0$, as n grows:

$$\operatorname{erfc}\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right) \simeq \frac{e^{-\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)^2}}{\left(\frac{\epsilon\sqrt{n}}{\sigma\sqrt{2}}\right)\sqrt{\pi}} = \frac{\sigma\sqrt{2}e^{-\frac{n\epsilon^2}{2\sigma^2}}}{\epsilon\sqrt{\pi n}} \tag{3.3}$$

Equation (3.3) not only tells us how an optimal concentration inequality function that is valid for SMC should asymptotically behaves with respect to the number of simulations, but also with respect to the precision. With respect to the number of simulations, most of the decay comes from the exponential $e^{-\frac{n\epsilon^2}{2\sigma^2}}$, which (unsurprisingly) corresponds to the form of the concentration inequality functions associated to Hoeffding's inequality and Bernstein's inequality of the AdaSelect and EBStop algorithms. Furthermore, it corresponds to the form of the concentration inequality function we exploited in section Section 3.6.1 to build a new SA specifically designed to tackle the problem of rare events with Theorem 3.1 as well.

Now note that Theorem 3.2 does not require the algorithm $\mathcal{A}$ to be a SA. The key assumption is that the random variables are iid, which in practice means that the sampling process (the system simulations) must be performed in an independent fashion. For SMC methods, this is generally a *de facto* requirement. What entails is that the asymptotic bound of Theorem 3.3 apply to **all possible estimation algorithms for SMC**, and not just to SAs. This implies that no other estimation algorithm will ever be able to asymptotically

outperform the optimal SAs we discussed in this chapter and those that can be obtained as specializations of the Generalized Stopping Algorithm. Moreover, since the estimations realized in the context of SMC often require high precision (*i.e.* small values for ϵ) and huge samples, this makes this asymptotic result particularly relevant: in the long run, for the most demanding use cases, an optimal SA will always be the best choice.

3.8 Conclusion

Adaptive stopping algorithms are a great alternative to Monte Carlo approaches for SMC methods. By minimizing the number of system simulations which need to be generated to compute estimations, they can not only make those methods faster but scale better as well. Moreover, as they are purely statistical algorithms, they have no impact or requirement on how the samples are produced, and can therefore be combined with other specialized SMC algorithms for the verification of particular classes of systems.

In this chapter, we generalized two optimal stopping algorithms, AdaSelect and EBStop, which allowed allowed us to provide a systematic method to generate tailor-made stopping algorithms from concentration inequalities. We showed how our main contribution, Theorem 3.1, can be exploited to improve the efficiency of SMC methods for specific settings, such as the rare event problem. Finally, we analyzed and discussed the theoretical limits of adaptive stopping algorithms in the context of SMC, highlighting the optimality of our results.

Future work We see multiple possible paths of exploration to expand this work. From a theoretical perspective, improving the structure of Algorithm 3 and optimizing its geometric version seems to be a promising prospect. For example, if the result used to minimize the drawback of geometric sampling in the case of EBStop [AMS07b] could be generalized, this could improve the performance of the geometric version of GSA. Theorem 3.1 could also maybe be duplicated for the estimation of the variance of random variables, rather than their expected value. The Efron–Stein inequality could be a good starting point.

In terms of application, we could of course specialize GSA for other classes of random variables, again hoping that the generated SAs would outclass the competitors. More ambitiously, we want to determine whether it would be possible to automatize to process of deriving a system-specific concentration inequality directly from (the exploration of) the structure of the model of that system, which would result in a tool able to produce tailor-made SAs for individual systems without any input needed from the user.

Optimization Algorithms | 4

This chapter discusses existing optimization strategies for SMC. It presents some of the ideas and challenges which have been analyzed in the literature, it looks into the details of a specific broad-spectrum and lightweight optimization algorithm for SMC and shows how it is possible to improve it, and lastly it highlights the limitations of elementary approaches and develop a new framework to suggest an avenue worth exploring to solve the issue of local extrema in the context of SMC.

The content of this chapter is related to the conference publication titled "Optimized Smart Sampling":
M. Parmentier, A. Legay, and F. Chenoy. "Optimized Smart Sampling". In: *International Conference on Bridging the Gap between AI and Reality*. Springer. 2023, pp. 171–187.

4.1 Introduction

Estimation algorithms are the bread and butter of SMC, but the most advanced and interesting SMC methods are often designed to achieve greater goals than simple estimations. Be it to verify the worst-case scenario of a system during conception or to optimize the behavior of a CPS when deployed, SMC can help to solve difficult optimization problems.

It might seem strange to rely on SMC for optimization at first. SMC methods never parse the whole exploration space by design, they are stochastic and their outputs only come with statistical guarantees. However, the reason why SMC is often needed in the first place, the state-explosion problem, generally becomes even more prevalent for optimization problems. Moreover, the unperfect quality of the outputs of SMC algorithms is not actually that much of an issue when trying to solve optimization problems in practice, as for real-world applications a relatively cheap and easy to obtain excellent solution is almost always preferred to a perfect but extremely costly one.

SMC has been used to deal with optimization problems in contexts as varied as cloud workflow resource allocation [Che+16] or the maintenance analysis of a train pneumatic compressor [Rui+16]. The methods developed depend heavily on the framework used to represent the CPS.

A series of recent works [DAr+15a; Jae+19] has made it possible to extend some SMC optimization algorithms to the case of MDPs. Most of these algorithms use deep learning techniques that are very effective but require knowledge of the system and often complex modeling of the schedulers to be analyzed.

For MDPs, the optimization problem can be expressed as follows:

Problem 4.1. *Given a MDP* $(S, s_0, A, \{P_a\}_{a\in A})$*, a BLTL formula* ϕ *and a specific class of schedulers* Π*, how to find a* optimal *scheduler* π^+ *or* π^-*, defined as:*

$$\pi^+ \in \arg\max_{\pi\in\Pi} \int_\Omega \mathbb{1}_{T\models\phi}\, d\mu_\pi \qquad\qquad \pi^- \in \arg\min_{\pi\in\Pi} \int_\Omega \mathbb{1}_{T\models\phi}\, d\mu_\pi$$

That is, a scheduler (not necessarily unique) with the greatest/lowest possible score, i.e. the greatest/lowest probability to generate traces for which the property ϕ *is true.*

The most straightforward method to solve Problem 4.1 with SMC is to first build a large number of schedulers, compute the score of each of those schedulers, then output one with the highest score [DAr+15a]. However, despite its simplicity, this approach calls for multiple remarks.

First, the right representation of schedulers must be chosen. For large models, representing all schedulers as full maps does not scale well in terms of memory space. As showed in [DAr+15a], one efficient alternative is to represent schedulers as seeds for a pseudo-random number generator, whose size can be adapted in function of the expected theoretical number of schedulers.

Second, the size and complexity of the models of real-world systems may prohibit the computation of the exact theoretical scores of the schedulers [Leg+19a], and those must then be estimated with SMC as well.

Third, the performance of such an elementary SMC algorithm is extremely dependent of the number of schedulers which are generated and whose scores are estimated. If that number is low with respect to the total number of possible schedulers for the system under scrutiny, then the chances that an optimal scheduler can be found among the schedulers generated by the algorithm is low as well. This can be (partially) solved by allocating a smaller fraction of the simulation budget of the algorithm to each evaluation of the schedulers, i.e. by producing less traces for each scheduler. Unfortunately, that pseudo-solution is useless for real-world applications of SMC, since those usually demand estimations with very high precision and confidence level.

A more sophisticated approach is the Smart Sampling algorithm [DAr+15a]. It starts with an initial budget of schedulers and applies a standard SMC estimation algorithm on the MC which results from each scheduler's choices.The fifty percents best schedulers are then kept and the operation is repeated until the scheduler for which SMC minimizes or maximizes the requirement to be validated is found.

Unfortunately, the Smart Sampling algorithm meets the same difficulties than other optimization algorithms. In particular, it suffers from the local extremum problem: it generally does not find the best scheduler but rather a scheduler which calculates the average probability of satisfying the property.

4.2 Background

The original Smart Sampling Algorithm from [DAr+15a] given in Algorithm 5 is the version of the algorithm tailored to find optimal schedulers with maximal scores. The condition on the per-iteration budget is derived from the Chernoff Bound to ensure (ϵ, δ)-estimations [Oka59a; DAr+15a].

In Step 1 (lines 1 to 7), the per-iteration budget B is used once to generate $\lceil\sqrt{B}\rceil$ schedulers (line 2) and produce $\lceil\sqrt{B}\rceil$ traces for each of those schedulers (line 5). The traces which satisfy the property ϕ are counted to compute a rough estimation of the schedulers' scores (line 6), and the greatest value among those estimations is selected as a first estimation $\hat{p}_{\max}$ for the score of a (near-)optimal scheduler. In Step 2 (lines 8 to 14), the naive estimation $\hat{p}_{\max}$ is used to balance the simulation budget so to maximize the probability of producing traces with near-optimal schedulers (line 8) [DAr+15a]. Then, the initial population of the algorithm is generated (line 9). A preliminary iteration is then performed to abandon the schedulers (lines 13) that did not produce any trace which satisfies the property ϕ (line 14). Finally, in Step 3 (lines 15 to 28), the main loop of the algorithm happens, and does not stop until the population has been reduced to one scheduler or until the confidence level that has been reached, as approximated with the Chernoff Bound (line 22), is smaller than δ (line 20). First, the scores of the schedulers are reset before the start of the main loop (line 19). At each step of the loop, the iteration population is updated and as long as the the number of simulations which have been realized at this step is smaller than $\lceil B/M_i\rceil$ (line 20), an additional trace is generated for each scheduler (lines 23 to 24). At the end of each step, the results are updated (line 25), the approximation of the optimal probability is updated and the population is cut in a way that only half of the population of schedulers, those with the best scores, are kept.

Algorithm 5 Original Smart Sampling Algorithm from [DAr+15a]

Input:
MDP $\mathcal{M}$ and BLTL formula ϕ
precision and confidence level ϵ and δ
per-iteration budget $B \geq \ln(2/\delta)/(2\epsilon^2)$
Output:
P and $\hat{p}_{\max}$, with P final population,
and $\hat{p}_{\max}$ an (ϵ, δ)-approximation of an optimal score

$N \leftarrow \lceil \sqrt{B} \rceil$; $M \leftarrow \lceil \sqrt{B} \rceil$
$P \leftarrow \{M$ schedulers sampled uniformly at random$\}$
for $\pi \in P$ **do**
  for $1 \leq i \leq N$ **do**
    $T_i^\pi \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
$R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T_i^\pi \mid T_i^\pi \models \phi\}|\}$
$\hat{p}_{\max} \leftarrow \max_{(\pi, \hat{n}_\pi) \in R} \hat{n}_\pi / N$

$N \leftarrow \lceil 1/\hat{p}_{\max} \rceil$; $M \leftarrow \lceil B\hat{p}_{\max} \rceil$
$P \leftarrow \{M$ schedulers sampled uniformly at random$\}$
for $\pi \in P$ **do**
  for $1 \leq i \leq N$ **do**
    $T_i^\pi \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
$R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T_i^\pi \mid T_i^\pi \models \phi\}|\}$
$P \leftarrow \{\pi \in P \mid \hat{n}_\pi > 0\}$

$R \leftarrow \{(\pi, 0) \mid \pi \in P\}$; $i \leftarrow 0$; confidence $\leftarrow 1$
while confidence $> \delta \wedge |P| > 1$ **do**
  $i \leftarrow i + 1$
  $M_i \leftarrow |P|$
  $N_i \leftarrow 0$
  while confidence $> \delta \wedge N_i < \lceil B/M_i \rceil$ **do**
    $N_i \leftarrow N_i + 1$
    confidence $\leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$
    for $\pi \in P$ **do**
      $T_{N_i}^\pi \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
  $R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T_i^\pi \mid T_i^\pi \models \phi\}|\}$
  $\hat{p}_{\max} \leftarrow \max_{(\pi, \hat{n}_\pi) \in R} \hat{n}_\pi / N$
  $P \leftarrow \{\pi \in P \mid \hat{n}_\pi$ is one of the greatest $\lfloor |P|/2 \rfloor$ values in $R\}$
return $P, \hat{p}_{\max}$

4.3 Algorithm Analysis

We reproduced two of the experiments of the original paper of the Smart Sampling algorithm [DAr+15a], the analyses of the IEEE 802.11 Wireless LAN Protocol (WLAN) and the Gossip Protocol (GOSSIP) [PRISM; KNP08; KNS02b].

We were interested in observing how a greater or smaller **reduction factor**, defined as the denominator of the fraction of the population which is preserved from step to step, could impact the performance of the algorithm. On one hand, increasing the reduction factor should *a priori* lead to poorer performances but a faster execution time for the algorithm: a higher number of schedulers discarded at each step means that near optimal schedulers which can be present in the initial population might be abandoned by mistake (especially at the first step), but it also means that there are fewer steps to the algorithm and that the total simulation budget will be a smaller multiple of the per-iteration budget. On the other hand, decreasing the reduction factor should *a priori* lead to better performances but a slower execution time for the algorithm: a smaller number of schedulers discarded at each step means that near-optimal schedulers which can be present in the initial population have a smaller chance to be abandoned by mistake, but it also means that there are more steps to the algorithm and that the total simulation budget will be a greater multiple of the per-iteration budget.

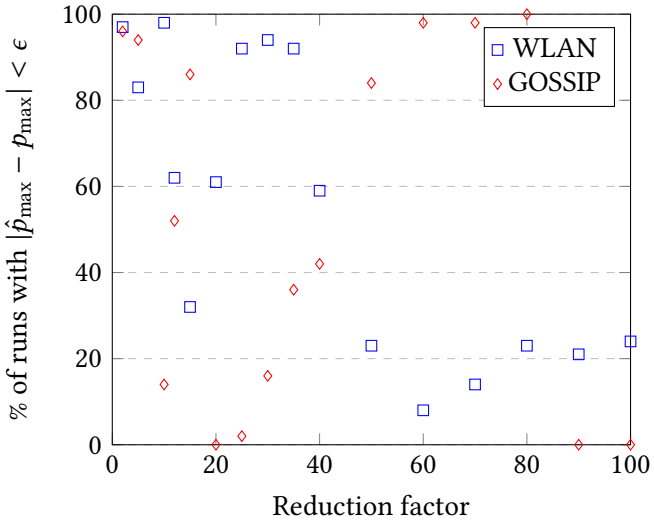
Since the WLAN and GOSSIP models have already been evaluated thoroughly with exhaustive model checking techniques [KNP11a; KNS02a; KNP], the theoretical score of an optimal scheduler is known for both of those experiments. Figure 4.1 shows the proportion of runs for the Smart Sampling algorithm that managed to output an estimation which was within the error bound of the actual optimal score in function of a varying reduction factor, with $\epsilon = 0.01$, $\delta = 0.1$ and a per-iteration budget of 15000.

Surprisingly, the expected correlations can not be found. Moreover, a performance drop phenomenon can be seen around specific values. The location of those values depends of the values of the parameters of the Smart Sampling algorithm. After analysis of the implementation and individual executions of the algorithm, our conclusion is that the original implementation of the Smart Sampling algorithm is flawed in three ways.

First, because it is designed to immediately stop as soon as the per-iteration budget divided by the number of remaining schedulers is greater than the Chernoff Bound, the algorithm can terminate much earlier than anticipated with a still diverse population of schedulers.

Second, because barely enough traces are at that point produced so that the estimations of the schedulers of the final population are valid (ϵ, δ)-estimations, most of them are badly estimated relatively to each other. This results in individual outputs for the original Smart Sampling algorithm with great variance.

Figure 4.1: Original Smart Sampling algorithm performance



Those two first flaws make for drastic changes in performance when the ratios between the size of the initial sample and powers of the reduction factor are close to integers, up to the last step of the algorithm. A slight change in one of those ratios can severely impact how soon the algorithm stops and how well the schedulers are evaluated at the last step. We hypothesize that this potential problem was not noticed with the first version of the Smart Sampling because the fixed value of 2 which was originally chosen for the reduction factor was low, making the algorithm's progress smooth in the sense that the population size decreases from step to step relatively slowly.

Third, additional tailor-made experiments showed that the original implementation of the Smart Sampling algorithm suffers from another critical weakness: noise. For models with in-built noise, i.e. for which all schedulers have a fixed low chance to produce failing traces, performance is often extremely poor with non-generous per-iteration budget. Indeed, when the ratio between the initial population size and the per-iteration is close to 1, the first evaluation of the schedulers is so vulnerable to the noise that the first wave of discarded schedulers can be essentially regarded as random. Even if near-optimal schedulers are present in the sample at the start, the probability they are abandoned before the second step of the algorithm is high.

4.4 Improved Versions of the Smart Sampling Algorithm

To improve the Smart Sampling algorithm, we modified how the Chernoff bound test dictates the termination of the algorithm, and we explored three ideas: artificially inflating the simulation budget for the first step, applying a simulated annealing strategy, and reintroducing new random schedulers at specific steps. Those modifications were implemented in *PLASMA* (Platform for Learning and Advanced Statistical Model checking Algorithms), a plug-in based interface for statistical model checking [LST16a].

Unless specified otherwise, all experiments were realized for the WLAN and the GOSSIP models, with $\epsilon = 0.01$, $\delta = 0.1$ and a per-iteration budget of 15000. For both models, since the theoretical optimal scheduler and its score are known, the goal was to compute the probability that the algorithm produces an estimation which deviates from the theoretical optimal solution by at most ϵ and check if the probability is indeed lower or equal to δ. The results, which include information about the execution times as well, are averaged over up to 100 runs (depending on the experiment). The experiments were performed on a 8 GB machine with 4 cores running at 3.2 GHz.

4.4.1 Modification of the Chernoff Bound Test

Three approaches were considered to deal with the problems arising from the potential early termination of the Smart Sampling algorithm due to the Chernoff bound test: keep the early termination but with a better evaluation of the schedulers of the final population (option 1), ignore the early termination but scale down the simulation budget with respect to the accumulated confidence level for the remaining iterations, (option 2), or completely ignore the early termination (option 3).

Option 1 leaves the main loop of Algorithm 5 intact, but adds a last loop of x simulations for each scheduler once the Chernoff bound test makes the main loop end. Then, the modified algorithm reevaluate the schedulers and order them by their scores one last time before returning the result. Option 2 requires to remove the Chernoff bound test at line 20, and to alter the condition for the simulation loop from "confidence $> \delta \wedge N_i < \lceil B/M_i \rceil$" to "$N_i < F(\text{confidence}, \lceil B/M_i \rceil)$", with F a pre-determined fixed function. Option 3 completely removes the Chernoff bound test both at line 20 and at line 24.

All three options were initially compared to each other fairly in the sense that the design choices (number x and function F) were made so that the number of additional simulations was identical between the options. For option 2, any test function F which scales linearly with confidence quickly leads to insignificant per-scheduler simulation budgets for the subsequent steps of

the algorithm. Therefore, the type of functions which was most experimented with are functions which returns $\lceil B/M_i \rceil$ as long as confidence is greater than δ, then returns (a fraction or a multiple of) the per-scheduler simulation budget of the last iteration of the main loop for which confidence was still greater than δ.

Option 1 is not only the option that doesn't fit the original philosophy of the algorithm, but it is also the option which produced the worst results. Because option 1 allocates all the additional budget evenly between the remaining schedulers of the final population, the sampling process ceases to prioritize the most promising schedulers. Option 3 produced the best results consistently, whereas option 2 produced good results as well when the multiple of the test function F was large enough. However, when the multiple of the test function was taken as 1, option 3 was always a better choice in terms of results while the difference in total simulation budgets between option 2 and option 3 was systematically small. Indeed, unless the per-iteration budget parameter is unnecessary large with respect to ϵ and δ, the Smart Sampling algorithm generally completes most of its potential steps before the Chernoff bound condition is verified. Therefore, option 3 is the option we have chosen for the modification of the Smart Sampling algorithm. A small verification can be added at the level of the preconditions to warn the user or ensure that the per-iteration budget cannot be excessively large with respect to ϵ and δ when the user specifies a per-iteration budget instead of letting the algorithm fix it by itself.

Figure 4.2 is the equivalent of Figure 4.1 for the Smart Sampling algorithm modified with option 3. Not only the modification improves the performances of the algorithm noticeably, but it eliminates the performance drop phenomenon as well. The modification is particularly impactful with high values for the reduction factor, to the point that the algorithm's performance barely diminishes as the the reduction factor increases.

Figure 4.3 and Figure 4.4 show the execution times for 100 runs of the Smart Sampling algorithm for the WLAN and GOSSIP experiments, for both versions of the algorithm.

For any given reduction factor, the execution times of the modified version of the algorithm are on average 13% greater than those for the original version of the algorithm. However, since it allows for a much larger reduction factor without a loss in performance, the modified version applied with a large reduction factor outperforms the original version (with a reduction factor of 2) both in terms of results and speed.

Figure 4.2: Modified Smart Sampling algorithm performance

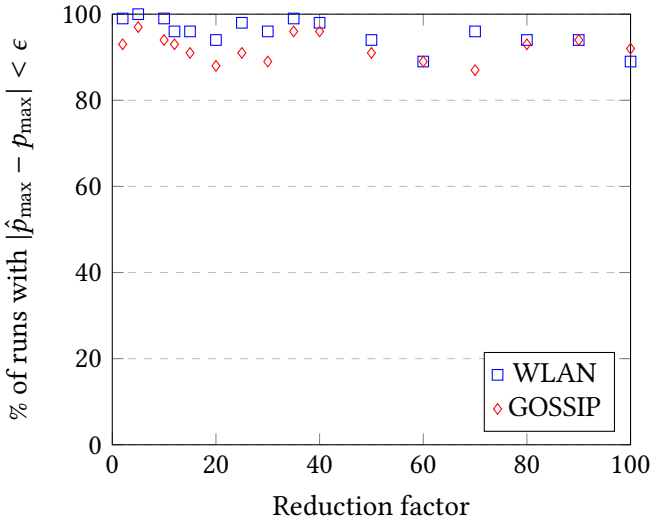


Figure 4.3: Execution times (WLAN)

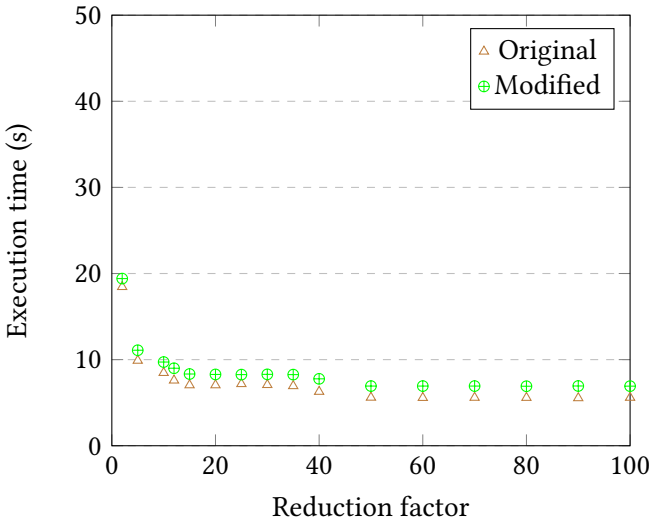
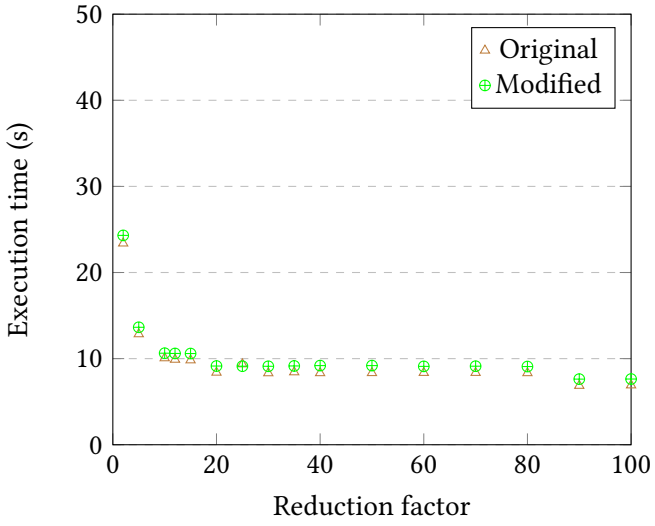


Figure 4.4: Execution times (GOSSIP)



In conclusion, it appears that the ideal new version of the Smart Sampling algorithm is the third suggested modification, applied with a reduction factor larger than 2. Since the trade-off between performance and speed is inevitable as the reduction factor increases, we cannot give a perfect universal value for the reduction factor. Nevertheless, it seems there is no reason to not take it at least as large as 5 or even 10, for we could never observe noticeable loss in performance at those values in any of our experiments, while already providing a significant speed-up to the algorithm. With a reduction factor of 5, the algorithm is accelerated by a factor of $\ln(5)/\ln(2) \simeq 2.32$. With a reduction factor of 10, the algorithm is accelerated by a factor of $\ln(10)/\ln(2) \simeq 3.32$.

4.4.2 Other optimizations

To further improve the Smart Sampling algorithm, we first looked at one of its other weaknesses discussed in Section 4.3: its fragility with respect to noise. During the first step of the algorithm, it is frequent that the algorithm drops near-optimal schedulers because their evaluation is based on so few traces that those evaluations have a high probability to be unreliable. When realistic simulation budgets are taken into account, the number of times each scheduler is simulated during the first iteration can even be close (if not equal) to 1.

Algorithm 6 Additional Budget at First Step

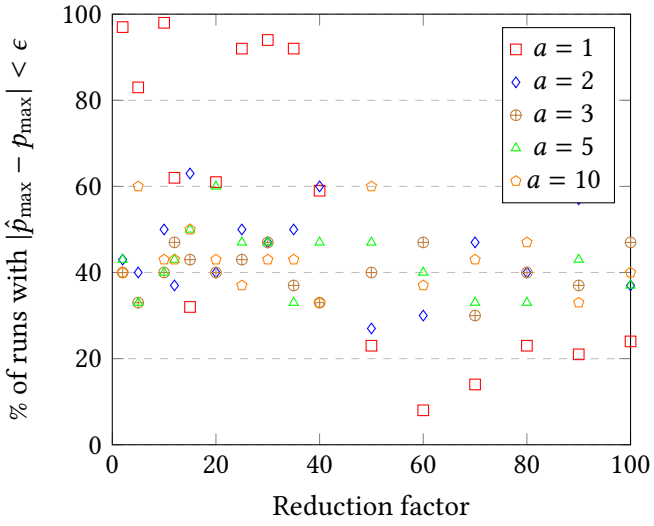
20: **while** $|P| > 1$
21: $i \leftarrow i + 1$
22: $M_i \leftarrow |P|$
23: $N_i \leftarrow 0$
24: **if** $i == 1$
25: **while** $N_i < F(\lceil B/M_i \rceil)$ ▷ F is a fixed function
26: $N_i \leftarrow N_i + 1$
27: **for** $\pi \in P$ **do**
28: $T^{\pi}_{N_i} \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
29: **end for**
30: **end while**
31: $R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T^{\pi}_i \mid T^{\pi}_i \models \phi\}|\}$
32: $\hat{p}_{\max} \leftarrow \max_{(\pi, \hat{n}_\pi) \in R} \hat{n}_\pi / N$
33: $P \leftarrow \{\pi \in P \mid \hat{n}_\pi$ is one of the greatest $\lfloor |P| / (\text{reduction factor}) \rfloor$ values in $R\}$
34: **else**
35: **while** $N_i < \lceil B/M_i \rceil$) ▷ New condition for the main loop
36: $N_i \leftarrow N_i + 1$
37: **for** $\pi \in P$ **do** ▷ New condition for the simulation loop

To counter that problem, a very simple solution was first tested: providing the algorithm with additional simulation budget for the first step. This means slightly modifying the condition of the simulation loop at line 24 of Algorithm 5 to introduce an exception for the first iteration. Algorithm 6 shows the main modifications which were brought to the code of Algorithm 5 to remove the Chernoff bound test and introduce additional sampling at the first iteration.

The kind of test function F which was mostly tested is a collection of simple multiplicative functions with a multiplicative parameter a ranging from 2 to 10. Depending on whether the chosen per-iteration budget allows for a large or small per-scheduler simulation budget at the first step, a lower or greater value should be taken for that multiplicative parameter. Indeed, if the per-scheduler simulation budget at the first step is satisfactory to begin with, picking a value as low as 2 is the right choice as to minimize the additional computing cost. However, if the per-scheduler simulation budget at the first step is close to 1, then taking it as large as 10 (or even larger) is necessary. In general, the function F can be of the form $F(x) = \min(K, ax)$, with a rather large and K being a upper bound on the per-scheduler budget for the first step, for a jack-of-all-trades solution.

On custom small scale models designed with a lot of in-built noise, providing additional simulation budget for the first iteration improved the Smart

Figure 4.5: More budget at first iteration strategy (WLAN)



Sampling algorithm's probability of identifying a (near-)optimal scheduler by around 6% with $a = 2$, 15% with $a = 3$, 26% with $a = 5$ and 30% with $a = 10$. Those numbers are average over reduction factors ranging from 2 to 100, albeit the gains in performance were generally more important with large (> 10) reduction factors.

Figure 4.5 and Figure 4.6 show the impact of providing additional simulation budget for the first iteration on the probability for the Smart Sampling algorithm to output a near-optimal scheduler for the WLAN and GOSSIP models. Despite the limited importance of the noise weakness of the Smart Sampling algorithm for those models, we can observe that even though the effect of providing additional simulation budget for the first iteration is less noticeable than when noise is involved, in the case of the GOSSIP experiment, that strategy has positive impact on the algorithm's performance nonetheless. However, as can been seen with Figure 4.6, which was produced with a reduced set of experiments of 30, that strategy still suffers from the instability of the statistical model checking process.

Of course, that small positive benefit is not worth if it comes with a prohibitive increase in computational costs. However, as shown with Figure 4.7 and Figure 4.8, the increase in computational costs is limited, at least for small values of the multiplicative factor a. Which is not a surprise: especially with low reduction factors, the number of iterations of the algorithm is generally large enough that adding the equivalent of a few artificial iterations at the

Figure 4.6: More budget at first iteration strategy (GOSSIP)

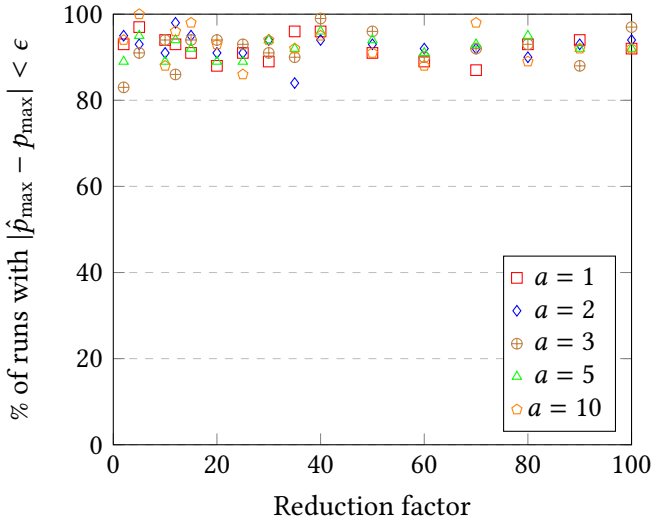
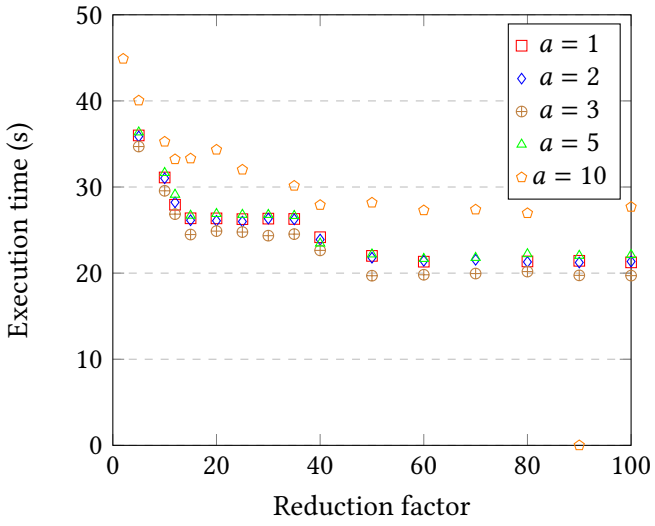
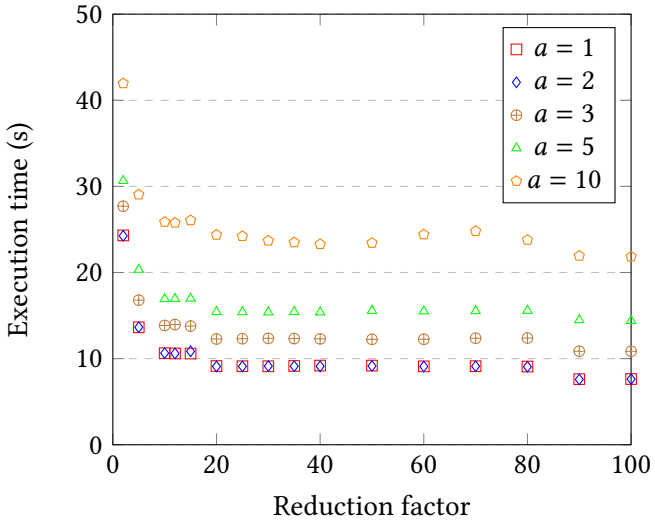


Figure 4.7: Execution times (WLAN)



start does not grow the total simulation budget by a significant margin. Across all reduction factors, the increase is negligible with $a = 2$ and $a = 3$, around 25% with $a = 5$ on average and between 30% and 150% with $a = 10$.

Figure 4.8: Execution times (GOSSIP)



We find difficult to advocate for such an increase in computational costs in any scenario, even when the model is expected to be "noisy". However, as the potential benefit in some situations is very valuable and the trade-off in terms of speed barely noticeable for small values of the multiplicative parameter a, we consider adopting the additional budget for first iteration strategy as the new baseline for the Smart Sampling algorithm the right choice. Fixing $a = 3$ appears to be a good starting point, with a suggestion to the user to push the value of that parameter up to 10 when working with noisy models.

Trying to extend the additional budget at first step strategy to a simulated annealing strategy was attempted. Instead of only inflating the simulation budget of the first iteration, one can try to artificially rebalance the total simulation budget between the different steps of the algorithm instead of allocating a constant per-iteration budget to all iterations. Skewing towards early iterations and towards late iterations were both tested. The conclusion is that skewing towards early iterations is often the best course of action, to the point that the best results were obtained when the skew was so much in favor of the first iteration that it was practically equivalent to the much simpler strategy which consists in providing additional budget at the first step.

Lastly, the idea of reintroducing schedulers as the algorithm progresses was also experimented with. The goal was to improve the Smart Sampling algorithm with respect to another of its fundamental limitations, i.e. its inability to identify schedulers beyond those present in the initial population, but also

Figure 4.9: Random scheduler reintroduction strategy (WLAN)

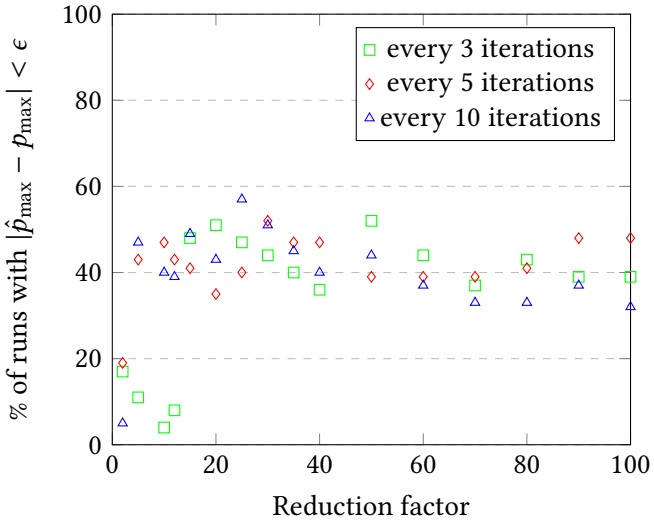
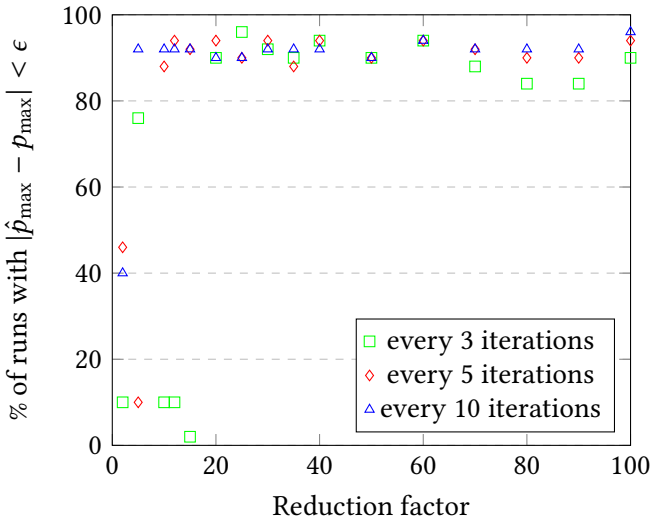


Figure 4.10: Random scheduler reintroduction strategy (GOSSIP)



reduce its weakness relatively to the local extremum phenomenon. Unfortunately, as shown in Figure 4.9 and Figure 4.10, the results are not conclusive. In all cases, that strategy did not produce better results than simply starting

with a larger initial population. In some cases, that strategy even made the algorithm's performance worse, by introducing badly evaluated schedulers with low scores right before the termination of the algorithm. We hypothesize that this kind of strategy is still promising nevertheless, but at the condition that the new schedulers are instead synthetized in a way that exploits the accumulated information about the discarded schedulers in order to reintroduce schedulers which are different and potentially have good scores, and not just random ones. We explore that idea in Section 4.5.

4.5 Genetic Algorithm for Statistical Model checking

The less impactful idea which was tried to improve the Smart Sampling algorithm was by far the one consisting in the reintroduction of random schedulers as the algorithm progresses with the removal of obviously bad schedulers from the population. This highlights how "more is not always better" for optimization algorithms, especially in the context of SMC. Even for small MDPs, the exploration space of schedulers quickly becomes so vast and intricate that we need to consider another approach: quality over quantity.

In other words, while a purely random sample might be satisfactory (and unavoidable) for the initialization of SMC optimization algorithms, it seems imperative to guide those algorithms for the generation of further samples. Similarly to how the adaptive stopping algorithms of Chapter 3 learn as they go, optimization algorithms should consider the features of the elements of their already accumulated sample to expand it.

Guiding the generation of new schedulers is an idea which has already been explored by many people and there exists many different approaches: importance splitting [JLS13; KH51; KM53; RR55], shields [Blo+20; BLX21; Bro+23; Bro+24], property-based synthesis [HR02; Pol+17], ...

None of those approaches is primarily aimed at solving the classic problem of local extrema, however. In particular, the Smart Sampling algorithm, like other SMC optimization algorithms, occasionally suffers a lot from this issue: the scheduler identified as optimal is sometimes a local extremum whose score is far from the score of the true global optimum. Guiding the generation of new schedulers can not solve this recurrent problem of the theory of optimization if done naively. In many cases, SMC optimization algorithms ultimately try to enhance the best schedulers they have found by altering them slightly to improve their result, for example with respect to a safety property. While this can potentially ensure that the final output corresponds to one of the best schedulers that exists *among those that are similar*, this can also prevent the exploration process from discovering very different schedulers that could perform even better.

We therefore decided to explore a strategy that is well-known to have the

potential to (partially) solve the local extremum problem: genetic algorithms.

Genetic algorithms balance their search of the exploration space by introducing diversity into their sample with *mutations*. This allows genetic algorithms to sometimes find new promising paths of exploration instead of always focusing on their best current path.

Nowadays, genetic algorithms have been exploited with great success in a variety of contexts. Although the core ideas of the strategy generally stay the same, the name has become an umbrella term which can be used to describe a large and heterogeneous class of algorithms. See [LGC19] for an extensive review of the literature on the topic.

In this section, our goal is to build the framework of a genetic algorithm for MDP scheduler optimization, as a way to provide a fundamentally better alternative to the Smart Sampling algorithm that is still as generic and broadly applicable as possible.

Unfortunately, to this day, our results are limited as the two attempts at implementing our genetic algorithm were inconclusive since the two corresponding collaborations (with other researchers) failed. The first results were promising however: even though the prototype version of the genetic algorithm was noticeably slow, it managed to consistently find truly optimal schedulers for custom-made MDPs whereas alternatives such as the Smart Sampling algorithms regularly missed them. A third attempt is planned in the near-future.

4.5.1 Scheduler Evaluation

The first feature of a genetic algorithm designed to solve Problem 4.1 we need to discuss is the fitness function. Of course, in this case, there is an obvious solution: the function that associates to each scheduler its probability to generate traces which satisfy the property ϕ.

Nevertheless, we have considered four different kinds of fitness functions in our experiments:

- The basic fitness function $f : \Pi \longrightarrow [0, 1]$ that returns the score of a scheduler:
$$f(\pi) = \int_{\Omega} \mathbb{1}_{T \models \phi} \, d\mu_{\pi}$$
- Fitness functions with a cutoff, for a threshold $K \in]0, 1[$:
$$f(\pi) = \left\{ \begin{array}{ll} \int_{\Omega} \mathbb{1}_{T \models \phi} \, d\mu_{\pi} & \text{if } \int_{\Omega} \mathbb{1}_{T \models \phi} \, d\mu_{\pi} \geq K \\ 0 & \text{if } \int_{\Omega} \mathbb{1}_{T \models \phi} \, d\mu_{\pi} < K \end{array} \right\}$$
- Skewed fitness functions, with an inspiration from entropy altering strategies such as the one exploited for the development of importance

sampling [Kah50; KM53]:

$$f(\pi) = \left(\int_{\Omega} \mathbb{1}_{T \models \phi} \, d\mu_{\pi} \right)^{n} \qquad \text{with } n \in \mathbb{N}_0$$

- Dynamical fitness functions, whose form vary over time, in a similar fashion to simulated annealing. In practice, what was experimented with were skewed fitness functions with a decreasing or cyclical parameter n.

At this point, in our experiments, there is not enough evidence to warrant the use of fitness functions more complex than the basic fitness function.

4.5.2 Scheduler Representation

How to represent the elements of the population of the genetic algorithm required a bit more innovation. Indeed, for such an algorithm, schedulers can not be represented with a single seed of a pseudo-random number generator similarly to what is done for the Smart Sampling algorithm, as we need to be able to link their representation with their behavior to make the inheritance of scheduler characteristics possible. Indeed, at each step of its execution, a genetic algorithm combines the (best) elements of its current population (the parents) to produce new elements (the children) in an informed fashion. The children must be able to (generally) inherit the (good) characteristics of their parents.

Four approaches were considered:

- The obvious solution is to represent schedulers explicitly, as maps from the set of states to the set of actions. However, this solution scales does not scale well (as it scales with the number of states of the model).
- Bit set (bit arrays) have the benefit that they can be modified and combined easily with each other. However, it was deemed too difficult to meaningfully link the behaviour of a scheduler to its bit set representation.
- Seed collections are the most promising solution by far. The idea is relatively straightforward: instead of representing each scheduler with a single seed (for a pseudo-random number generator), we represent each scheduler as a collection of c seeds. To determine the decision made by such a scheduler in a given state, we regard the scheduler as c distinct schedulers and we combine the decisions of those c schedulers with a predetermined function. In practice, we picked a majority vote strategy for that function, but other choices could be considered.

Note that this approach is compatible with both memoryless and history-dependent schedulers. Moreover, it not only scales well (the space complexity of that representation remains linear with respect to the number of schedulers and has no other dependency), but the additional cost can be scaled in function of the goals of the optimization process and the available resources by allowing a greater or smaller number of seeds for each scheduler. Experimentally, with a small-scale prototype, we noticed that $c = 32$ and $c = 64$ seemed to be acceptable trade-off values in terms of scalability and efficiency of the genetic algorithm.

- Probabilistic schedulers, or collections of probabilistic schedulers, form the last alternative for the applications for which their use is relevant. While they are an attractive solution from the theoretical perspective, with relatively natural definitions for the key features of the genetic algorithm, the difficulty of attributing a consistent score to those schedulers and their possible lack of interpretability make them a tricky choice. This approach has not been explored yet.

We mostly focused on the seed collection approach.

4.5.3 Scheduler Generation

At each step of its execution, the genetic algorithm must produce a new population. This process involves three paradigms, which are related to the principles of intensification and diversification for local search.

- *Elitism*: this consists in selecting (and keeping as it is) a proportion of the current population, the elements which have the highest fitness. The goal is to stabilize the optimization process and prevent negative progress.

- *Darwinism*: this consists in selecting pairs (or n-tuples) of elements with high fitness (the parents) to combine their representations (their "DNA") to create new elements (the children) whose behaviour must share characteristics with their parents' behaviours. The goal is to advance the optimization process by generating new elements that are similar yet different to the best known candidates.

  Additionally, random and spontaneous mutations, *i.e.* partial alteration of the representations after or before reproduction, can be performed. The goal is to occasionally expand the search of the genetic algorithm in the exploration space. This can potentially introduce unexpected breakthroughs and allow the algorithm to escape local extrema.

- *Adventurism*: this consists in introducing completely new elements in the population, without (directly) exploiting the representations of the current (best) elements. The goal is to provide another avenue to the algorithm to let it escape local extrema. As discussed in Section 4.4.2 and Section 4.5, purely random adventurism is likely to be unproductive when the exploration space is significantly larger than the examined population, such as what generally happens in the context of SMC. If the representation of the elements can be linked to their behavior however, one can analyze the current population at each step with statistics and guide the adventurous generation of new elements, either by forcing the new elements to share (some) of the characteristics of the (best) elements of the population, or on the contrary by forcing those new elements to differ as much as possible from what could be found in the population up to that point of the execution of the algorithm.

For the design of our genetic algorithm for the optimization of schedulers for MDPs, *elitism* is straightforward.

By representing the schedulers as collections of seeds, the core process of *darwinism* can be implemented by concatenating subsets of the representations of each parent of equal length to produce (the representation of) a new scheduler. If the collections of seeds of both parents overwhelmingly choose a specific action in a given state, the created child will make the same decision for that state: inheritance does happen. Mutations can be implemented by randomly replacing one or a few of the seeds in a scheduler representation. This shouldn't modify the choice of the scheduler for the states for which the votes of most seeds went to a single action, but could make the mutated scheduler modify its decision for the states for which there was no clear consensus. This is compatible with inheritance as well.

Lastly, advanced *adventurism* can be implemented in at least two diametrically opposed (but both potentially useful) directions. By identifying crucial decisions, *e.g.* selecting an action in a key state of the MDP which leads to a small loop, that are overwhelmingly present in the population of schedulers, we can either force the algorithm to focus exclusively on the schedulers that make that decision (if we believe this choice is a necessary condition for optimality) or force the algorithm to consider (at least some) schedulers that make a different decision for that specific state (if we believe that choice leads to a local extremum). This can be done automatically, in function of the goals and preferences of the user.

We have the goal of finding the right balance between elitism, darwinism and adventurism. In our first experiments, it turned out that elitism could be reduced to a minimum in most settings without impacting negatively the performance of our genetic algorithm. Darwnism requires a fine tuning of the

mutation rate: we are not sure yet if it will be possible to recommend a generic value. Last but not least, we think that statistical analysis has the potential to make adventurism an important tool to not only boost the performance but the speed of the algorithm as well, since it could significantly improve how quickly the algorithm samples different regions of the exploration space.

4.6 Conclusion and future work

We have barely touched on the difficult topic of SMC optimization algorithms. We mostly looked at a single algorithm, designed for a specific class of system models, and yet we have discovered numerous pitfalls along the way. Optimization is not easy to begin with, and optimization in stochastic settings is even worse.

Nevertheless, we made some progress. In this chapter, we have analyzed and corrected a lightweight algorithm for the optimization of MDP schedulers. In addition to preventing insufficient estimation of the schedulers' scores, which could lead to erroneous results, we have discovered that modifying the reduction factor of the algorithm could lead to a significant increase in speed without any sacrifice in performance. We have explored multiple strategies to improve the Smart Sampling algorithm further. Our two main conclusions were that (a) artificially inflating the simulation budget of the first step was sensible, (b) the reintroduction of random schedulers in the population as the worst schedulers were removed barely had any effect. Taking inspiration from this second conclusion, we ended with a discussion on smart sample generation and scheduler synthesis and proposed a framework for a genetic algorithm for SMC.

Future work On the short run, the goal is to implement a complete and efficient version of the genetic algorithm described in Section 4.5. We expect that a lot of testing will be required to fine-tune the algorithm. We also want to investigate whether classic local search techniques could be used to improve the genetic algorithm. Ultimately, we want to evaluate the resulting tool on a challenging use-case.

On the long run, if the genetic algorithm turns out to be successful, we could try to transpose this idea to allow the verification of models other than MDPs. Conversely, we could decide to focus on MDPs: it would be interesting to examine whether the Smart Sampling algorithm or our genetic algorithm could be adapted to optimize a MDP with respect to a reward function.

Implementation 5

This chapter focuses on the theory and implementation of parallelization as a strategy to improve the usability of SMC and to deal with the state-explosion problem. It presents a general framework to formalize the challenge of verifying properties of cyber-physical systems which depend on key performance indicators. We explain why adaptive stopping algorithms are perfectly suited for parallelization and we discuss how parallelization must be implemented to avoid the introduction of statistical bias. Then, we detail the structure of the architecture we deployed to perform our experiments. The results are analyzed to highlight the impact and efficiency of the parallelization of the verification process.

The content of this chapter is related to the journal publication titled "Scaling Up Statistical Model Checking of Cyber-Physical Systems via Algorithm Ensemble and Parallel Simulations over HPC Infrastructures" :
L. Picchiami, M. Parmentier, A. Legay, T. Mancini, and E. Tronci. "Scaling up Statistical Model Checking of Cyber-Physical Systems via algorithm ensemble and parallel simulations over HPC infrastructures". In: *Journal of Systems and Software* (2024), p. 112238.

5.1 Introduction

The growing need for tools designed for the formal verification of complex systems faces a fundamental problem of scalability. In particular, for real-world CPSs, exhaustive methods of model checking are basically always impossible to apply. For instance, even the simplest model of a prototype for a self-driving vehicle will have too many states and possible paths of executions for any comprehensive algorithm to be run within a realistic time frame. This is once again due to the state-explosion problem [Cla+12]: the size of a state-based model scales exponentially with the number of variables used to represent the system.

Therefore, stochastic algorithms must be used instead. SMC, sometimes in conjunction with machine learning [Lar+22], has for instance been successfully exploited for the validation and verification of large privacy- and security-sensitive structures, such as railway signalling systems or informa-

tion sharing platforms for healthcare [Bas+22b; Bar+21b].

Introducing parallelization to SMC algorithms can seem to be an obvious solution to enhance their performances and scales of use. While in some cases [AM11; PMT20a], this has been done with great results, the complexity of most of the more advanced techniques of SMC makes it often very difficult to implement at best, or fundamentally impossible at worst [Bul+11]. Nevertheless, examples of tools that implement a distributed version of some of their algorithms and exploit parallelization are UPPAAL-SMC [Ben+96; Dav+15], Modest [BHH12; TU; HH14] and Plasma Lab [JLS12a; Boy+13].

It turns out that stopping algorithms (SAs), as introduced in Chapter 3, are not only very efficient at minimizing the sample size of an estimation process but are also highly modular and well-suited for parallelization. Indeed, since those algorithms are purely statistical and agnostic with respect to how the samples are generated, they can not only be combined with a wide range of modelization and simulation strategies but also be run in parallel with each other.

This idea lead to the creation of EAA, a composite algorithm that concurrently runs a set (*ensemble*) of different SAs, by feeding them all with the same sequence of iid samples, and terminates as soon as the first SA of the ensemble converges with an approximation of the quantity of interest satisfying the requested statistical guarantees.

We investigated the actual benefits of parallelization for SMC by implementing the EAA algorithm with an efficient architecture exploiting HPC infrastructures to speed-up the generation of samples via the use of many identical simulators running in parallel.

This means that the parallelization of our approach is two-fold: both the SMC algorithms and the simulation process have been parallelized.

We tested and analyzed our new tool by conducting simulation-based SMC-driven verification of Simulink/Stateflow models of three CPSs of industrial size: AT, FCS and ALMA, which are three well-known case studies [MMT21; Man+21; Man+13; ZPC10; Man+14; Man+17; HAF14; Man+16b]. This allowed us to quantify the impact and efficiency of the parallelization, and examine the prospects of that strategy for SMC.

5.2 Formal Framework

CPSs are naturally modeled as dynamical or hybrid systems (see *e.g.* [Son13]). Model-based verification of a CPS amounts to checking whether some system requirements are verified. One of the simplest forms of system verification with SMC is the evaluation of a parameter of a given system in order to determine if its value is above or below a given threshold.

Such a task requires three components: a model of the system, the set of

possible operational scenarios, and a formalization of the specification to be verified.

Due to their complexity and diversity, many different formal frameworks have been proposed to define models for CPSs. State-based transition models, defined explicitly or through differential equations (with discrete or continuous time) are the most prevalent.

In this chapter however, for the sake of generality, we abstract away from any specific formal framework for the definition of the system model and simply assume (along the lines of, *e.g.* [Man+14; Man+16a; EP+22a; EP+22b]) that our *system under verification* (SUV) is modeled as a black-box input-output system, which takes as input an *operational scenario*, *i.e.* a time function of **both** the controllable inputs and the other uncontrollable events the system is subject to (*e.g.* faults in sensors and actuators or changes in system parameters), and produces a time function of system outputs (called *output function* or *trajectory*).

Definition 5.1 (System under verification model)**.** *A SUV model $\mathcal{S}$ is a tuple $(\mathcal{T}, U, O, S)$ where:*

- *$\mathcal{T}$ is the* time-set *($\mathbb{R}_0^+$ or $\mathbb{N}$ for continuous- and discrete-time systems, respectively, or an interval thereof);*
- *The sets U and O are the* input and output spaces*;*
- *$S : U^{\mathcal{T}} \to O^{\mathcal{T}}$ is the* I/O *(or* simulation*)* function*. For any* input function *$u \in U^{\mathcal{T}}$ (*operational scenario*), $S(u)$ denotes the* output function *$o \in O^{\mathcal{T}}$ of $\mathcal{S}$, defining the output $o(t)$ of $\mathcal{S}$ for each time point $t \in \mathcal{T}$, when the system is fed with u.*

Models for SUVs must therefore at the very least be executable (black box) models, with respect to discrete or continuous time, such that they output a unique output function (trajectory) for any given operational scenario.

The full set of operational scenarios of interest for a system (*i.e.* those considered possible or on which the verification activity needs to focus), together with a probability measure for each scenario to materialize is collectively called the *environment* of the SUV.

Definition 5.2 (Stochastic system environment)**.** *Let $\mathcal{S} = (\mathcal{T}, U, O, S)$ be a SUV. An environment $Env = (\mathbb{U}, P)$ for $\mathcal{S}$ is defined by a set $\mathbb{U} \subseteq U^{\mathcal{T}}$ of scenarios and an associated probability measure $P : \mathbb{U} \to [0, 1]$.*

For a SUV model $\mathcal{S}$, any element of $U^{\mathcal{T}}$ corresponds to one possible scenario of execution that the system can experience. In practice, since finite sampling is a necessity, only models whose input space can be represented with a finite number of variables, *i.e. finitely parametrisable* models, are used.

Definition 5.3 (Finitely parameterizable stochastic system environment)**.** *An environment Env is* finitely parameterizable *if there exist a finite-dimension parameter vector space* Λ *and a bijection between* Λ *and* $\mathbb{U}$*.*

The most straightforward class of system specifications are the specifications that can be defined as functions of the observable parameters: *key performance indicators* (KPIs). A KPI of a SUV is a function which associates a real value to each system trajectory. Any bounded KPI can be normalized such that it can be seen as a random variable in $[0, 1]$. Exclusively working with normalized KPIs ensures that even the most restrictive SMC algorithms can be exploited for their estimations.

Definition 5.4 (Normalized key performance indicator)**.** *A (normalized) key performance indicator (KPI) for a SUV* $\mathcal{S}$ *is a function* $\kappa : O^{\mathcal{T}} \to [0, 1]$*.*

The KPIs of a SUV can be used to define threshold-based properties. Despite their conceptual simplicity, those properties already cover many (if not most) of the verification requirements that need to be verified for real-world use cases.

Problem 5.1 (Threshold-based verification problem)**.** *A threshold-based verification problem for a SUV model* $\mathcal{S} = (\mathcal{T}, U, O, S)$*, a (finitely parameterizable) environment* $Env = (\mathbb{U}, P)$*, a threshold* $\theta \in [0, 1]$ *and a normalized KPI* $\kappa : O^{\mathcal{T}} \to [0, 1]$ *consists in deciding whether* $E_{u \in \mathbb{U}}(\kappa(S(u))) \leq \theta$*, where the expected value is computed with respect to the probability distribution* P *of scenarios as defined in Env.*

Any approximation algorithm which provides an unbiased (ϵ, δ)-estimator for the expected value of a $[0, 1]$-valued random variable can be applied to the binary random variable $X = (S(\mathbb{U}) \leq \theta)$ to offer a probabilistic answer to a threshold-based verification problem.

Indeed, if an (ϵ, δ)-approximation $\hat{\mu}$ of $\mu = E(\kappa(S(X))$ can be computed, then we know that, with probability at least $1 - \delta$:

$$\mu(1 - \epsilon) \leq \hat{\mu} \leq \mu(1 + \epsilon).$$

Therefore:

$$\frac{\hat{\mu}}{1 + \epsilon} \leq \mu \leq \frac{\hat{\mu}}{1 - \epsilon}.$$

In conclusion, since $\frac{\hat{\mu}}{1-\epsilon} \leq \theta$ implies that $\mu \leq \theta$, it suffices to check whether:

$$\frac{\hat{\mu}}{1 - \epsilon} - \theta \leq 0. \tag{5.1}$$

If that inequality holds, then the requirement of the system associated with the threshold θ and the KPI κ is successfully verified with a precision of ϵ and a $1 - \delta$ level of confidence. A threshold-based verification problem with a lower bound threshold can be defined and solved in a similar fashion.

5.3 Parallelization of Simulation-Based Estimation Algorithms

Any method of SMC which relies on the generation of large samples of independent simulations of a system can easily be parallelized (with itself) and one can expect a linear increase in speed if the following conditions are fulfilled: simulation time takes the most part of the process, samples are not modified at any point, and the logic of the algorithm is fully independent of the sample generation process. This was already known in 1949 [MU49]. Adaptive stopping algorithms not only satisfy all those conditions, but since they are purely sampling algorithms, they do not require specific model types and can be applied on top or in conjunction with other algorithms. Moreover, since they also only mostly care about the size of the generated samples and are thoroughly agnostic with respect to how those samples are generated, they can not only be parallelized with themselves but with each other as well.

Even with a straightforward master-slave architecture however, one needs to ensure that the parallelization does not break the independence of the generated samples and the unbiased quality of the estimator(s).

The problem of guaranteeing independence between the samples produced by each slave has a simple technical solution: the pseudo-randomness of each slave must be set and initialized separately, but independently as well. Simply choosing a different seed for the pseudo-random number generators of each slave may not even be sufficient, as the cyclical nature of pseudo-random number generators means that a different seed only lead to a different start location in their cycle of possible outputs. A Matsumoto-Nishimura scheme [MN00] is a possible solution: this essentially creates a new and independent variant of a pseudo-random number generator for each slave. However, any strategy involving completely different pseudo-random number generators for each slave is satisfactory as long as they have been proven to be independent of each other.

The problem of how to keep the unbiased quality of the estimator is a more subtle one. Perhaps surprisingly, a master-slave scheme in which the master exploits the samples produced by the slaves as they come introduces two bias to the estimation process. First, a theoretical bias in favor of samples which are easier and faster to generate. Second, a technical bias in favor of the subclass of samples which, because of specific technical choices (seeds of pseudo-random number generators), would happen to be only or mostly generated by workers that turn out to be the most efficient by a significant margin. Fortunately, those bias can be avoided by sticking to an ordered processing of the generated samples, as formalized by Theorem 5.1 (which is largely inspired by the work of Younes and Bulychev et al. [You04; Bul+11]). Two strategies are possible. The first one, *cyclical sampling*, involves process-

ing the samples in a cyclical order (with respect to the index of slaves). If the slaves are not homogeneous in speed however, this simple strategy can severily limit the benefits of parallelization. The other strategy, more complex but more robust, called *dynamic sampling*, consists in buffering the generated samples in a FIFO queue and processing them in the same order as they were commissioned.

Theorem 5.1. *Let $\mathcal{M} = (\mathcal{T}, U, O, S)$ be a SUV model and $f : O^{\mathcal{T}} \to [0,1]$ a KPI. Let $\mathcal{A}$ be an algorithm which can be applied to the random variable X obtained by composing f with the observable parameters of the system to compute its expected value μ. Let us assume that with standard (independent) sequential sampling, the corresponding estimator $\hat{\mu}$ is an unbiased (ϵ, δ)-estimator. Let $N_{(\epsilon,\delta)}$ be the size of the required sample for a fixed ϵ and a fixed δ. Then:*

1. *With cyclical sampling, the corresponding estimator $\mathring{\mu}$ is an unbiased (ϵ, δ)-estimator of μ.*
2. *With dynamic sampling, the corresponding estimator $\breve{\mu}$ is an unbiased (ϵ, δ)-estimator of μ.*

Proof. Remember that in this context, a sample $\{x_1, ..., x_k\}$ is an ordered collection of realizations of the random variable X, *i.e.* outputs of the KPI for simulations of the system.
The key observation of the proof is that, while there is no way to compare the probabilities of given samples being generated (for a given run of the algorithm $\mathcal{A}$, *i.e.* within a specific time frame) when they are of different sizes, the probability that a sample of fixed size is generated is always uniquely determined by the distribution of the random variable X. The goal is to show that samples produced by cyclical sampling and dynamical sampling have the same probability to be generated than those generated by standard sequential sampling. For any sample $M = \{x_1, ..., x_k\}$, let us denote $P_{\text{sequential}}(M)$ the probability that M would be produced with standard sequential sampling (when generating a sample of its size), $P_{\text{cyclical}}(M)$ the probability that M would be produced with cyclical sampling, and $P_{\text{dynamic}}(M)$ the probability that M would be produced with dynamic sampling. Note that, thanks to the hypothesis of independence for the generation of samples, for samples of size m, the probability distribution of $P_{\text{sequential}}(\,\cdot\,)$ is the joint probability distribution of X^m (the Cartesian product of X with itself m times).

1. With cyclical sampling, we have access to m ordered samples $M_1, ..., M_m$ with $|M_i| = {N_{\epsilon,\delta}}^*/m$ for all $i \in \{1, ..., m\}$, with ${N_{\epsilon,\delta}}^*$ being the smallest integer greater or equal to $N_{\epsilon,\delta}$ that is a multiple of m.

$$P_{\text{cyclical}}\left(\bigsqcup_i M_i\right) = \prod_{i=1}^{m} P_{\text{sequential}}\,(M_i) = P_{\text{sequential}}\left(\bigsqcup_i M_i\right)$$

The first equality follows from the definition of cyclical sampling and from the independence between the sample generating process of each slave. The second equality follows from the fact that the probability distribution of $X^{N_{\epsilon,\delta}{}^*}$ is equal to the joint probability distribution of m copies of $X^{N_{\epsilon,\delta}{}^*/m}$. Therefore, as $\hat{\mu}$ is unbiased, $\mathring{\mu}$ is unbiased as well. Moreover, since samples of size $N_{\epsilon,\delta}$ guarantee ϵ, δ-estimation for $\hat{\mu}$, a sample size of $N_{\epsilon,\delta}{}^*$ does as well, and thus $\mathring{\mu}$ is also a ϵ, δ-estimator.

2. With dynamic sampling, since samples are processed fully independently with respect to the order in which they are received, we formally end up with $N_{\epsilon,\delta}$ ordered samples of size 1: M_1, ..., $M_{N_{\epsilon,\delta}}$.

$$P_{\text{dynamic}}\left(\bigsqcup_i M_i\right) = \prod_{i=1}^{N_{\epsilon,\delta}} P_{\text{sequential}}\left(M_i\right) = P_{\text{sequential}}\left(\bigsqcup_i M_i\right)$$

The first equality follows from the definition of dynamic sampling, from the independence between the sample generating process of each slave, and from the independence for the sampling process of each slave itself. The second equality follows from the fact that the probability distribution of $X^{N_{\epsilon,\delta}}$ is equal to the joint probability distribution of $N_{\epsilon,\delta}$ copies of X. Therefore, as $\hat{\mu}$ is unbiased ϵ, δ-estimator, $\breve{\mu}$ an unbiased ϵ, δ-estimator as well.

□

Committing to processing the elements of a combined sample in the order they were requested allows theoretically for a linear speed up, at least in the limit [You04]. A complete breakdown of the linearization and a loss of its benefits is theoretically possible (if one slave happens to be consistently and definitively much slower than the others). However, at least for relatively homogeneous clusters, it is in practice reasonable to expect the average simulation speed of each slave to be relatively close to each other, especially in the long run. Interestingly, it is possible to even avoid this potential problem when exclusively working with probabilistic quantities, within a context of hypothesis testing for instance, by (temporarily) replacing samples which have not been received yet by either their worst or best possible value, which can even be 0 and 1 when no *a priori* knowledge is available. See [You04] for more details.

5.4 Ensemble of Approximation Algorithms

As explained in Section 5.2, any adaptive SA which computes an (ϵ, δ)-approximation of the expected value of a $[0, 1]$-valued random variable provides a SMC method to solve threshold-based verification problems.

In a setting such as the model-based verification of a CPS, where generating samples of the system KPI(s) is particularly expensive because it requires numerical simulation of the system model, using a single SA to perform SMC might result in an unnecessary large number of samples to be generated. This is even true for the SAs proved to be optimal (e.g. AA or EBStop), since the optimality for the sample sizes is guaranteed only probabilistically and up to a multiplicative factor. Unfortunately, there is no way to know *a priori* which SA will be the fastest, as the number of samples *actually* needed by any given SA to converge on a given verification task depends not only on the requested properties of the approximation, but also on statistical properties of the KPI to be approximated, which are typically unknown in advance.

In a similar fashion to AI-based techniques such as *ensemble learning* methods [SR18; Don+20], we propose a general procedure, named EAA, that exploits a finite set $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of different (ϵ, δ)-approximation algorithms.

The EAA algorithm (see Figure 5.1 for a conceptual high-level view and pseudocode) first initializes the states of all the algorithms in the ensemble (line 9). Then, it repeatedly generates samples of the random variable X. Here, this means generating a random scenario u for the SUV (line 10), simulating the SUV model on it, and computing the associated value (x) of the KPI of interest. Each sample x is then used to feed and advance *all* algorithms of the ensemble (line 12) *by one step only* (*i.e.* processing only x). This is done by the functions $\text{advance}_{\mathcal{A}_l}()$ ($l \in [1, k]$).

As soon as one of the algorithms (say $\mathcal{A}_l$) reaches its termination condition, EAA stops and returns the approximation $\hat{\mu}_l$ provided by $\mathcal{A}_l$ (line 15), which is a valid (ϵ, δ)-approximation of the true mean μ of the value of the KPI of interest. The value $\hat{\mu}_l$ is then used to answer the threshold-based verification problem, by checking whether Equation (5.1) holds.

Since the procedure terminates as soon as the first algorithm in the ensemble, say $\mathcal{A}_l$, reaches *its own* stopping criterion, the number of samples required by EAA is always the *minimum* number of samples that would have been required by any of the algorithms in the ensemble.

Moreover, since we know for every specific run of EAA which of its underlying algorithms stopped first, and since we are only interested in knowing whether the output is a valid, EAA can indeed be used to produce (ϵ, δ)-approximations. If one is afraid of noticeable early termination bias, it always remains possible to select a slightly smaller δ to compensate. The exact formula depends on the number of algorithms integrated to EAA.

The design of EAA revolves around proper subroutines that advance each algorithm in the ensemble by one step (*i.e.* processing just one sample) at the time and manipulate the algorithm persistent state appropriately.

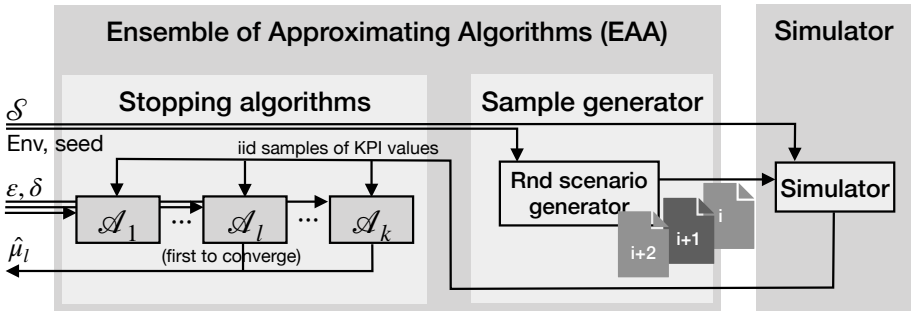


Figure 5.1: Conceptual view of EAA (sequential reference architecture).

Algorithm 7 EAA

Input:
$\mathcal{S}$, SUV model
Env, stochastic environment
seed, random seed
$\epsilon, \delta \in (0, 1)$
$\mathcal{A}_1, \ldots, \mathcal{A}_k$, set of (ϵ, δ)-appr. algorithms
Output:
(ϵ, δ)-approximation of the KPI of interest

```
1:  initialize rnd scen. generator with Env and seed
2:  initialize simulator with S
3:  initialize A_1, ..., A_k with ε and δ
4:  s_1, ..., s_k ← initial states of A_1, ..., A_k
5:  while True do
6:      x ← next_sample
7:      for l ∈ {1, ..., k} do
8:          s_l ← advance_{A_l}(s_l, x)
9:          if A_l has reached its stopping condition then
10:             return μ̂_l

11: function NEXT_SAMPLE
12:     u ← generate next rnd scenario
13:     x ← simulate u
14:     return x
```

Namely, each advance$_{\mathcal{A}_l}$() subroutine: (a) takes as input the current state s_i of algorithm $\mathcal{A}_l$ and a new sample x; (b) processes x by advancing the algorithm state into s'_l; (c) returns the resulting state s'_l. Such subroutines can be easily written by refactoring any approximation algorithm.

For our implementation of EAA, we selected the two SAs known to be optimal or quasi-optimal (probabilistically and up to a multiplicative factor): EBGStop and AA.

Advancing each algorithm in the ensemble only requires a few algebraic operations. Thus, the computational overhead of advancing multiple algorithms in the ensemble is expected to always be negligible with respect to the (much longer) time needed to perform the simulations of the SUV model. This expectation will be shown correct in Section 5.6.4.2.

It should be noted that EAA "parallelizes" the verification in two senses: it distributes the sample generation process and runs the integrated estimation algorithms as concurrent programs. The most fundamental aspect of the parallelization corresponds to the former. In our case, the latter has little to no impact on performance, as their structure of EBGStop and AA as stopping algorithms allows us to deconstruct them into one-step versions and advancing those one-step versions has minimal computational cost (with respect to the simulations). If more estimation algorithms were integrated into EAA, algorithms that would not necessarily all be stopping algorithms, then this aspect of the parallelization may play a bigger role.

5.5 Implementation of EAA Over a High-Performance Computing Infrastructure

While EAA allows us to stop the verification process as soon as the first algorithm of the ensemble reaches its termination condition, it does not address the second major bottleneck of model-based CPS verification via SMC: the rate at which samples are provided to the approximation algorithm.

As discussed in Section 5.3, one must be careful when distributing the simulation tasks of a verification task based on the estimation of a system parameter. We therefore implemented the strategy of dynamical sampling with a ticket-based system.

Our overall architecture is shown in Figure 5.2. It is designed to run on a possibly very large HPC infrastructure, and includes: a multi-process or multi-threaded module devoted to advancing the SAs of the ensemble one sample at a time (lines 3–10 of Figure 5.1); n instances (with n which can be very large) of a simulation module, each one running an identical copy of a simulator of the SUV model; and lastly one multi-process or multi-threaded

module hosting a service, named APSG, devoted to the generation of random scenarios and the delegation of the corresponding simulations to the n available parallel simulators.

Since the latter service runs asynchronously with respect to the SAs, the overall architecture implements an instance of the *producer-consumer paradigm*.

The exact operation of APSG is described in details in Figure 5.2.

First, the function $initialize()$ (line 6) initializes APSG with the given SUV model $\mathcal{S}$, its environment Env, and a random seed to generate scenarios from Env. This function instantiates the n simulators with identical copies of $\mathcal{S}$ and initializes the random scenario generator. Note that, since the environment is finitely parametrizable, scenario generation reduces to generating pseudorandom numerical vectors within some bounded finite-dimensional domain (see Definition 5.2). Then, function $initialize()$ starts the process(es) (or thread(s)) $generate\_samples()$ and returns immediately.

Process $generate\_samples()$ (line 10) continuously generates (see bullet point 1 in Figure 5.2) a random scenario u from the system stochastic environment, retrieves (*dequeues*) a simulator sim from the queue of idle simulators (bullet point 2), and delegates to sim the two tasks of simulating $\mathcal{S}$ under the scenario u and of computing the value of the KPI of interest.

The queue of idle simulators is initially filled with identifiers of all the available simulators (line 8). If found empty, the process waits (*blocking dequeue* operation) for a simulator to become idle (line 13).

Each scenario is identified by a progressive number (integer i in the pseudocode).

Whenever a simulator terminates its task on scenario i, having computed value x for the KPI of interest, it calls function $sample\_produced()$ of APSG. This function (line 16) and bullet point 3 in the figure) inserts entry $i \mapsto x$ in the map *samples*, thus storing the value of the system KPI on scenario i (this corresponds to the *produce* step in the producer-consumer paradigm), and the simulator is added to the queue of idle simulators, signalling that it is ready to receive new simulation requests. To prevent memory explosion, map *samples* has a maximum capacity. If the map is full, the insertion of the new sample is blocked until an entry is removed.

The module running the SAs continuously (and asynchronously) requests samples from APSG by calling function $next\_sample()$ (this corresponds to the *consume* step in the producer-consumer paradigm). This function (line 19) and bullet point 4 in the figure) returns the KPI value associated to scenario with id j in the map *samples* and waits if such an entry does not (yet) exist (because not yet computed). Before returning, this function frees-up memory by removing from map *samples* the entry associated to scenario j and increments j, which becomes the id of the next sample to be provided to the SAs.

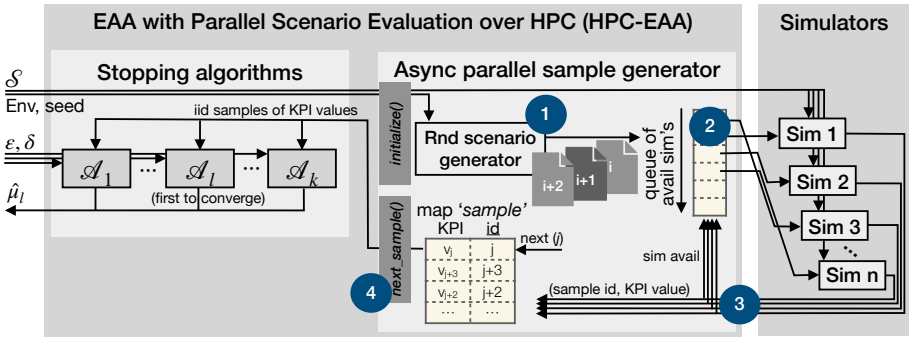


Figure 5.2: HPC-EAA: high-level parallel architecture.

Algorithm 8 Asynchronous Parallel Sample Generator (APSG)

global $i \leftarrow 0$
global $j \leftarrow 0$
global avail_sims $\leftarrow \{\}$
global samples $\leftarrow \{\}$
param n
function INITIALIZE($\mathcal{S}$, Env, *seed*)
    initialize random generator with Env and *seed*
    initialize the n available simulators with $\mathcal{S}$ and add them to avail_sims
    start process generate_sample()
process next_sample()
while ¬ halt requested **do**
    $u \leftarrow$ generate rnd scenario
    $sim \leftarrow$ dequeue from avail_sims
    send (i, u) to sim for simulation and KPI evaluation
    $i++$
function SAMPLE_PRODUCED(sim, i, x)
    add $(i \mapsto x)$ to samples
    enqueue sim to avail_sims
function NEXT_SAMPLE
    $x \leftarrow$ samples$[j]$
    free-up samples$[j]$
    $j++$
    return x

It is worth noting that, as intended, APSG provides the SAs with the very same sequence of samples that would be produced with the basic (purely sequential) architecture of Figure 5.1, if run with the same random seed. In other words, the order of the scenarios generated is independent of the number n of simulators. Thus, increasing n will have no other effect than increasing the rate at which the samples are made available to the SAs, hence reducing the total completion time. In this way, we avoid any bias in the (pseudo-)randomness of the samples and therefore in the final result of the approximation.

As mentioned before, with the version of EAA (with EBGStop and AA only) we used for our experiments, the key aspect of the parallelization is the proper distribution of the sampling. In our case, running the subroutines of EAA as concurrent processes was not strictly necessary.

5.6 Experiments

5.6.1 Case Studies

Our case studies were realized with three Simulink/Stateflow models of industry-scale systems: Automatic Transmission (AT) [Mat23b], Fuel Control System (FCS) [Mat23c] and Apollo Lunar Module Autopilot ALMA [Mat23a]. These models have already been profusely studied in the literature, see, *e.g.* [MMT21; Man+21; Man+13; ZPC10; Man+14; Man+17; Man+16b; HAF14; Abb+13; Bar+20].

5.6.1.1 Automatic Transmission (AT)

System model. The AT model defines a typical automotive drivetrain. Nonlinear ordinary differential equations are used to specify the engine dynamics, the four-speed automatic transmission and the vehicle dynamics. The model has two inputs, the throttle opening (expressed in percentage) and the brake torque (expressed in ft-lb), and two outputs, the engine speed (in RPM) and the vehicle speed (in mph).

The throttle opening delineates the accelerating dynamics, whereas the brake torque represents the braking dynamics of the vehicle.

In the literature, the AT system has been extensively used for search-based falsification purposes (see, *e.g.* [Abb+13; HDF18; Dok+15]) and benchmarks (*e.g.* [HAF14; Ern+22]), especially when the design of input trajectories has a strong impact on the verification (see, *e.g.* [Bar+20]).

Environment. We modeled a stochastic environment for the system that generates disturbances as input time functions for the SUV model. In our setting, the brake is always set to 0, whereas the throttle has a fixed average value μ_d that is perturbed with a given variance σ_d^2, *i.e.* the values are sampled

from a normal distribution $N(\mu_d, \sigma_d^2)$ truncated within $[0, 100]$. We set $\mu_d = 40$ and $\sigma_d^2 = 20$ for the truncated normal distribution, thus modeling a varying accelerating dynamics.

Property to be verified. In the context of search-based falsification, a typical system-level specification requires that the engine and vehicle speeds do not exceed given safety thresholds for the whole simulation duration. Therefore, we designed a (metric) safety property which quantifies how much time the engine and the vehicle speeds spend above such thresholds.

To define our system-level requirement, we first have to define the maximum time window during which any of the two speed is above its thresholds. We define the *Max Time Speed* (MTS) and the *Max Time Engine* (MTE) at time t as:

$$\mathrm{MTS}(t) = \max\{t_2 - t_1 \mid 0 \le t_1 \le t_2 \le t \wedge (\forall t' \in [t_1, t_2] : \mathrm{speed}(t') > M_s)\}$$
$$\mathrm{MTE}(t) = \max\{t_2 - t_1 \mid 0 \le t_1 \le t_2 \le t \wedge (\forall t' \in [t_1, t_2] : \mathrm{engineRPM}(t') > M_e)\}$$

where $\mathrm{speed}(t)$ and $\mathrm{engineRPM}(t)$ are the values of the vehicle and engine speeds at time t, whilst M_s and M_e are the respective maximum safe values. The *Max Time Violation* at time t is then defined as:

$$\mathrm{MTV}(t) = \frac{\max\{\mathrm{MTS}(t), \mathrm{MTE}(t)\}}{h}$$

where h is the simulation time horizon.

The verification activity aims to establish whether $E(\mathrm{MTV}(h)) \le \theta$ for some threshold. We set $M_s = 4000$ RPM and $M_e = 120$ RPM (according to previous work, see *e.g.* [Fan+17; Cla+18a]), $h = 1800$ s (*i.e.* 30 minutes of system simulation time), and $\theta = 0.05$, so as to assess whether the system violates the desired specification for any time window longer than 5% of the total simulation time.

5.6.1.2 Fuel Control System (FCS)

System model. The FCS model represents a control system for a fault-tolerant gasoline engine that is expected to accept up to one sensor fault, and has been widely used as a benchmark of various simulation-based verification approaches (see, *e.g.* [ZPC10; Man+13; HAF14; MMT21; Man+21] and citations therein).

The system has four sensors: the *Throttle Position* (TP), *Engine Speed* (ES), *Exhaust Gas Oxygen* (EGO) and *Manifold Air Pressure* (MAP). The TP and ES sensors provide the current air throttle position and the engine speed to the engine controller. The EGO sensor reads the current oxygen level in the engine's exhaust gas to adapt the fuel supply, whereas the MAP sensor reads the suction pressure of the airflow at the engine's intake manifold.

The model outputs the following two quantities: the *air-fuel ratio*, that is, the ratio between the air mass flow rate computed by the intake manifold and the fuel mass flow rate pumped by the injectors, and the *fuel flow rate* itself.

The system aims at keeping the air-fuel ratio close to the stoichiometric (ideal) value of 14.6 since it is a good compromise between power, fuel economy and emissions. However, this requirement becomes more challenging to meet in the presence of sensor faults. In particular, when a fault on a single sensor is detected, the system modifies its behavior by operating the engine with a higher fuel rate. In case of two or more sensor faults, the system shuts down the engine.

Environment. We modeled a stochastic environment for the system where temporary faults may occur on the ES, EGO and MAP sensors. Finite state machines (with Stateflow charts) were used to simulate fault injections and the subsequent recovery operations, similarly to [ZPC10; HAF14]. Each finite state machine injects faults on a given sensor according to an exponential distribution whose mean corresponds to the MTBF. Similarly, fault recovery operations occur after a time window whose length is randomly chosen according to an exponential distribution having a mean of 1 s. We set the MTBF values for the EGO, ES, and MAP to 3 s, 7 s and 8 s respectively, which is the configuration used in [ZPC10] for verification purposes.

Property to be verified. We experimented with one of the key system-level specifications for this model (see, *e.g.* [Man+13; MMT21; Man+21]), that is, establishing whether the fuel flow rate is identically zero for too long time. Therefore, we designed a KPI which outputs the (ratio of the) length of the longest time window during which the fuel flow rate is identically equal to zero. We define *Max Time Fuel* (MTF) at time $t \geq 5$ s as follows (the first 5 s of each trajectory have been ignored to allow for the initial setup of the system):

$$\mathrm{MTF}(t) = \frac{\max\{t_2 - t_1 \mid 5 \leq t_1 \leq t_2 \leq t \wedge (\forall t' \in [t_1, t_2] : \mathrm{fuel}(t') = 0)\}}{h - 5}$$

The goal of the verification is to establish whether $E(\mathrm{MTF}(h)) \leq \theta$ for some threshold. We set $h = 105$ s and $\theta = 0.01$, so as to assess whether the fuel flow rate is identically zero for more than 1% of the time on average.

5.6.1.3 Apollo Lunar Module Autopilot (ALMA)

System model. The ALMA model defines the logic of the phase-plane control algorithm for the autopilot program of the lunar module used in the Apollo 11 mission. This model has been extensively used in the literature for simulation-based verification (see, *e.g.* [MMT21; Man+21] and citations therein).

The system is equipped with 3 sensors (yaw, roll, pitch) and 16 reaction jets that actuate a rotation over one or more axes. In every space mission, *i.e.* in every system simulation, the controller takes as input a request to change the module's attitude and computes which reaction jets need to be activated to achieve the desired rotation and stabilize the system over the new current attitude.

Environment. We modeled a stochastic environment where temporary faults may occur on yaw, pitch and roll sensors, similarly to what we did for the FCS model. For this model, a fault corresponds to the injection of white noise to one of the sensor signals. Analogously to the FCS model, the occurrences of the faults are chosen according to an exponential distribution whose mean corresponds to a MTBF of 4 s. Similarly, fault recovery happens after a duration once again determined by an exponential distribution with a mean of 1 s.

Property to be verified. We designed a system-level requirement that quantifies the outcome of a given space mission in terms of the average module's attitude error over the entire simulation. To define such an error, we first have to define the attitude error indicators for each axis. For each axis s, if $a_s(t)$ and R_s denote the current and target attitudes at time t along s, respectively, we define the attitude error e_s at time t as follows:

$$e_s(t) = |a_s(t) - R_s|$$

that is, the absolute difference between the target and the current attitudes of the lunar module along axis s. Since the rotation request is over one or more axes, we define the *Attitude Module Error* (AME) at time t as follows:

$$\mathrm{AME}(t) = \max\left\{e_y(t), e_p(t), e_r(t)\right\}$$

where the $e_y(t)$, $e_p(t)$, and $e_r(t)$ are, respectively, the attitude errors on the yaw, pitch and roll axes at time t. Our final KPI is thus the *Normalised Mean Attitude Module Error* at time t, defined as:

$$\mathrm{NMAME}(t) = \frac{1}{2\pi t}\int_0^t \mathrm{AME}(\tau)\mathrm{d}(\tau)$$

where 2π is a normalization factor, since a rotation along a given axis ranges in $[0, 2\pi]$.

Similarly to the AT and the FCS models, the goal of our experiment is to verify whether $E(\mathrm{NMAME}(h)) \leq \theta$ for some threshold. We set $h = 60$ s and $\theta = 0.005$. Note that greater values for θ systematically lead to system mission failures.

5.6.2 Implementation

We implemented our production software using Python and Cython, and exploited the MPI standard [GLS99] to allow communication between processes.

The overall architecture has been explicitly designed to be run on an a HPC infrastructure of networked computational nodes, and reflects that of Figure 5.2 with some adjustments to boost performance. The SAs and APSG modules of Figure 5.2 have been implemented as concurrent threads of a process named HPC-EAA, while the n simulators have been deployed as independent processes on n additional computational nodes. HPC-EAA runs various parallel threads: one devoted to the advancement of each SA, one devoted to the generation of random scenarios and their delegation to an available simulator taken from the queue, and one devoted to processing the receiving queue of samples and populating the map *samples*. The use of such a thread-based implementation for the SAs and APSG modules limits the use of the network to the communication with the simulators.

5.6.3 Experimental Setting

For each case study we computed (ϵ, δ)-approximations of the KPIs of interest for 9 different values for ϵ (ranging from 10^{-1} to 10^{-3}), 3 values of δ (10^{-1}, 5×10^{-3}, 10^{-2}), and 7 values for the number of workers n (1, 64, 128, 256, 512, 1024, 2048). Each experiment has been repeated 10 times, using 10 sequences of samples, each one produced by a different sequence of iid scenarios.

Conducting such an analysis using our production tool as described in Section 5.6.2 would be a practically unviable option, as it would require dozens of years (see Figure 5.7, Figure 5.8 and Figure 5.9) and incur huge costs for the full allocation of a large-enough private HPC infrastructure. Indeed, for a proper analysis to be carried out, the production tool would need to be run 10 times for each case study and for each value of ϵ, δ and n (including $n = 1$). Moreover, each execution of the tool would require the full availability of $n+4$ computational nodes (CPU cores), including $n + 4 = 2048 + 4 = 2052$.

This obstruction is becoming common when analyzing performance and scalability of modern software explicitly designed to perform intensive computations on large HPC infrastructures, and is being handled in the literature from the methodological point of view. Along these lines, and by following recent approaches such as, *e.g.* [EMT23], we operated in an indirect way as follows.

First, we avoided the use of an expensive private HPC infrastructure, by exploiting a shared HPC cluster at Sapienza University, which is managed by a centralized scheduler which orchestrates the launch of the computational jobs submitted concurrently by the various research teams. Although this was a mandatory choice, it created some issues, as jobs requesting too many

computational nodes to be allocated would receive a very low priority and could remain in the waiting queue forever.

Hence, as in [EMT23], we disassembled our tool into its single components, instrumented each component with proper timers within the code, and ran each of them asynchronously. More precisely:

1. We generated 10 random seeds for each case study and computed 10 sequences of scenarios long enough to allow all SAs to terminate for all values of ϵ and δ. We saved these sequences of scenarios into a centralized database, together with their ids and generation times (which are all on the order of a few microseconds).

2. We submitted to the HPC scheduler the simulation jobs for all scenarios independently, and measured the time taken by each of them to simulate the selected scenario and to compute the corresponding value of the system KPI (we ignored the simulator set-up time, which would not have had to be paid for each sample if we had instead used the production tool). For each scenario, the simulation time and the KPI value were also saved in the centralized database. Note that, as expected, simulation is by far the slowest step in the whole workload, with average times being orders of magnitude higher than those of the others, namely: 0.1377 s, 0.2279 s, 7.1654 s for AT, FCS and ALMA, respectively.

3. For each case study, the average half turnaround network time for each message type from HPC-EAA to a simulator and vice versa was measured independently, by running the real (MPI-based) application over 1000000 scenarios. Thanks to a low-latency high-throughput local network, these times are all on the order of a hundred of microseconds (*i.e.* around 10^{-4} s), hence several orders of magnitude lower than simulation times.

We asked the HPC scheduler to run all the computations on nodes of identical machines, each one equipped with 2 AMD EPYC 7301 CPUs with 32 cores and 256 GB RAM, and hyper-threading disabled. This ensured that the measured time durations can be seamlessly combined together.

With the above data, we were able to virtually exercise our tool on 10 random experiments for each case study, assessing convergence of the SAs and overall performance for each setting (ϵ, δ, and n). Indeed, the availability of the generation and simulation times of each scenario as well as of the average half turnaround time of network messages of each type allowed us to perform a reliable estimation of the execution time that our production tool would have shown, under each setting, if run over fully reserved portions of the HPC infrastructure at hand. For each experiment, the SAs and

APSG modules were run normally (as a multi-threaded process on a single node), while the set of simulators were replaced by the database containing the pre-computed samples. A discrete event-based emulator module running on the same node (see [EMT23]) orchestrated the execution of the whole experiment in a faster-than-real-time way, by mimicking the advancement of time via a global counter accumulating the time durations of all steps of the distributed EAA algorithm (*i.e.* scenario generation, blocking dequeue operations from the queue of available simulators, network communication with simulators, blocking look-up operations and insertion/deletion of entries to/from the *samples* map and advancement of all SAs). The time durations of the steps performed normally were measured via the timers instrumenting the code, while those of the emulated steps were taken from the database.

We note that, although we did our best to also take into account, in our estimations, the steps which are several orders of magnitude faster than the dominating tasks (system model simulations), it is known that disassembling and instrumenting the production code can introduce unnoticed small divergences between the estimated time and the time that our tool would have shown if run normally. However, in this case, such potential small divergences would actually be unmeasurable and not repeatable in practice. This is due to how modern large HPC infrastructures (even those fully reserved for a single job) are managed by complex orchestrating and probing services, and by software virtualization and containerization layers, which may introduce a significant amount of noise.

In other words, the estimations below, as all estimations of the completion time of complex massively parallel and computationally intensive software designed for large HPC infrastructures, reasonably estimate what would happen if the software were to be run on bare metal.

5.6.4 Results

5.6.4.1 Reduction of the Number of Required Samples due to EAA

To evaluate the benefit of using an ensemble of SAs, we repeated our experiments 10 times for each case study. For each of them, we fed the SAs module with one of the 10 random sequences of iid samples computed in advance for that model, as described in Section 5.6.3, and computed the number of samples required by each of the two SAs of the ensemble to converge. The number of samples required by EAA is then the minimum of the two.

Figure 5.3, Figure 5.4 and Figure 5.5 (top) show, for each case study and for each of the 10 random sequences of samples, the number of samples required by EAA to converge, for each value of ϵ and δ.

Similarly, Figure 5.3, Figure 5.4 and Figure 5.5 (bottom) show the difference in the number of samples between the two SAs, which corresponds to

the saving achieved by using EAA when compared to the algorithm that happened to be the slowest for each (ϵ, δ)-approximation. We can see how using EAA is a particularly effective way to keep the number of required samples to a minimum. Indeed, our experiments show that EAA may require up to 31300400 fewer samples when compared to using a single algorithm, with relative reductions as large as 78.73%.

Predicting *a priori* which algorithm of the ensemble will terminate first is a difficult task, since it depends not only on ϵ, δ and on the statistical properties of the KPI of interest (with the latter being typically unknown in advance), but may also be influenced by the actual random sequence of iid samples (*i.e.* on the seed of the pseudorandom number generator). The results for the ALMA case study (see Figure 5.5, bottom) are a good illustration of that phenomenon. EAA also provides a stabilizing effect in this regard.

Indeed, in contrast with what emerges in Figure 5.5 (bottom), the curves in Figure 5.5 (top) for the same case studies and the same values for ϵ and δ (but generated using different random sequences of samples) overlap with each other almost perfectly.

5.6.4.2 Increase in the Sample Production Rate due to the Asynchronous Parallel Sample Generator

We now analyze the impact of deploying APSG on a parallel HPC infrastructure. In Figure 5.6 (a) we show an estimation (with the method outlined in Section 5.6.3) of the maximum rate $\rho_{\text{APSG},n}$ of sample production when varying the number n of available simulators. For each value of n, $\rho_{\text{APSG},n}$ has been obtained by replacing the SAs module with a simple loop requesting 1 million samples from APSG when governing n parallel simulators. Hence, these values are the highest sample production rates actually achievable by APSG on our infrastructure when using n simulators of the given model. Figure 5.6 (a) shows that all $\rho_{\text{APSG},n}$ are practically linear with respect to n, with a maximum deviation between the actual and the ideal (linear) rates $< 0.1\%$.

In Figure 5.6 (b) we show again the three values for $\rho_{\text{APSG},n}$, this time using a logarithmic Y axis and along with the maximum rate ρ_{EAA} of consumption of samples by the SAs module. Rate ρ_{EAA} has been obtained by replacing APSG and the controlled simulators by a simple pseudorandom generator feeding the SAs module with numerical values in $[0, 1]$, and measuring how long it took to consume 1 million of such samples. This value is thus the lowest rate of samples that APSG would need to provide to the SAs module (running on our infrastructure) to allow the latter to advance at full speed.

From Figure 5.6 (b) it clearly emerges that ρ_{EAA} is always several orders of magnitude higher than all $\rho_{\text{APSG},n}$ (even when the largest number

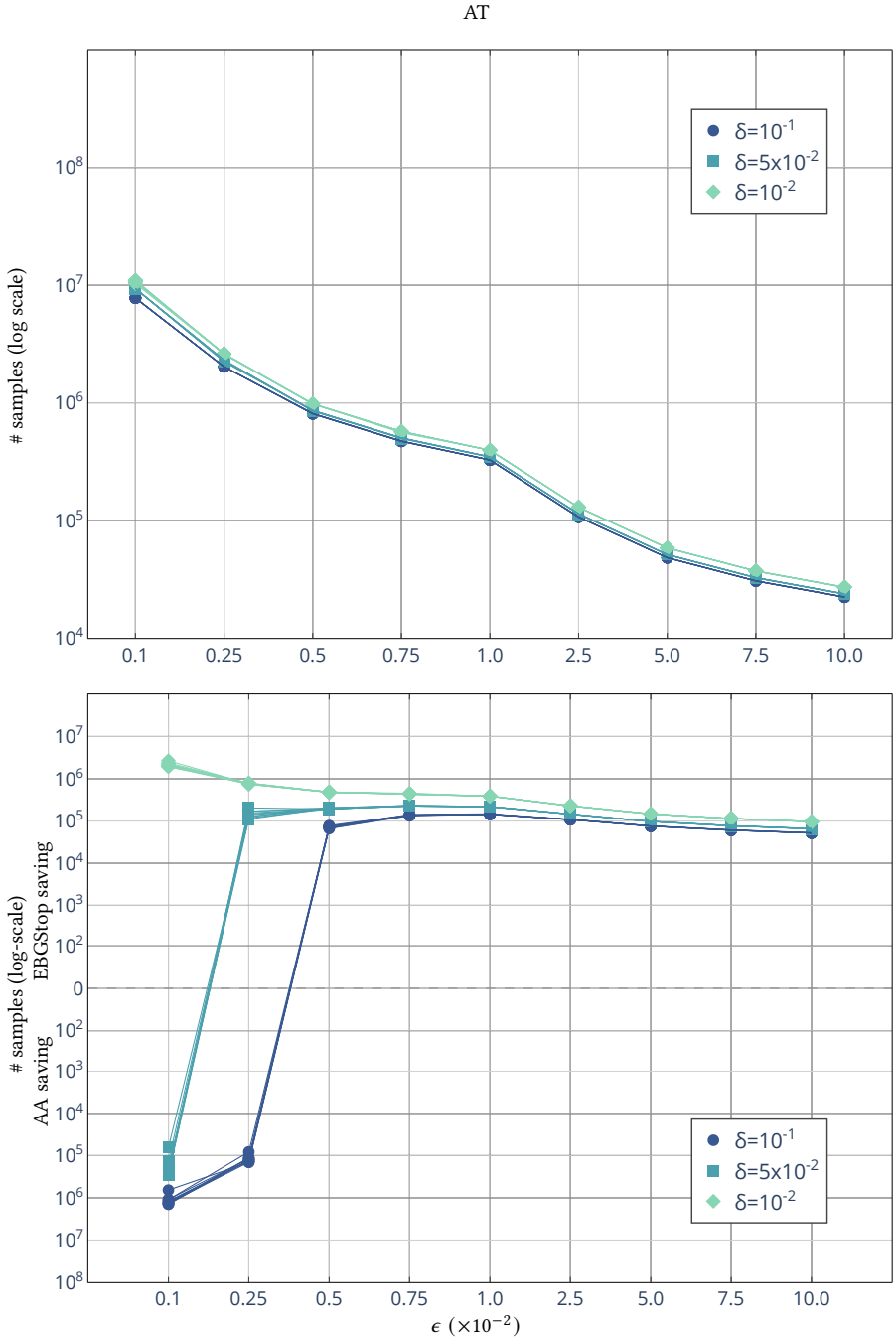


Figure 5.3: Top: average number of samples (vertical log-scale axis) required by EAA to converge, for various values of ϵ and δ. Bottom: average savings in the number of samples achieved by EAA in each setting wrt. the algorithm requiring the most samples.

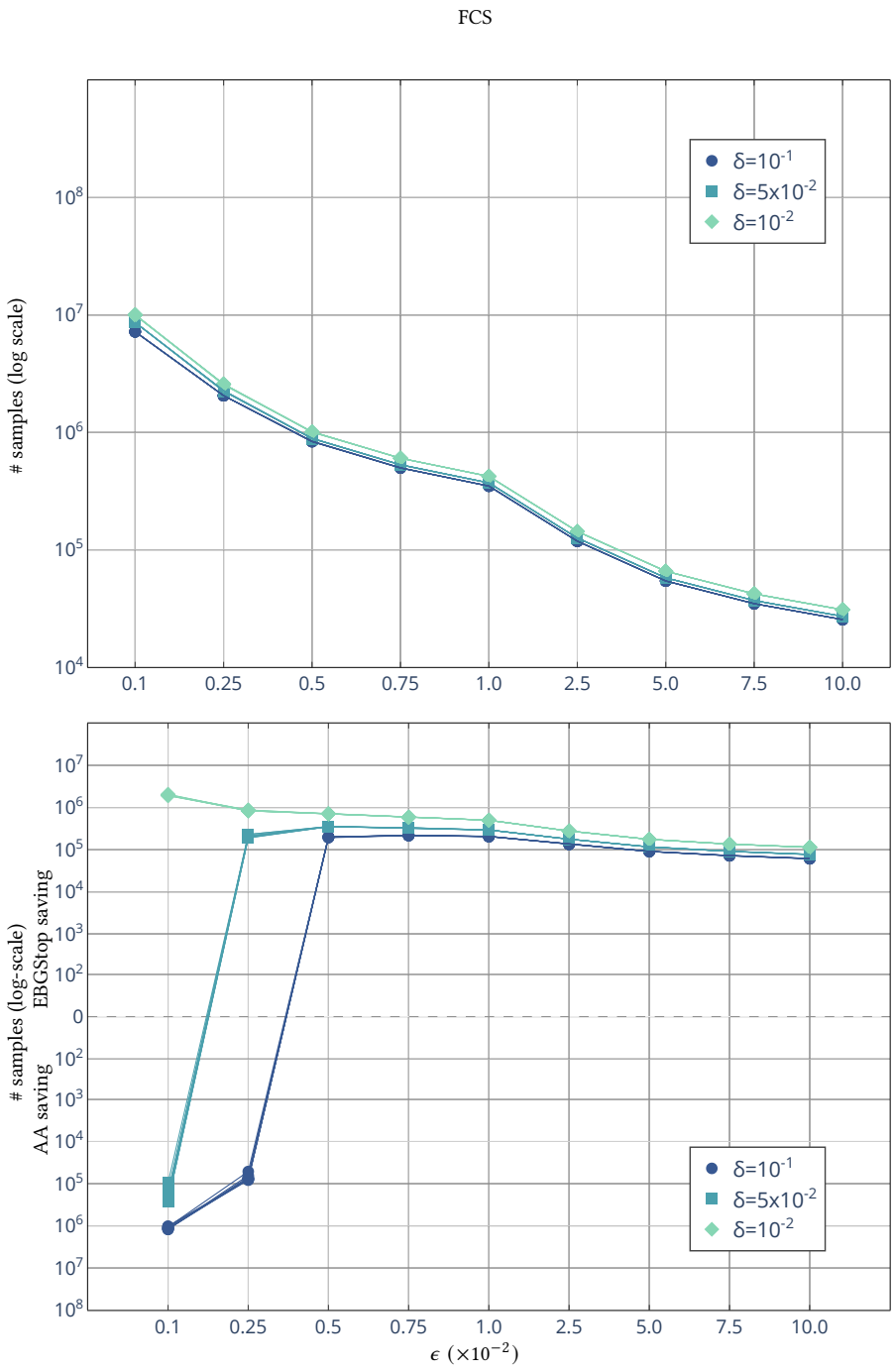


Figure 5.4: Top: average number of samples (vertical log-scale axis) required by EAA to converge, for various values of ϵ and δ. Bottom: average savings in the number of samples achieved by EAA in each setting wrt. the algorithm requiring the most samples.

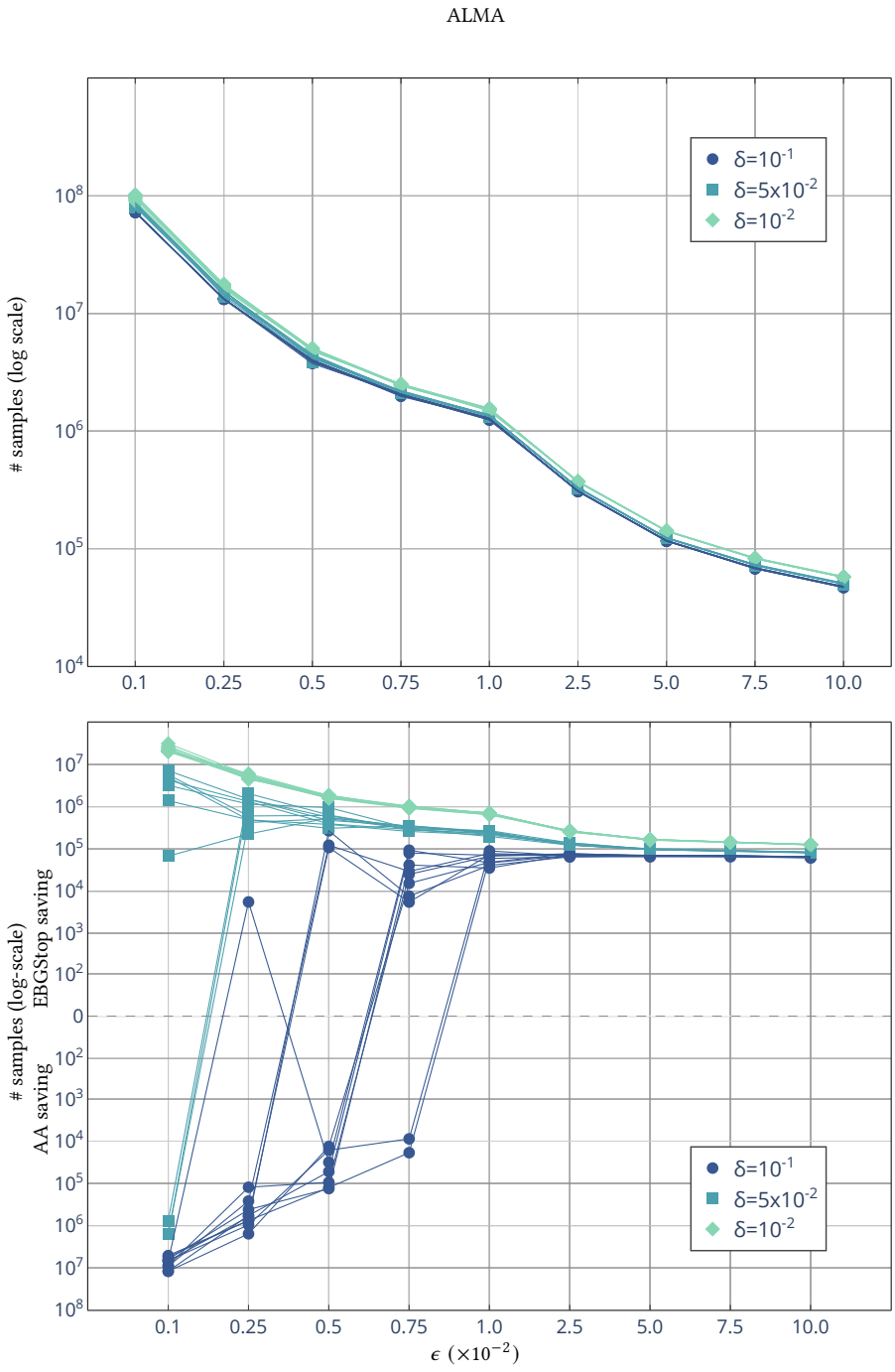
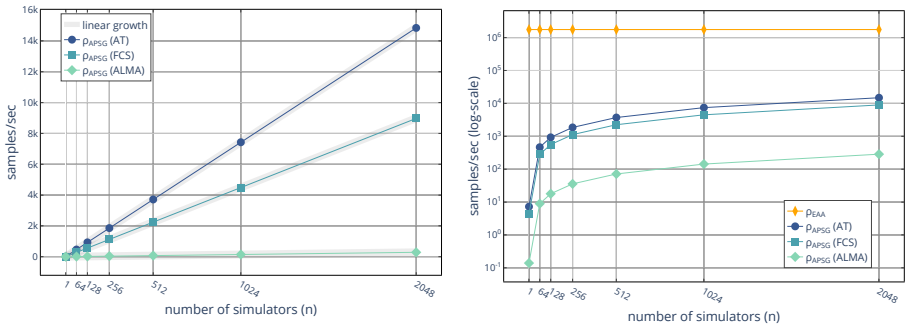


Figure 5.5: Top: average number of samples (vertical log-scale axis) required by EAA to converge, for various values of ϵ and δ. Bottom: average savings in the number of samples achieved by EAA in each setting wrt. the algorithm requiring the most samples.



(a) **(b)**

Figure 5.6: (a) Rates of sample production by APSG ($\rho_{\mathbf{APSG},n}$), for various numbers of available simulators (the grey curves are the *straight* lines passing by the two left-most points of each data series, and show the almost perfect linearity of the sample production rates wrt. n).
(b) Comparison (log-scale Y axis) of the sample consumption rate of EAA ($\rho_{\mathbf{EAA}}$) with the sample production rates of APSG from Figure (a).

of simulators, $n = 2048$, is deployed, in which case it is still more than 100 times larger). This means that the SAs module would spend almost its entire time waiting for APSG to produce the requested samples. Hence, an increase in the number of simulators is expected to result in an essentially proportional reduction in the completion time of the overall parallel tool. In other words, we expect our architecture to show an efficiency very close to 100% even when a large number of simulators are used. This suggests that adding more algorithms to the ensemble of SAs would not introduce any noticeable computational overhead, but could help in reducing the number of required samples for EAA if those additional algorithms happen to benefit from the statistical properties of the system KPI.

5.6.4.3 Overall Performance Scalability Analysis

To confirm the above statements, we give a performance scalability analysis of our parallel implementation, estimating (again with the method outlined in Section 5.6.3) its completion time if deployed on fully-reserved HPC infrastructures of various sizes.

Figure 5.7, Figure 5.8 and Figure 5.9 show the time that our tool would require to terminate on each of the 10 experiments for each case study, as a function of ϵ, δ, and the number n of parallel simulators (notice the left log-scale Y axes). Each figure shows, for each value of δ, the average completion time of the parallelization, with error bars indicating the minimum and the maximum values among the 10 experiments.

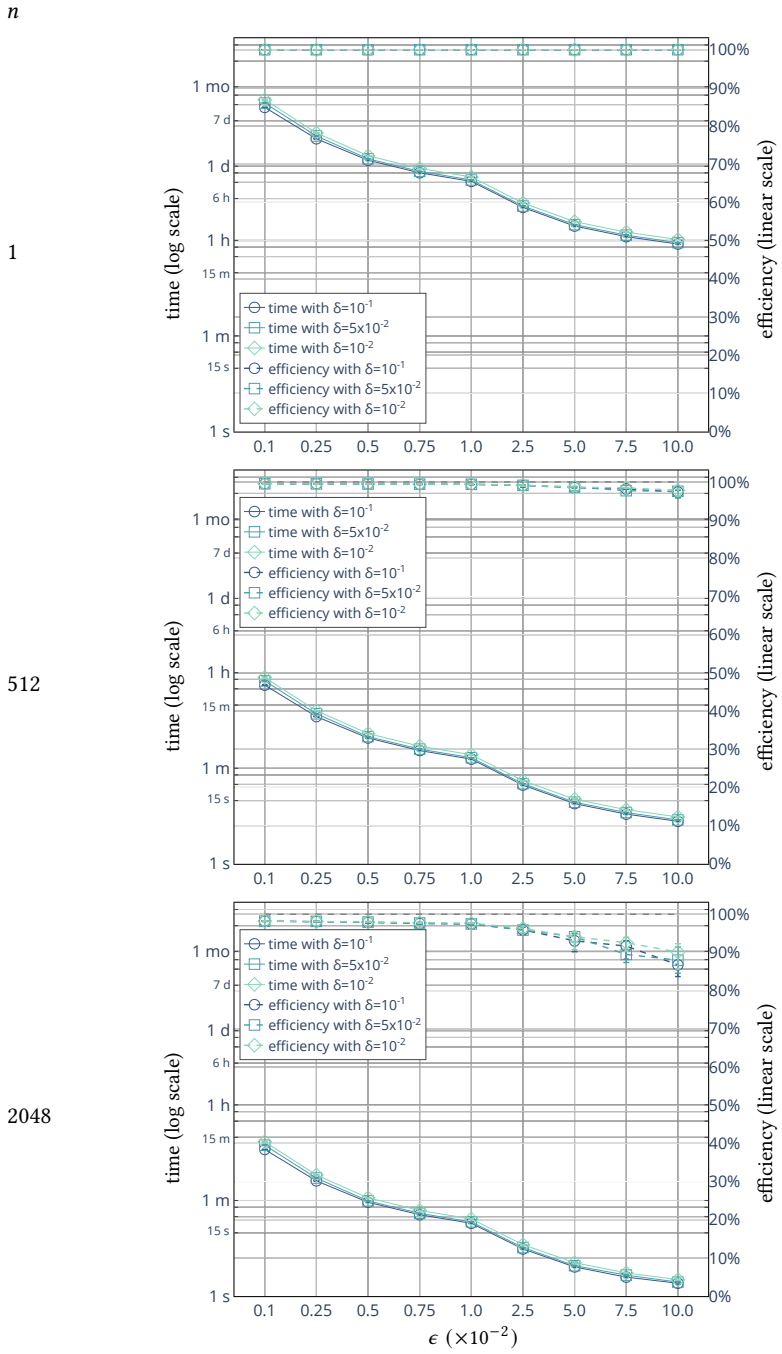


Figure 5.7: Scalability analysis of our parallel tool: overall completion time (bottom curves, left Y log-scale axis) and efficiency (top curves, right Y axis) for the AT case study and different settings and numbers of simulators $n = 1$, $n = 512$ and $n = 2048$.

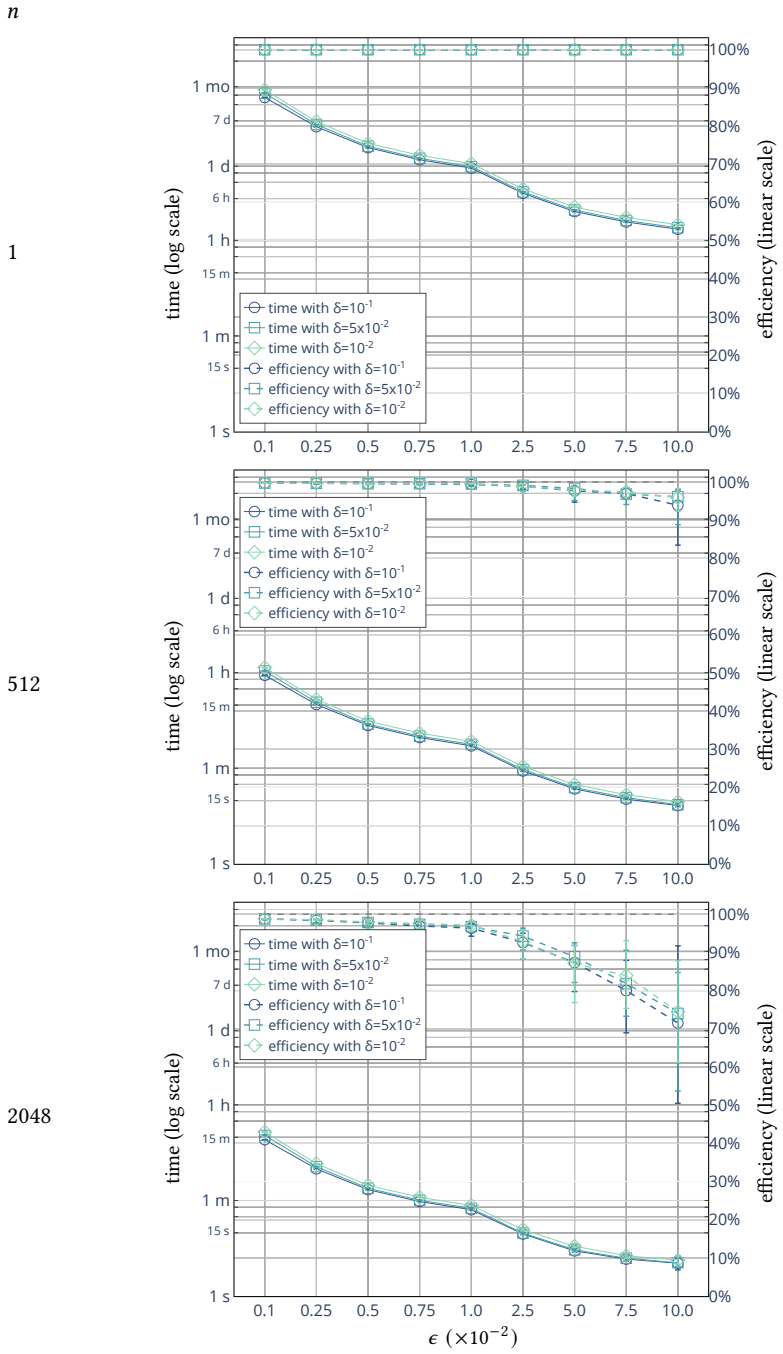


Figure 5.8: Scalability analysis of our parallel tool: overall completion time (bottom curves, left Y log-scale axis) and efficiency (top curves, right Y axis) for the FCS case study and different settings and numbers of simulators $n = 1$, $n = 512$ and $n = 2048$.

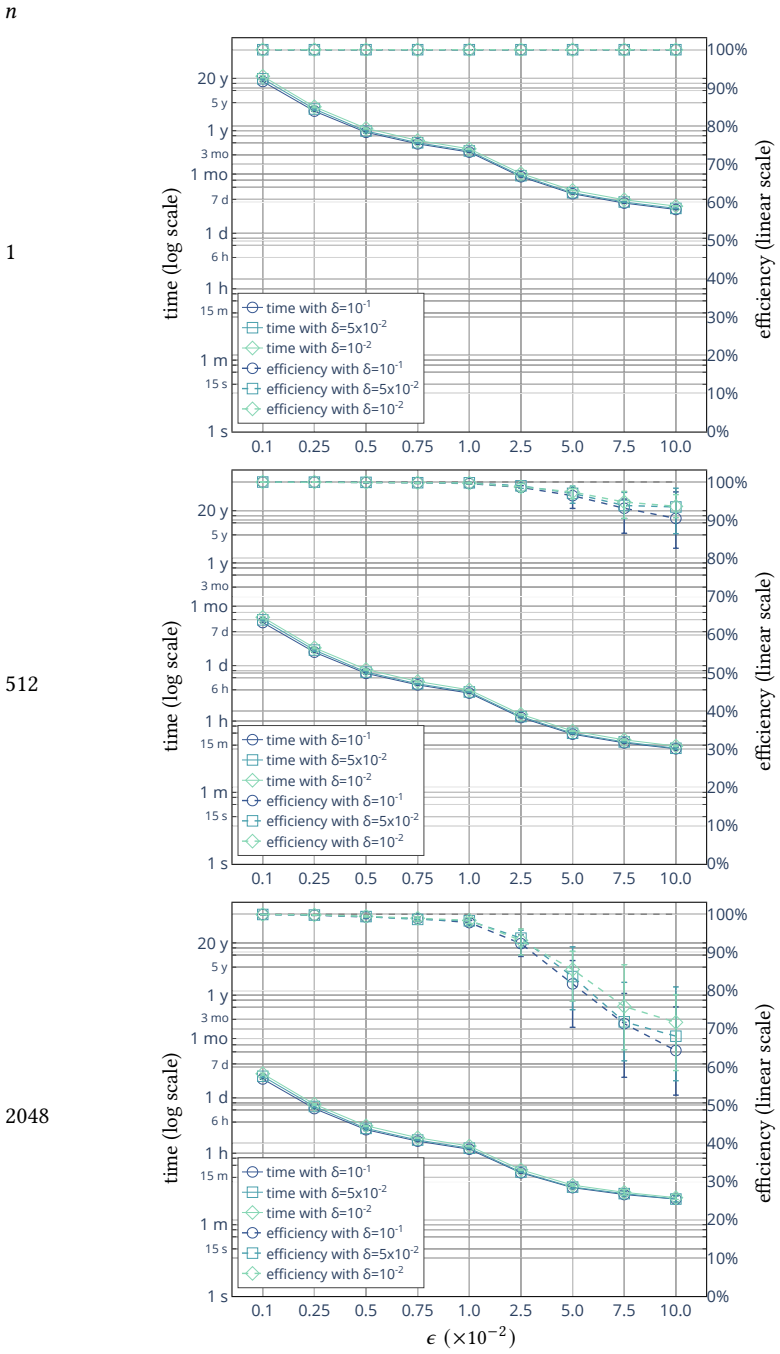


Figure 5.9: Scalability analysis of our parallel tool: overall completion time (bottom curves, left Y log-scale axis) and efficiency (top curves, right Y axis) for the ALMA case study and different settings and numbers of simulators $n = 1$, $n = 512$ and $n = 2048$.

Error bars are often tiny and barely visible, and this shows again the stability of the algorithm performance with respect to the random sequence of samples.

As expected, all the curves are qualitatively very similar, modulo the time reductions essentially proportional to n. Each plot in Figure 5.7, Figure 5.8 and Figure 5.9) also shows (see the curves at the top, to be read against the linear-scaled right Y axes) the average estimated efficiency of the parallelization (with error bars indicating the minimum and the maximum values among the 10 experiments).

The efficiency of the parallel computation involving n simulators is computed, as usual, as $\frac{t_1}{n \times t_n}$, where, in our case, t_n is the (estimated) time of the computation when using n simulators and t_1 is the time of *same* computation (*i.e.* same values for ϵ, δ and same random sequence of samples) when using a single simulator.

As expected, the estimated efficiency always sticks to $\sim$ 100%, except when the number of simulators (hence, the size of the envisioned HPC infrastructure) is too large for the problem at hand, in which case a significant number of samples is produced (in parallel) by APSG but never used by EAA. Although such circumstances are a clear symptom of an over-dimensioning of the hardware infrastructure, they do not pose a usability problem for the tool, since in this case the overall completion time is in the range of a few seconds or minutes.

5.7 Conclusion

There is no doubt that parallelization is (part of) the future of SMC. Anyone who wants to design a new SMC method with the goal of effectively providing a tool which can actually be exploited for the verification of industry-scale CPSs simply must develop a distributed version of their algorithm. There is no way around it. Moreover, they should systematically aim at combining and parallelizing their new approach with other known algorithms.

Researchers in the field are increasingly aware of that reality and distributed (sampling) methods exist in many tools already, such as those mentioned in the introduction of this chapter (UPPAAL-SMC, Modest, and Plasma Lab). Based on our current knowledge, it seems the idea of combining multiple algorithms and running them in parallel is less widespread, however.

In this chapter, after explaining why SAs form a class of approximation algorithms which are particularly well-suited for parallelization and discussing the right way to distribute a sampling process, we proposed EAA as an illustration of that strategy. We successfully implemented and tested a version of

EAA exploiting two optimal SAs, EBGStop and AA, together with a massively parallel architecture deployable over large HPC infrastructures. We exercised our tool on three industry-scale case studies, verifying specifications of non-trivial properties, and showed that parallelization can indeed make SMC a viable approach for the verification of complex systems, even when the number of required simulations can appear prohibitive from a sequential perspective. Finally, we exposed an in-depth experimental analysis of our tool, detailed the benefits of combining multiple SAs, quantified the effective reductions in sample sizes and execution times, and analyzed the efficiency of the parallelization and the impact of the number of parallel simulators. This allowed us to highlight how, in any non-trivial setting where the simulations represent the bulk of the execution time, one can expect an increase in speed and scalability of SMC methods which is directly proportional to the number of workers of the parallel architecture.

Future work As the conclusions of our analysis are relevant for the field of SMC as a whole, the parallelization of any existing or future SMC algorithm could be regarded as an extension of this work. Therefore, we will instead focus on how EAA could be expanded upon.

A direct extension of EAA would obviously be one which integrates other SAs, potentially making it even more versatile and applicable to a larger class of KPIs. For example, integrating the SA considered in Section 3.6.1 could improve its performance when evaluating KPIs associated to rare events.

To go further, our tool could be updated to allow for the verification of formula-based properties, expressed for instance in linear temporal logic. In its simplest form, this would be effortless, since the probability that a chosen property holds for a random possible execution of a system can be seen as a KPI of that system. A more ambitious plan would be to explore whether a strategy such as the one behind the elaboration of EAA could help for the verification of hyper-properties [Aro+22; Dob+23].

Repeating the analysis of the impact of parallelization on the effectiveness and applicability of SMC with a composite algorithm combining multiple optimization algorithms rather than simple estimation algorithms would be interesting as well. Identifying a class of optimization algorithms for SMC which share some of the key properties of stopping algorithms that make their parallelization relatively straightforward would be a first step.

Our tool could then also be enhanced even further by improving the sampling process itself, similarly to what has been done in [PLC23] (see Chapter 4), especially when taking into account geometric sampling.

Finally, comparison with what already exists will of course be a must, eventually. Unexpected difficulties (for example in terms of security and data integrity) could arise when developping a distributed version of our tool for

unprotected or faulty networks. Both in terms of SMC itself and practical considerations, examining the details of the implementations of the distributed tools of UPPALL-SMC and Modest should be very valuable.

Application 6

In this chapter, we present how the ideas of SMC can easily lead to great innovation when applied to original settings. We develop new strategies to design robust work schedules for CPSs by replicating classic strategies of SMC for a new class of system models: temporal networks. More precisely, we build a new framework to analyze the brittleness of task networks with respect to an arbitrary set of user-specified properties. The proposed algorithms allow the detection and enumeration of activities that, with modest task execution duration variation, make the success of execution no longer guaranteed in terms of the given set of properties. We also introduce a metric for measuring an activity's brittleness and describe how that measurement is mapped to task network structure. Those new tools allow us to propose a method for the analysis of system robustness to help human designers and automated task network generators to focus on and address sources of undesirable brittleness. Lastly, the bulk of this chapter is dedicated to the application of our approach to a set of task networks that were developed for NASA's Perseverance rover and to some of the challenges of biotechnology manufacturing.

The content of this chapter is related to the journal publication titled "Property-Based Brittleness Analysis of Temporal Networks":
T. Vaquero, S. Chien, J. Agrawal, M. Saint-Guillain, and M. Parmentier. "Property-Based Brittleness Analysis of Temporal Networks". In: *Journal of Aerospace Information Systems* 20.7 (2023), pp. 398–417.

6.1 Introduction

Temporal plans for agents that need to deal with execution uncertainty must be designed for execution robustness. Many real world operational contexts, ranging from planetary rovers to human operations (*e.g.* in biomanufacturing), match this type of problem due to their interaction with a hard-to-predict environment. In the planetary rover's case, plans (also called *task networks* (TN), see Section 2.3 for a reminder of the formal definition) must be carefully designed and generated on the ground to allow successful execution of tasks while meeting all the required timing and resource constraints, and being safe at all times. Accurately determining some of those timing constraints

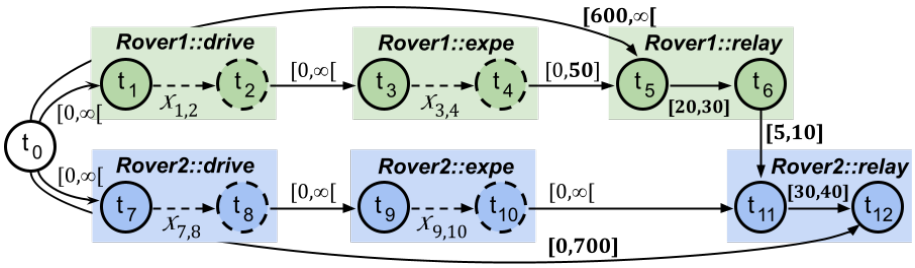


Figure 6.1: A simplified hypothetical sol on Mars for two planetary rovers, encoded as a PSTN. Bold: controllable. Dashed: contingent. Each rover has three activities in sequence: drive towards a science site, experiment, and relay results to an orbiter. A special time point t_0 represents the beginning of the operations. The rovers work independently during their driving and science activities. They do not coordinate until the communication time window, which strictly happens between time 600 to 700. Communication tasks cannot overlap, and *Rover1* is chosen to relay first. However, duration of driving and experimental activities are highly uncertain.

a priori, specially activity durations, is quite challenging though due to the natural unpredictability of the environment [RB17; Chi+18]. The traditional approach used in planetary rover missions is to add significant temporal margin for execution robustness; however, this negatively affects rover efficiency [Gai+16]. Ideally we would use plans that not only are consistent and maximize the vehicle's productivity, but that are also robust to unexpected events and delays. As a corollary, it is therefore critical to *evaluate robustness and identify activities that are brittle with respect to temporal unpredictability.*

Figure 6.1 shows a simplified example of a task network involving two planetary rovers, modeled here as a *probabilistic simple temporal network* (again, as defined in Section 2.3). The existing literature on PSTNs provide metrics in order to measure robustness, that is, evaluate how likely a task network execution is to succeed under temporal uncertainty (e.g. delays). However, it is hard to determine potential sources of brittleness when those metrics do not meet a desirable threshold. Suppose this is the case with our rover example of Fig. 6.1, where one is evaluating brittleness and gets 30% likelihood of succeeding; then in practice, the relevant questions about this PSTN include:

(a) What activities in particular are culminating in this low robustness, and why is that the case?

(b) How brittle these activities are to delays? And brittle with respect to what (e.g. temporal or resource constraints/property)?

(c) What are the possible ways we can change or relax some of the temporal constraints to bring both certain activities' and the entire task

network's brittleness values to an acceptable level?

Answering these questions is relevant not only for designer during task network development and authoring, but it is also relevant to automated planners supporting the process of finding robust plans. Herein we mostly focus on supporting the human designer, but the algorithm can be also applied to the automated synthesis of robust plans. We propose algorithms to address questions (a)-(c). Section 6.2 introduces the relevant background from the literature, leading to the existing PSTN robustness metrics. Questions (a)-(b) are addressed in Section 6.3. Question (c) is addressed in Section 6.4.3.

Brittle with respect to what? A property-based approach

Before answering questions (a)-(c), it is important to notice that these questions are not well framed without a context. Depending on the context, one may argue a task network to be robust in terms of the given temporal constraints; however in that specific case the challenge might be actually on the given resource constraints (*e.g.* energy consumption constraints), which may be significantly more problematic than deadlines when activities deviate from their nominal durations.

Most of the literature on the subject is focused on robustness as defined w.r.t. temporal properties. An example of property that is commonly found in the literature for task networks robustness analysis is the *dynamic or strong controllability* (*e.g.* [Akm+19]). Herein a task network holding such property means that there exists a scheduling policy/strategy that satisfies all the temporal constraints *in any possible situation imposed by nature*, and thus a successful execution is guaranteed (at least in terms of temporal requirements), despite the unpredictability of activity durations. An relevant example of non-temporal property for planetary exploration is that of rover *energy resource availability*, which should guarantee the rover's energy resource to remains above some predefined threshold, in order to meet the rover's safe requirements during execution. We re-frame the concept of *robustness* (and therefore *brittleness*) in terms of an arbitrary set of *user-defined properties* $P = \{P_1, \ldots, P_k\}$ to be verified over the considered task network.

Proposed framework

Given a set of user-specified (temporal and/or non-temporal) properties, we had the objective to address the challenge of identifying the activities that are most brittle to temporal unpredictability, in terms of the desired properties. In other words, the algorithms proposed in this chapter aim at detecting and enumerating activities that, with low levels of variation in their duration (*e.g.* a relatively small delays), prevent the task network from being guaranteed to

hold all or part of these properties in any possible situation. The main goal was to provide a more informative analysis that is complementary to existing robustness metrics, that goes beyond temporal aspects only, and that is fundamental for guiding designers and planners to increase robustness and address undesirable levels of brittleness in a more automatic way. To meet these desiderata, we generalized and extended previous work on temporal brittleness analysis [Vaq+19] to 1) incorporate a set of user-specified properties that are not necessarily temporal, and 2) to provide designers with options to relax some of the temporal requirements in the task network (*e.g.* activity start time and execution time windows) in order to accommodate their specified acceptable level of brittleness. In addition to the application cases, which include both a set of task networks in development for NASA's Perseverance rover and a real world biomanufacturing robust scheduling problem, our contributions are the following:

- A novel set of algorithms to analyze and measure brittleness of task networks with respect to a set of user-specified (temporal and non-temporal) properties, while detecting and ranking the most brittle activities;
- A visual tool for mapping brittle activities to a task network's structure that aims to help designers to identify temporal bottlenecks and sources of brittleness;
- An advisory tool for suggesting relaxation of task networks' temporal requirements to address specific undesirable levels of brittleness.

Moreover, by exploiting basic results from statistical property checking theory, we show how our algorithms in fact generalize to any possible set of properties $P = \{P_1, \ldots, P_n\}$ for an arbitrary confidence level set by the user. The only assumption is the existence, for each property P_i, of a simulation algorithm able to check whether P_i holds in a specific given scenario. We also show how, for some properties (*e.g.* dynamic controllability), our algorithms can be instantiated to make use of existing property checking methods (e.g. from [Cui+15]), hence providing a stronger (*i.e.* with total confidence) result whenever possible.

6.2 Background

In this section we use NASA's Perseverance rover (also know as M2020 rover) operation plans (called here task networks) not only to illustrate concepts in the relevant background, but also as an application of the proposed robustness and brittleness analysis. Figure 6.1 provides the flavor of task networks

we focus on in this work. Alternative operational contexts are demonstrated in Section 6.5 with the case of manufacturing in biotechnology. We first describe the main elements of the M2020 rover's task networks. We then go over temporal network formalism and the existing robustness metrics.

6.2.1 M2020 Rover's Task Network

M2020 rover landed successfully on the surface of Mars, specifically at the Jezero Crater, on February 18, 2021. The rover aims to seek signs of ancient life and collect samples of rock for possible return to Earth. Herein, we focus on M2020 *sol types* [Jet18], the current best available data on expected M2020 rover plans during a sol (a Martian day). We use the term *task networks* to refer to the M2020 sol types.

A M2020 task network represents the set of target science, engineering and communication activities and constraints for a given Martian sol. Based on [Chi+18; Agr+19], a M2020 task network is a tuple $TN = \langle A, H, SOC^{init}, SOC^{min}, SOC^{max}, SOC^{handover} \rangle$. Herein, we do not use any particular formalism to describe a M2020 task network, but rather focus on listing, naming and describing its main elements.

The main element of a task network refers to a set of rover activities $A = \{a_1\langle TC_1, D_1, e_1\rangle ... a_n\langle TC_n, D_n, e_n\rangle\}$, where for each activity $a_i \in A$:

- TC_i is the temporal constraint tuple $\langle TC_{i,dur}, TC_{i,est}, TC_{i,lst}, TC_{i,let} \rangle$ which includes the nominal (or predicted) duration $TC_{i,dur}$, earliest start time $TC_{i,est}$, latest start time $TC_{i,lst}$, and latest end time $TC_{i,let}$ (also called cutoff time);
- D_i is the set of the activity's dependency constraints in the form of $a_i \rightarrow a_j$, meaning a_i depends on a_j, i.e. the end time of a_j must be before the start time of a_i;
- e_i is the rate at which the energy is consumed/generated by activity a_i.

All activities in A are *mandatory* and must be scheduled; therefore, no disjunction is considered in this work. In addition to the set of activities A, our task network has the following elements:

- H is the execution horizon that constrains when all the activities have to finish by;
- SOC^{init} is the initial state of charge (SOC) of the rover at the beginning of the sol, i.e. at the start of task network execution.
- SOC^{min} is the global minimum state of charge that is required at all times during execution, i.e. the battery charge cannot go below that minimum during rover's operation on that sol.

- SOC^{max} is the global maximum state of charge during execution, i.e. the battery charge will never exceed the maximum amount during charging (sleep mode) - when the maximum is reached, the rover's battery gets into a shunting mode in order to avoid overheating and time waste.
- $SOC^{handover}$ is the handover minimum state of charge that is required at the end of the task network execution, i.e. the battery charge cannot be below that minimum at the end of rover's operation on that sol.

It is worth noting that the original Mars 2020 task network described in [RB17; Chi+18; Agr+19] also includes an activity's constraint related to the use of instruments (in this case specifying that only one activity can use the required instrument at a time), but here we will only consider the energy resource for simplification.

The duration of planetary rover activities can be quite unpredictable (see [Gai+16] for analysis of MSL variability) and it is expected that M2020 will have similar uncertainty associated with its activity durations $TC_{i,dur}$. Knowledge about rover's activity duration is hard to acquire *a priori*, but partial knowledge might exist in the form of lower and upper bounds (set bounded) or a probability density function (pdf). Under those assumptions, temporal networks are a suitable foundational formalism to analyze the aforementioned M2020 task networks.

6.2.2 Formalization of the M2020 Rover's Task Network with Temporal Networks

We have already introduced the basics of temporal network theory that are relevant to this work in Section 2.3. We now relate those notions to the representation elements to the aforementioned M2020 rover's task network.

Simple Temporal Networks (STN). In the context of M2020 task network, one could represent the temporal requirements using a STN model. Each activity $a_i \in A$ has two corresponding time points representing its start time and end time. A special time point $t_0 = 0$ represents the start time of the plan execution. The earliest and latest start time constraint of an activity a_i are encoded as a temporal constraint of the form $(t_{i\_start} - t_0) \in [T_{i,est}, T_{i,lst}]$, while the constraint $(t_{i\_end} - t_{i\_start}) \in [T_{i,dur}, T_{i,dur}]$ would represent the duration of the activity as being exactly $T_{i,dur}$. Moreover, the latest end time of an activity is represented by the temporal constraint $(t_{i\_end} - t_0) \in [0, T_{i,let}]$ (activity's earliest end time is not specified in our task network and, therefore, not considered here).

STN with Uncertainty (STNU). In the M2020 task network context, an STNU representation can be used to capture the uncertainty related to activity du-

rations. All or some of activities' duration can be encoded as contingent constraints in C_C.

Probabilistic STN (PSTN). In the M2020 task network case, the duration of certain rover activities (e.g., drive) can also be represented as a random variable. This is the particular representation we have chosen for our experiments.

6.2.3 Controllability

An STN is said to be consistent *iff* there exists a schedule (an assignment to the time points T) that satisfies all the temporal constraints [DMP91].

In the STNU and PSTN realm, consistency is not directly applied due to the unpredictable assignment of contingent time points T_C and edges C_C. A STNU/PSTN relies instead on checking *temporal controllability* [VG02], which verifies whether an agent can generate valid schedules to any situation that may arise in the external world.

Controllability theory is usually applied to STNU where different levels of controllability can be checked. An STNU is said to be *strongly controllable* [VG02; VF99] *iff* there exists at least one "universal" schedule that fits any situation: an assignment to the executable time points T_E is guaranteed to exist and to satisfy all temporal constraints regardless of the nature's assignments to contingent time points (revealed during execution). A strongly controllable task network is applicable in cases where agents have to compute a schedule offline before making any observations and don't have the opportunity to adapt online. Nevertheless, these cases are not usually practical in dynamic and unpredictable environments. A more practical level would be *dynamically controllable* [MMV01; Mor14], in which we check whether there is an execution strategy that at any time during execution, the partial sequence executed so far is ensured to extend to a complete solution whatever durations remain to be observed/revealed from the contingent activities. In this case, an agent would be able to choose online a valid assignment of executable time points based on observed past contingent time points, without violating any future temporal constraints. Controllability analysis can also be applied in PSTN when bounds can be established from the pdfs, which in turn incurs some risk of contingent constraints falling out of those bounds [FYW14]. [Sai+21] provided the first formal definitions of both strong and dynamic controllability, as well as other controllability levels.

Dynamic controllability can also be one of the user-specified properties in the proposed brittleness analysis. Herein, we rely on existing temporal controllability checkers to support the analysis.

6.2.4 Robustness Analysis

There are several existing work that measures robustness of a temporal network due to temporal uncertainty of the environment (e.g., unforeseen delays). Different metrics are used to measure robustness. In this section we will cover the main metrics and approaches found in the literature.

In [COS98; Wil+14], robustness is computed by measuring an STN flexibility: a metric that quantifies the aggregate slack in a STN. In the naive approach to compute a STN flexibility ($flex_N$), Floyd-Warshall's all-pairs-shortest-path algorithm is first used to infer the tightest bounds over the STN's temporal constraints C and, consequently, the earliest time ($est(t)$) and latest time ($lst(t)$) for all time points $t \in T$. The flexibility for scheduling time point t would be defined as $flex_N(t) = lst(t) - est(t)$. The flexibility of an STN S would be then $flex_N(S) = \Sigma_{t \in T} flex_N(t)$ (e.g., $flex_N(S) = 100$ secs). This naive approach ignores possible correlations that might exist between starting times of activities, which might result in over estimating flexibility [Lun+17]. In order to avoid double counting, [Wil+14] propose a more suitable approach for STNs in which a linear program formulation is used to maximize the sum of the time intervals each time point t can be scheduled in, under temporal constraints C. In [Hua+18], an STN is represented as a polyhedron and the flexibility linked to its volume. Similarly to flexibility, the term durability is introduced in In [LOB19] as the ability of a schedule generated for an STN to withstand disruptions within the constraints of the given STN. A geometric view of the solution space of a given STN is used to provide the intuition for the durability: a solution's distance from the boundaries corresponds to the measure of the solutions' ability to handle disturbances while satisfying all the temporal constraints defined in the STN. Although useful, it is potentially hard to judge from the above metrics if the amount of flexibility is enough in certain unpredictable environments – it might not necessarily imply robustness [Cui+15]. Moreover, the above methods do not inform where exactly more flexibility is needed in the temporal network if the metric value is below the acceptable range.

In the STNU realm, one can use the strong and dynamic controllability checking methods as an indication, or as a property, to decide whether a task network is robust. However, when the STNU is not deemed controllable, or when the activity duration bounds might vary or are defined stochastically, controllability alone no longer provides an informative measure of robustness. In what follows we go over measures that support these cases.

The work described in [Cui+15] defines robustness as the greatest level of disturbance (deviation from expected outcomes) on an STNU at which it is still successfully executed. Computing robustness is framed as a linear constraint optimization problem to compute the maximum deviation from the

nominal case (i.e., width of the contingent bound) on any activity at which the STNU is dynamically controllable. A particular application of this approach is the maximum delay problem, in which the goal is to compute the maximum possible delays of all contingent activities (using the nominal duration as a reference) and identify the smallest possible delay across all those contingent activities. That smallest delay is the metric for measuring robustness, and in this case uncertainty is explicitly modelled and considered to compute robustness. The lower bound on the contingent activity duration is set to the nominal duration and the delay is the varying upper bound of the contingent interval. If a PSTN is used instead as input in this approach, the level of disturbance at which the schedule becomes no longer dynamically controllable corresponds to the probability of a failure during execution. The same constraint optimization problem framework applies in the probabilistic case, in which the objective is to maximize the probability that the contingent activity durations fall within that maximum delay. The robustness metric in this case would be a probability of execution success (e.g., a given PSTN is 97% robust).

In [Akm+19], the *degree of strong controllability* (DSC), as well as the degree of dynamic controllanility (DDC), of a STNU is introduced as an estimate measure of robustness. This measure is particular relevant for cases where one is trying to evaluate not only if an STNU is strongly (resp. dynamic) controllable or not, but also how "far" it is from being controllable if it is proven to be not strongly (resp. dynamic) controllable. The intuition here is that STNUs with lower DSC are "less strongly controllable," while STNUs with DSC closer to 1 are "more strongly controllable". For example, a strongly controllable STNU would have DSC equal to 1. The DSC is then defined as the proportion of contingent edge realizations in which the temporal network remains strongly controllable. A parallel could be done wrt PSTN, in which the DSC or DDC would rather be defined in terms of probability mass of the respective controllable realizations. The method proposed in [Sai19] attempts at computing an exact lower bound on the DDC, while handling ordinary distributions. Yet, it applies to particular PSTNs only, leading to an approximation in the general case. [Sai+20; Sai+21] then proposed a general method for computing DDC lower bounds of any PSTN, computable in fixed-parameter pseudo-polynomial time, and propose the first exact formal (mathematical) definitions of DSC and DDC. We also focus on evaluating task networks against the dynamic controllability. However, we consider properties to evaluate robustness, i.e. others than conjunctive temporal ones.

In [Tsa02; Bro+15; Sai19], robustness is also quantified as the likelihood that a probabilistic task network will be executed successfully. In [Tsa02], a naive robustness measure, $Robustness_N$, is computed by first using a shortest path algorithm to determine the tightest bounds for each contingent con-

straint $c_{i,j} \in C_C$; then it uses those bounds to compute the area under the corresponding pdf f between those bounds. Applying it to all contingent constraints, the resulting naive metric would be $Robustness_N = \prod_{c \in C_C} \int_{l_{i,j}}^{u_{i,j}} f_{i,j}(x)dx$. The naive approach assumes no correlation between the temporal constraints, and as before, suffers from over estimation. To account for interrelations, [Bro+15] proposed a Monte Carlo approach in which the execution of a PSTN is simulated multiple times and the duration of contingent activities is sampled as they are executed. Herein, the simulated execution uses dynamic controllability techniques to determine how it should adapt to the sampled values of durations. The ratio of successful executions corresponds to the *Robustness* value. The work in [Sai19] looks into evaluating robustness of a solution to single-machine scheduling problem with stochastic processing times. Herein, robustness is computed as the probability of a schedule to remain feasible with respect to temporal constraints and unary constraints.

Similar to the likelihood of success, the *risk* of violating any temporal constraints in a PSTN can also be interpreted as a robustness metric [FYW14]. Risk is in fact the complement of robustness measures mentioned above.

It is worth mentioning the body of work on sensitivity analysis for scheduling [Sto91; Ali+03; HP04] that studies parameters variation in the problem specification (including task duration) and its impact on scheduling performance (e.g., makespan, solvability). In [Sto91] for instance, authors measure the maximum amount of parameters change that maintains schedule optimality. In contrast, we focused on the impact of temporal disturbance (e.g. delays) with respect to *execution* as opposed to scheduling performance.

The aforementioned metrics (controllability, success probabilities, degree of controllability and risk) provide an intuitive way to evaluate if an STNU or PSTN execution is likely to succeed and whether the entire network is brittle to disturbance or not. However, they focus on evaluating robustness with respect to temporal properties only, specifically strong or dynamic controllability. We focus on generalizing robustness measures to cover non-temporal properties.

6.2.5 Synthesis of Robust Plans

The commonly used approach to synthesize plans for execution robustness in the literature has been to incorporate (strong or dynamic) controllability properties (in the form of constraints) into the planning process, that is, at STNU creation. Some of the existing methods interleaves search with verification of strong or dynamic controllability, e.g. (partially) translating the original planning problem to STNU during search to check whether candidate solutions are controllable [Bit16; Dvo+14]. In [NKD18], a constraint satisfaction problem (CSP) solver and an STNU solver are integrated to handle the

search over candidate plans that must satisfy both dynamic controllability and consumable resource constraints. In [Abr+19], the SREA system transforms the input PSTN into a strongly controllable STNU while approximately minimizing the probability of failure. Controllability and resource constraint is also considered in [Umb+18].

In [San+16a], the risk metric is used to guide the search for a robust plan, where a risk-minimization approach is used to generate a schedule under strong controllability constraints, in which a user-specified risk is encoded as one of the components of the optimization function. The work in [Lun+17] describes two algorithms for computing optimal dispatch strategies that maximizes robustness based on risk bounds: the Static Robust Execution Algorithm (SREA) and Dynamic Robust Execution Algorithm (DREA). Herein both approaches uses linear programming to solve the optimization problem. SREA for example transforms the input PSTN into a strongly controllable STNU while approximately minimizing the probability of failure.

The traditional usage of controllability constraint is quite helpful on driving the search for robust plans. That provides a single-value metric to evaluate the schedule candidate during search. Allowing the planner to more deeply evaluate a schedule candidate and avoid specific robustness issues is promising. For example, a planner could search using a supporting analysis that would measure brittleness of individual activities in the plan and identify particular activities and constraints that negatively impact the candidate plan robustness. In what follows we describe algorithms that could be used as such supporting analysis. As previously mentioned, we focus on providing these algorithms to human designers, leaving their use in the context of automated planner for future work.

6.3 Task Network Brittleness Analysis

We propose a brittleness analysis approach for PSTNs, based on a set of user-specified *properties*.

Our approach 1) enumerates activities, from most brittle to least brittle with respect to the target properties (e.g. Dynamic Controllability), 2) informs which property each activity is brittle to, 3) support detecting the causes of potentially undesirable brittleness levels, and 4) provide suggestions about how to adapt temporal constraints to reduce brittleness.

In this work, to simplify formulations we represent temporal uncertainty by associating a normal distribution (i.e. *mean* (μ) and a standard deviation, *sigma* (σ)) (e.g from a normal distribution) to the random variables assigned to every contingent activity a's duration in the PSTN. In practice, provided very little information about the activities, normal distributions are often used, with parameters fixed arbitrarily, estimated experimentally or adjusted de-

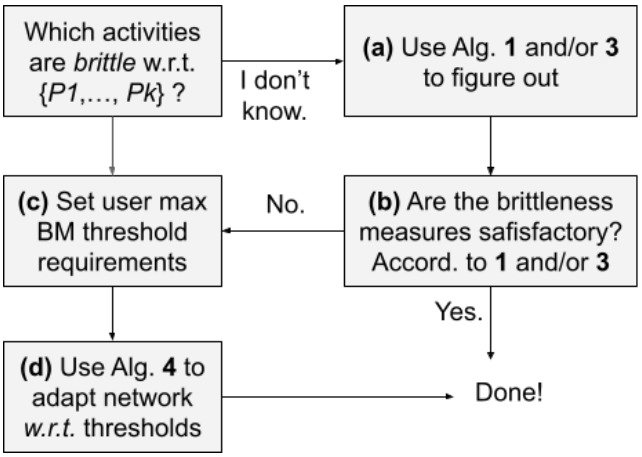


Figure 6.2: The proposed task network brittleness analysis framework.

pending on practical constraints. This is also the case for the Mars 2020 application. We could however generalize to any well-known family of distributions.

We call the set of contingent activities A_C and the non-contingent, controllable activities A_R, where $A = A_R \cup A_C$ and for every activity $a_i \in A_C$ its temporal constraints tuple is now represented as $TC_i = \langle \mu_i, \sigma_i, TC_{i,est}, TC_{i,lst}, TC_{i,let} \rangle$. In Figure 6.1, drive and experiment activities would be refereed to contingent activities, while relay activities would be referred to controllable activities.

6.3.1 The brittleness analysis framework: the full picture

The proposed analysis framework aims at providing solutions to questions (a)-(c), mentioned in Section 6.1. The full picture/process is depicted in Figure 6.2. Provided a task network *TN*, and in particular a set of user-specified *properties* $P = \{P_1, \ldots, P_n\}$, we design Alg. 1 and 3 to determine **(a)** the level of brittleness of each activity within *TN* with respect to each property. Thereafter, depending on the results, the user may consider **(b)** the brittleness level of some $a_i, \ldots, a_j$ activities as too risky, in which case they would **(c)** impose a *maximal brittleness measure* (BM) on each target activity, and eventually use Alg. 4 in order to **(d)** compute the adequate temporal constraints relaxations that satisfies these BM requirements. Informally, Alg. 4 finds the most reasonable adaptations of the network temporal constraints (e.g. earliest start times or latest end time), such that the defined maximal brittleness thresholds are met.

6.3.2 User-specified Properties

User-specified properties are key to our approach and are one of the main differentiators from preexisting work. Herein we consider a set of properties, $P = \{P_1, \ldots, P_n\}$ as an input to the brittleness analysis, where each P_k may or may not hold in all situations imposed by nature during a task network TN execution. Checking TN against the set P means checking whether the conjunction of all its elements is satisfied. Our analysis generalizes robustness analysis from a single temporal property to multiple temporal and non-temporal properties.

As mentioned in Section 6.2, a common property used in literature to analyze robustness in different applications is the *Strong or Dynamic Controllability*. Examples of properties that one would need to consider in the M2020 rover case would be: a) *Dynamic Controllability*, P_{DC}; b) *Global Minimum State of Charge*, $P_{SOC^{min}}$; and c) *Handover Minimum State of Charge*, $P_{SOC^{handover}}$. In $P_{SOC^{min}}$, engineers are looking into whether the minimal state of charge of the rover's battery never drops below a specified level (i.e. no violation) in any activity delay situation that might occur. Similarly, in $P_{SOC^{handover}}$, engineers are interested in checking whether the rover's state of charge at the end of the sol operations (handover to the next sol) is no less than a minimal specified value in any delay scenario. Both $P_{SOC^{min}}$ $P_{SOC^{handover}}$ are resource related properties while P_{DC} is purely temporal. We will expand on those properties in Section 6.5.

6.3.3 Property checking algorithms

Let $Check(TN, P_k)$ equal one if a property P_k holds in a task network *TN*, in any situation imposed by nature, zero otherwise. Deciding $Check(TN, P_k)$ is hard in general. Depending on the type of the properties considered, efficient decision algorithms may however be designed. We call these *complete checking algorithms*. For example, both strong (SC) and dynamic controllability (DC) has been shown to be computable in polynomial time [MMV01; Mor14; Cui+15]. The global and handover minimum state of charge properties also benefit from dedicated decision methods, further described in Section 6.5.

However, for any other property P_k which is not efficiently decidable, or does not benefit yet from a complete checking algorithm from the literature, we can exploit SMC in order to determine, with an arbitrary predefined confidence level, whether P_k holds for our *TN* in any possible situation. The only required assumption in this work is the existence of an algorithm $\Phi(TN, P_k \xi)$ that returns one if property P_k holds for *TN* in a given scenario ξ, zero otherwise. A scenario ξ is defined by the realizations of all the contingent task durations and, when P involves nonanticipativity, the moments at which each duration realizes. Provided all the durations and realization times, the prob-

lem becomes fully deterministic and $\Phi(TN, P_k, \xi)$, which must be seen as a verification function about a simulated possible scenario, is assumed to be computable (otherwise the proposed statistical model checking method is not applicable for P_k). For instance, if one is interested in the SOC^{max} property, then checking $\Phi(TN, SOC^{max}, \xi)$ under a given scenario ξ amounts at simulating the entire execution of TN provided the execution durations realized in ξ, eventually returning 1 if the maximal state of charge is never exceeded in that scenario, 0 otherwise. In practice however, depending on P_k, the decision function $\Phi(TN, P_k, \xi)$ may not be efficiently computable.

In what follows, we adopt a top-down didactic approach, by first describing the proposed brittleness analysis algorithm, provided $Check(TN, P_k)$ as a black box. Note that the proposed analysis framework is only applicable in the case of properties P_k for which either a complete checking algorithm exists, or the associated $\Phi(TN, P_k, \xi)$ function is efficiently computable. Whereas the complete checking algorithms considered are all provided by the literature, statistical model checking methods are described at the end of the section.

6.3.4 Ceteris Paribus Analysis

In a nutshell, we use a *ceteris paribus* approach (i.e. "all else unchanged"), to analyze the brittleness of each contingent activity individually against the set of properties P by exploring different temporal disturbance (e.g. delays) scenarios and evaluating which properties are violated and how interrelated an activity's brittleness is to other activities temporal disturbance. We then overlay that information onto the task network structure to analyze why certain activities are more brittle than others. We first introduce the core analysis algorithm.

In the core of the brittleness analysis, we introduce a *ceteris paribus* approach to measure the degree of disturbance (deviation from the mean) at which each target contingent activity $a_i \in A_C$ makes the task network violate one or more properties $P_k \in P$. In the P_{DC} case, activity $a_i \in A_C$ duration deviation from the mean would be that at which the task network become dynamically uncontrollable.

For each activity, we measure its maximum duration deviation from the mean (e.g. maximum lower bound and maximum upperbound, delay, from the mean duration), while fixing the uncertainty of all other remaining contingent activities to a certain level, i.e. to a certain lower and upper bounds on their duration. The process of fixing the uncertainty of all other activities results in what we call here a *ceteris paribus scenario*.

Deviation from the mean here is related to a *sigma multiplier* that is used to determine the lower and upper bounds of an activity's uncertain duration around the mean. Such a sigma multiplier is defined as follows:

Definition 6.1 (Minimum sigma multiplier)**.** *The* minimum sigma multiplier $z_i^u \in \mathbb{R}_{>0}$ *is defined as the smallest sigma multiplier of a contingent activity* a_i *that makes the task network violate one or more properties* P*.*

Note that if a contingent activity a_i has a minimum sigma multiplier z_i^u, the activity duration's lowerbound and upperbound for that particular activity are $[\mu_i - z_i^u \times \sigma_i, \mu_i + z_i^u \times \sigma_i]$ for a particular *ceteris paribus* scenario. Such a sigma multiplier z_i^u can be associated to set of properties $P^{i,u} \subseteq P$ that are violated.

The above definition assumes that the activity duration uncertainty is centered around the mean. While this is certainly the case for normal distributions, many other distributions do not have this property. Our approach would cover asymmetry since we are interested in the minimum sigma multiplier. For instance, we could consider *upper* or *lower* minimum sigma multipliers. If a contingent activity a_i would have an upper or lower minimum sigma multiplier z_i^u, the activity duration's bounds for that particular activity would be $[-\infty, \mu_i + z_i^u \times \sigma_i]$ or $[\mu_i - z_i^u \times \sigma_i, \infty]$ a for a particular *ceteris paribus* scenario, respectively. This would allow us to deal with one-sided uncertainty.

The intuition here is: the smaller the z_i^u, the more brittle the activity is in the network. Note that if the mean and sigma are associated to a normal distribution, the sigma multiplier z_i^u is linked to the probability mass, p_i^u, under the probability density function (*pdf*) within $[\mu_i - z_i^u \times \sigma_i, \mu_i + z_i^u \times \sigma_i]$: hence the smaller the probability p_i^u, the more brittle is the activity.

Measure of Brittleness. We propose to consider the minimum sigma multipliers z_i^u (or p_i^u in the probabilistic case) as our central *measure of brittleness* and the means to identify brittle activities. We use the terms 'degree of brittleness' or 'brittleness level' interchangeably to refer to that measure.

Brittleness. Therefore, the britleness of a task network should be understood as follows: the sensitivity of the potential of the task network to have (global) solutions with respect to the (local) uncertainty associated with each specific contingent activity. In other words: it's a measure of how the variability of specific task durations can impact the possibility of finding a schedule which allows all tasks to be completed within time.

In what follows we describe the proposed analysis algorithm to compute z_i^u for each target contingent activity.

Algorithm 9 is designed to order activities by their degree of brittleness. The input are the following: a task network TN; the set of user-specified properties P we want evaluate TN against; and a vector of initial sigma multiplier Y that will be used here to set a *ceterus paribus scenario* when analyzing each individual activity, i.e., when analyzing a particular contingent activity

Algorithm 9 Ceteris-Paribus-Brittleness-Analysis

Input:

A Task Network TN

A set of properties P

A list/vector of sigma multipliers Y, containing one multiplier for each contingent activity in TN, used to create the *ceteris paribus* scenario

Output:

A list of contingent activities ordered by their corresponding z_i^u, along with their respective z_i^u values and the set of properties P_i^u that were violated at that sigma multiplier

1: $Z^u \leftarrow \{\}$
2: **for** $a_i \in A_C$ **do**
3: $\quad z_i^u, P_{i,u} \leftarrow ComputeActivitySigmaMultiplier(a_i, Y, TN, P)$
4: $\quad Z^u.append((a_i, z_i^u, P_{i,u}))$
5: $Z^{u,ordered} \leftarrow order(Z^u)$
6: **return** $Z^{u,ordered}$

a_i we will use sigma multiplier Y to set the duration lower and upper bounds of all other activities and therefore creating a scenario for a_i. We will see later that we can manipulate Y to create different scenarios when analyzing each activity a_i. A special case of Y is when all its elements are zeros, called here Y^0, meaning that the lower and upper bounds of all the other contingent activities are set to their nominal/mean value (no uncertainty). We will later elaborate on that special case.

The most important process of the Algorithm 9 occurs in line 3.

For each activity $a_i \in A_C$, we compute its sigma multiplier z_i^u and the corresponding set of violated properties $P^{i,u} \in P$, and append the tuple (a_i, z_i^u, Pi, u) to the multipliers results Z^u. The function $ComputeActivitySigmaMultiplier$ (line 3), represented in Algorithm 10 starts by first creating a new TN' in which all other contingent activities $a_j \in A_C (j \neq i)$ have their contingent duration bounds set to $[\mu_j - y_j \times \sigma_j, \mu_j + y_j \times \sigma_j]$ (where $y_j \in Y$). We then search for the minimum sigma multiplier z_i^u applied to a_i's duration bounds that violates one or more elements of P by using $Check(TN, P_i)$ in every new TN' generated, as illustrated in Figure 6.3 (a). The search for z_i^u can be done in different ways. One can incrementally add uncertainty delta to activity a_i's duration by applying an increasing sigma multiplier z_i until it hits z_i^u (as shown in Algorithm 10). Another approach would be to use a dichotomic search. More elaborated approaches for finding z_i^u could be constructed, but it is out of the scope of our research since both the incremental or the dichotomic search approaches suffice for the purpose of this work.

Algorithm 10 ComputeActivitySigmaMultiplier

Input:

A contingent activity a_i

A list/vector of sigma multipliers Y, containing one multiplier for each contingent activity in TN

A Task Network TN

A set of properties P

Output:

a_i's sigma multiplier z_i and the set of properties P^i that are violated at that sigma multiplier

1: $z_i \leftarrow 0$
2: $P^i \leftarrow \{\}$
3: $isPviolated \leftarrow False$
4: $TN' \leftarrow$ initialize the duration lowerbound and upperbound of activity a_i to its respective mean, and for all other contingent activities a_j in TN set their duration lowerbound and upperbound to their corresponding sigma multiplier in Y, i.e. $[\mu_j - y_j \times \sigma_j, \mu_j + y_j \times \sigma_j]$ where $y_j \in Y$
5: **while** $isPviolated == False$ **do**
6: $z_i \leftarrow z_i + \Delta$ ▷ increments z multiplier, different Δ can be used
7: $TN' \leftarrow$ set duration lowerbound and upperbound of activity a_i to be $[\mu_i - z_i \times \sigma_i, \mu_i + z_i \times \sigma_i]$
8: **for** all $P_k \in P$ **do**
9: **if** $Check(TN', P_k) == 0$ **then**
10: $P^i.append(P_k)$
11: $isPviolated = True$
12: **return** z_i, P^i

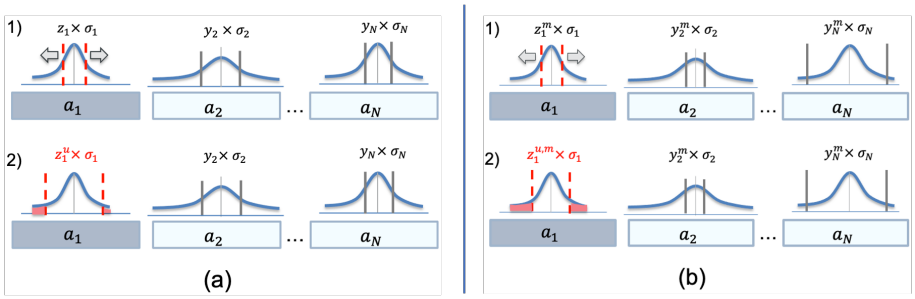


Figure 6.3: (a) Algorithm 9 determines the sigma multiplier z_i^u for each target contingent activity $a_i \in A_C$ while applying a sigma multiplier vector Y to all other contingent activities a_j ($j \neq i$) (see Algorithm 10): a.1) represents the search process for the minimum sigma multiplier z_i^u for activity a_i; a.2) represents the point in which z_i^u is found (and consequently the subset of properties that are violated $P^{i,u} \subseteq P$ is also found). (b) Algorithm 11's Monte Carlo simulation process for each contingent activity a_i: b.1) represents the search process for the minimum sigma multiplier $z_i^{u,m}$ in simulation m; b.2) represents the point in which $z_i^{u,m}$ is found (and consequently the subset of properties that are violated $P^{i,u} \subseteq P$ is also found) in simulation scenario/run m.

Finally, every tuple (a_i, z_i^u, Pi, u) is stored and later ordered by the value z_i^u (line 6). The ordered list of contingent activity is then returned, $Z^{u,ordered}$. Note that we not only rank the activities by the degree of brittleness, but we can also identify exactly which properties $P^{i,u}$ were sensitive to that z_i^u.

A special case of the analysis in Algorithm 9 is when the $Y = [0...0]$, i.e. Y^0 meaning that all other contingent activities a_j have duration set to be exactly the nominal case, $[\mu_j, \mu_j]$. In that case, we can determine the the maximum/upperbound value of each z_i^u, i.e. increased values in Y can only decrease our brittleness metric for each activity.

Figure 6.4 shows an example of ranking of activity brittleness based on z_i^u (in the Y^0 case) from one of the studied M2020 task networks with 20 rover activities, in which we consider three brittleness constraints $P = \{P_{DC}, P_{SOC^{min}}, P_{SOC^{handover}}\}$. Here we use $1/z_i^u$ values to make the brittleness representation more intuitive: the higher $1/z_i^u$, the more brittle the activity is. Note that the chart in Figure 6.4 can be mapped to maximum delays ($z_i^u \times \sigma_i$) providing actual timing information. Figure 6.5 shows the same ranking example but now using the probability mass $1 - p_i^u$, relating to a risk/probability of the activity's duration falling outside the bounds defined by using z_i. Herein, $1 - p_i^u$ is capturing the probability of the activity duration taking longer then the maximum delay. Moreover, Figure 6.6 illustrates which properties are violated at z_i^u - in this example Dynamic Controllability is the dominant property.

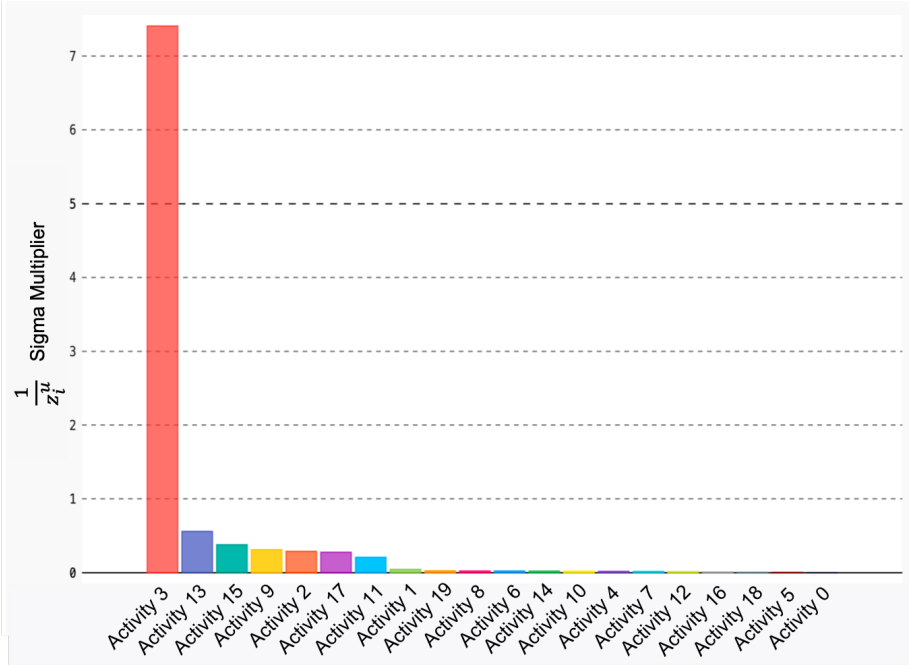


Figure 6.4: Ranking of brittle activities from one of the M2020 task network studied in this work (*TN 8*, see experiments). Each activity a_i has a corresponding sigma multiplier z_i^u that represents the temporal deviation bounds in which the task network violates the conjunction of properties in P. The vector of sigma multipliers Y^0 (special case) is applied to all other contingent activities. The chart shows value of $1/z_i^u$ for each activity.

These ranking charts can inform designers or planners what activities are most brittle, which property are dominantly more brittle, and which of the activities might need further inspection if they do not meet a desirable brittleness level.

It is worth noting that varying the values in the vector of sigma multiplier Y can provide useful information about interrelated contingent activities. For example, If we iterate $y_i \in Y$ from 0 to z_i^u and apply our algorithm 9, we can evaluate the gradual impact of other activities uncertainty to each activity a_i. If the resulting values of z_i^u do not change with Y variation, then it shows that activity a_i is independent of other activities' duration and uncertainty (or their effect is insignificant). If z_i^u value do changes with varying Y, then a_i is interrelated to other activities uncertainty.

In what follows, we describe a complementary analysis that explores the domain space of Y to better analyze interrelation and brittleness in more diverse *ceteris paribus scenarios.*

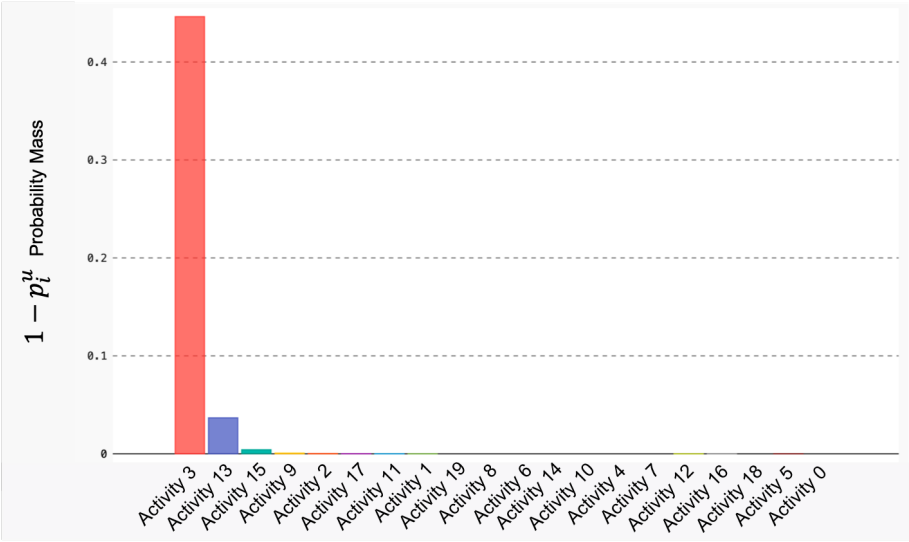


Figure 6.5: Ranking of brittle activities from one of the M2020 task network studied in this work (*TN 8*, see experiments). Each activity a_i has a corresponding probability p_i^u that represents the temporal deviation bounds in which the task network violates the conjunction of properties in P. The vector of sigma multipliers Y^0 (special case) is applied to all other contingent activities. The chart shows value of $1 - p_i^u$ for each activity relating it to a risk/probability of the duration falling outside the bounds.

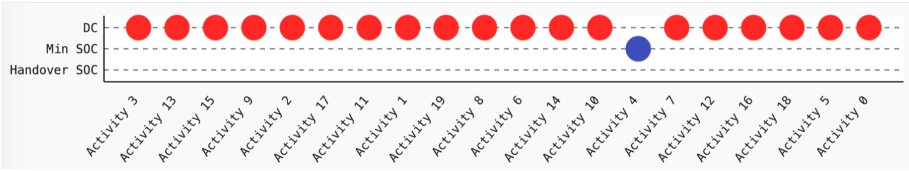


Figure 6.6: Properties violated for each activity at deviation z_i^u from one of the M2020 task network studied in this work (*TN 8*, see experiments). Dots mark which corresponding property has been violated at z_i^u. The chart provides complementary information to the ranking shown in Figures 6.4 and 6.5 - the ranking order of the activities is kept the same.

6.4 Brittleness Analysis

6.4.1 Monte Carlo Simulation Analysis

We now explore the impact of an activity's brittleness level given different random duration bounds applied to all others contingent activities in the task network. Specifically, we randomly select sigma multipliers $y_j \in Y$ for each contingent activity a_j when analyzing activity a_i ($j \neq i$) to create random *ce-*

Algorithm 11 Ceteris-Paribus-Brittleness-Analysis-with-Monte-Carlo-Simulation

Input:

A Task Network TN

A set of properties P

A number M of Monte Carlo simulations per contingent activity

A list/vector W used to bound the sigma multipliers when creating a random *ceteris paribus scenario*

Output:

A mapping between contingent activities and a list of values of z_i^u for each simulation

1: $Z^{u,MC} \leftarrow \{\}$ ▷ initializes the mapping
2: **for** $a_i \in A_C$ **do**
3: $Z_i^u \leftarrow \{\}$ ▷ initializes the list of σ multipliers for a_i
4: **for** m from 1 to M **do** ▷ Perform Monte Carlo simulations
5: $Y^m \leftarrow RandomSelectYMultipliers(a_i, W, TN, P)$
6: $Z_i^{u,m} \leftarrow ComputeActivitySigmaMultiplier(a_i, Y^m, TN, P)$
7: $Z_i^u.append(Z_i^{u,m})$
8: $Z^{u,MC}.append(Z_i^u)$
9: **return** $Z^{u,MC}$

teris paribus scenarios and explore the domain space of the uncertainty level in the whole task network. We explore the range $0 \leq y_j \leq W_j$ for each element in Y where $W_j \in W$ is an arbitrary upper bound on the sigma multiplier assigned to define activity a_j's lower and upper bounds on the contingent duration. A straightforward value for W_j would be z_j^u (which requires first running the aforementioned *ceteris paribus* algorithm with the special case Y^0. If normal distributions are associated to the contingent activities in the task network, then one could use values such as $W_j = 3$, covering 99.7% of the possible values of duration for activity a_j, or $W_j = 5$ covering 99.9% of the cases, or a less conservative value $W_j = 1$ covering 68.2% of cases.

We propose a Monte Carlo approach inspired from SMC techniques to augment our analysis and explore different *ceteris paribus scenarios*, represented in Algorithm 11. For each activity a_i, we run M Monte Carlo simulations in which we first randomly select a set of sigma multipliers Y^m (line 5), one per contingent activity a_j ($j \neq i$) in simulation m, i.e. y_j^m where $0 \leq y_j^m \leq W_j$. As mentioned before, one could run Algorithm 9 with Y^0 and use all or some of z_i^u to set the list/vector W (a list of arbitrary upper bounds on the sigma multipliers, one per contingent activity). In order to select Y^m, function $RandomSelectYMultipliers$ creates an new TN by i) setting

a_i duration bounds to $[\mu_i, \mu_i]$ (i.e., no deviation to the target activity duration), and ii) applying/sampling a random y_j to each activity a_j's duration bounds. We select Y^m such that the resulting new TN does not violates any of the properties in P. Here we use a generate-and-test approach to create Y^m.

Given a scenario established by Y^m, the algorithm then computes the specific $z_i^{u,m}$ (line 6) for that scenario and simulation m. Function $compute - ActivitySigmaMultiplier$ (Algorithm 10) is the same as the one used in Algorithm 9.

6.3 (b) illustrates that process. The resulting set of sigma multipliers Z_i^u represents activity a_i's variation of the brittleness level due to varying degrees of uncertainty across the task network. Those sets are then aggregated (line 9) to provide the result of the algorithm $Z^{u,MC}$: a mapping between contingent activities and the distribution of their corresponding sigma multipliers.

The data points in $Z^{u,MC}$ can be plotted to a box-whisker chart, Figure 6.7, illustrates how brittle each activity is with respect to the variation of the uncertainty level of others activities from the same M2020 task networks analyzed in Figure 6.4. Herein, we can more clearly visualize how dependent (large variations) or independent (no or insignificant variations) the activities are to others.

6.4.2 Mapping Brittleness to Task Network Structure

We implemented a tool for representing the outputs of the proposed analysis alongside a graphical representation of the temporal network (such as the one shown in Figure 6.1). The objective is to help a designer to map brittleness to elements in the network structure in order to identify brittleness sources. In addition to the temporal constraints (edges), time points representing the start and end times of activities, and means and sigma, we provide the following information for each activity a_i in the task network:

1. **Brittleness rank**: activity's rank in the brittleness scale, where 1 is the most brittle. Here we use color code to visually represent activity from most to least brittle;

2. **Sigma multiplier**: the z_i^u from Algorithm 9 (with Y^0) and the mean, min, max values of z_i^u from Algorithm 11.

3. **User-specified properties violated**: the list of properties $P_k^u \in P$ from Algorithm 9 (with Y^0).

4. **Delay^u**: refers to $\sigma_i \times z_i^u$ representing the max delay of the activity (using z_i^u from Algorithm 9 with Y^0) in which the network violates properties in P;

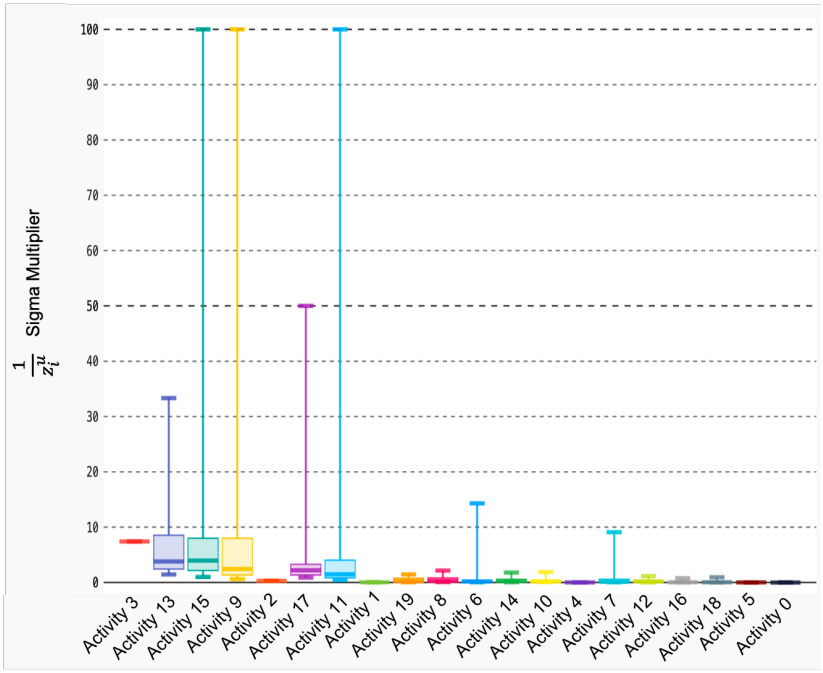


Figure 6.7: Box-whisker chart of the variation of sigma multipliers Z_i^u of each contingent activity a_i in the Monte Carlo approach from one of the M2020 task network studied in this work (*TN 8*, see experiments). The higher the mean of $1/z_i^u$, the more brittle the activity is.

5. **Specified execution time window**: the time window specified in the input TN based on the earliest start time and cutoff time (latest end time). This is to contrast with the inferred time window below.

6. **Inferred execution time window** (Tw_{Exec}^{Infer}): here we run Floyd-Warshall's all-pairs shortest path algorithm to infer the actual available time window for execution due to other temporal constraints. This relates to the temporal flexibility of the activity.

7. **Inferred start time window**: the length of time window in which we can start the activity. This item also relates to the temporal flexibility of the activity start $flex_N(t_{i\_start})$

8. **Dependencies**: according to the network's structure, we present how many activities need to be scheduled/executed *before* activity a_i, as well as how many activities need to be scheduled/executed *after* the activity. This item provides some notion on where this activity is on the dependency chain, if any.

By visual inspection or structure analysis, designers and planners can study the causes of brittleness given the parameters above while tracing the associated temporal constraints on the task network structure. In the applications section we describe how the above added information support identifying the causes on brittleness and the common reasons to why an activity is highly brittle.

6.4.3 Addressing Undesirable Activities' Brittleness Levels

With the afore-proposed activity-centric, property-based brittleness measure and analysis at hand, we go back to the desideratum of determining, in an automated fashion, possible ways to change or relax some of the temporal constraints in the task network to bring specific activities' brittleness to a user-acceptable level (i.e. question (c) listed in Section 6.1). In this section, we propose an advisory system that suggests temporal constraint relaxations (e.g., decrease the earliest start time of activity a_{drive} by 300 time unit) that meets user-specified brittleness measure thresholds for a set of target activities in the *ceteris paribus* analysis - usually those activities that were found to be problematically brittle. If the suggestion is not satisfactory from the user/planner's perspective, the system would generate the next best relaxation solution.

We focus on temporal constraint relaxation suggestions only. Moreover, the system described in what follows handles only one type of property in P: the temporal dynamic controllability P_{DC} (strong controllability is also supported) [MMV01; Mor14]. Handling multiple types of constraint relaxation and properties, specifically resources, is a quite interesting research avenue which is left for future work.

Our task network relaxation problem addressed by the proposed advisory system refers to $RP = \langle TN, A_{Brittle}, RA^{TC}, BM^{TH}, bm^{default} \rangle$, where:

- $TN = \langle A, H \rangle$ is our task network, where $A = A_C \cup A_R$. Note that here we ignore resource constraints entirely;
- $A_{Brittle}$ is the set of target activities, $A_{Brittle} \subseteq A_C$, that are deemed too brittle from the user/planner's perspective, i.e. its brittle measure does not meet an acceptable value. These are the activities that one wants to make sure they are in the brittleness measure acceptable range.
- RA^{TC} is a list of relaxable activities' temporal constraints such as earliest and latest start times or latest end times. ra_i^{TC} refers to the set of activity a_i's temporal constraints that can be relaxed.
- BM^{TH} is a mapping between target activities $a_i \in A_{Brittle}$ to a brittleness measure threshold bm_i^{TH}. Herein bm_i^{TH} refers to either an ac-

Algorithm 12 Brittleness Level Guided Task Network Relaxation

Input:
A task network temporal relaxation problem
$RP = \langle TN, A_{Brittle}, RA^{TC}, BM^{TH}, bm^{TH}_{default} \rangle$

Output:
A set of relaxations, R^{TC}, that maps a subset of RA^{TC} to increments or decrements of time

▷ initializes solution
1: $R^{TC} \leftarrow \{\}$
▷ translates task network to its corresponding STNU
2: $TN_{STNU} \leftarrow generateSTNU(TN, A_{Brittle}, BM^{TH}, bm^{TH}_{default})$
▷ computes relaxation
3: $search\_pointer, R^{TC} \leftarrow ComputesRelaxationSolution(TN_{STNU}, RA^{TC})$
▷ Provides next best solution if required
4: **while** User needs next solution and one exists **do**
5: $R^{TC} \leftarrow search\_pointer.nextSolution()$
6: **return** R^{TC}

ceptable minimum sigma multiplier z^{TH}_i or an acceptable confidence level p^{TH}_i (probability mass between lower and upper bounds around the mean) in case of normal distribution.

- $bm^{TH}_{default}$ is the default brittleness measure threshold for all non-target activities.

The output of our system is a set of relaxations, R^{TC}, that maps a subset of RA^{TC} to increments or decrements of time. In the aforementioned example on activity a_{drive}, the solution would look like $R^{TC} = \{ra^{TC}_{drive,est} \rightarrow -300\}$ (subscript $drive, est$ refers to earliest start time constraint of activity a_{drive}) which means that $drive, est$ should be reduced by 300.

Algorithm 12, *Brittleness Level Guided Task Network Relaxation*, provides the pseudo code for the proposed advisory system. The first main step corresponds to the *translation* (line 2) of the input task network TN to a STNU, TN_{STNU} following the basic encoding described in the Background section (STN and STNU) where the following is applied:

1. each target activity a_i's duration in $A_{Brittle}$ is set to be a contingent edge with bounds $[\mu_i - bm^{TH}_i \times \sigma_i, \mu_i + bm^{TH}_i \times \sigma_i]$ in case bm^{TH}_i is a sigma multiplier. If a confidence level p^{TH}_i is provided instead of a sigma multiplier, then the bounds are set to $ci(p^{TH}_i, \mu_i, \sigma_i)$ where ci computes and returns the *confidence interval* of the duration distribution.

2. each non-target activity a_j's duration in A_C ($a_j \notin A_{Brittle}$) is set to be a contingent edge with bounds $[\mu_j - bm^{TH}_{default} \times \sigma_j, \mu_j + bm^{TH}_{default} \times \sigma_j]$ in case $bm_{default}$ is a sigma multiplier. The same aforementioned case applies if $bm_{default}$ is a confidence level. Note that if $bm_{default} = 0$ we have all the non-target activities set to the nominal (mean) duration.

3. The duration of activities in A_R are left unchanged.

The resulting STNU has to be dynamically uncontrollable otherwise no relaxation is needed, i.e., the dynamic controllability constraint is already satisfied with the given thresholds. Given the uncontrollable TN_{STNU}, we then *compute a solution for the relaxation problem under the dynamic controllabitity constraint* (line 3), i.e. a solution, if one exists, is guaranteed to be dynamically controllable. In this work we use the Conflict-Directed Relaxation with Uncertainty (CDRU) algorithm [Cui+15] to solve a linear program optimization problem over the uncontrollable STNU. CDRU takes the uncontrollable STNU and finds a least-cost relaxation of the temporal constraints in RA^{TC} that makes it dynamically controllable. Herein, relaxing refers to tightening or loosening temporal constraints edges, which is driven by relaxation cost functions associated with the change to the edges/links. Cost functions associated to temporal constraint edges of the STNU can be provided by users/planners to represent preferences the relaxable set of activity temporal constraints RA^{TC}. In this chapter we do not explore that option and use instead a uniform cost since the focus here is on the satisfability of the dynamic controllability constraint bc_{DC} in the relaxation problem RP. For more details on the relaxation approach see [Cui+15].

Finally, Algorithm 12 produces an interactive process of generating the next best solution/relaxation (lines 4-5). If the user requests a new solution the algorithm will provide the next best option if one exists; otherwise it terminates. This is possible thanks to the pointer $search\_pointer$ that stores the CDRU's search state and learned conflicts to keep computing solutions. We believe this provides a powerful tool for designers and planners to explore different options of temporal constraints relaxations and therefore ways to bring the task network to the acceptable brittleness level.

It is clear that the CRDU algorithm is key to the relaxation process proposed in this work. Our algorithm can be seen as a wrapper function of CRDU that puts it in the context of addressing undesirable brittleness levels in task networks.

6.4.4 Statistical Model Checking for Task Networks

Until this point, we have assumed that the function $Check(TN, P_k)$, which returns 1 if the property P_k holds in the task network TN, always exists as a

black-box.

As mentioned in Section 6.3.3, exhaustive methods to compute $Check(TN, P_k)$ exist and have already been explored in the litterature for some classes of TNs and some types of properties, such as strong and dynamic controllability [MMV01; Mor14; Cui+15].

In many contexts however, no algorithm to compute the function $Check(TN, P_k)$ is available, either because it has not been invented yet or because the complexity of the exhaustive methods is prohibitive. In particular, the inherent stochastic nature of PTSNs calls for an approach that takes into account and rely on their probabilistic features.

We now present how SMC can offer a general solution to the problem of designing an algorithm for the function $Check(TN, P_k)$ which is applicable for any TN and any property, with the only assumption that we must be able to compute the simulation function $\Phi(TN, P, \xi)$.

From the point of view of SMC, this is a special case of probability estimation and/or hypothesis testing. The goal is to verify whether a task network TN, under a scenario ξ, has a probability of 1 to verify the property P.

Any estimation algorithm of SMC could be exploited to solve this problem (including those of Chapter 3): first estimate the probability that P holds for TN under a random possible scenario ξ, then check whether that estimation is equal to 1. For the sake of simplicity, one could simply use a classic Monte Carlo algorithm.

If the property P isn't critical and a non-zero rate of false positives can be considered for the algorithm computing $Check(TN, P_k)$, then a threshold-based approach such as the one described in Section 5.2 is perfectly appropriate.

However, in the context of the TN brittleness analysis, it is common that the algorithm computing $Check(TN, P_k)$ must truly verify whether the property *always* holds. In that case, we can be much more straighforward: simply generate new scenarios ξ until $\Phi(TN, P, \xi)$ returns 0. If $\Phi(TN, P, \xi)$ still hasn't returned 0 after a while, then return 1.

The only question that remains is therefore the following:

Problem 6.1. *With a confidence level $\delta > 0$, how large should a sample of realizations of a random variable X following a potentially non-trivial Bernoulli distribution to be allowed to claim that the random variable is actually constant?*

Note that Problem 6.1 becomes theoretically trivial (in the context of this chapter) if the total number of different scenarios is finite and known. Thus, we will focus on the case where that number is unknown (and assumed potentially infinite).

The Rule of Three

Albeit we note that more elaborate methods to solve that problem do exist (such as sequential testing or the adaptive stopping algorithms of Chapter 3), we must insist that the goal of this work was to provide a solution to human designers of task networks (*i.e.* people potentially unfamiliar with the field of SMC) which is both easy to understand and to implement.

An elementary solution to Problem 6.1 comes from a result that is considered folklore in statistics: the rule of three.

Theorem 6.1 (The Rule of Three)**.** *If a sample of size $n > 0$ of realizations of a random variable X potentially following a non-trivial Bernoulli distribution contains only success, a 95% confidence interval for the probability of success is $[1 - 3/n, 1]$.*
More generally, for a confidence level $\delta > 0$, a $1 - \delta$ confidence interval is given by $[1 + \ln(\delta)/n, 1]$.

Proof. For a random variable $X \sim \mathcal{B}(p)$, under the assumption that $0 < p < 1$, to guarantee that the probability that a sample of size n contains only failures is smaller or equal to δ , p needs to be small enough so that $(1-p)^n$ is smaller or equal to δ:

$$(1 - p)^n \leq \delta$$

Since ln is a non-decreasing function, this is equivalent to:

$$n \ln(1 - p) \leq \ln(\delta)$$

Around $x = 0$, the Taylor polynomial of degree 1 for the function $f(x) = \ln(1 - x)$ is $T(x) = -x$. In other words: $\ln(1 - p) \simeq -p$.

$$-np \leq \ln(\delta)$$

$$p \leq \frac{-\ln(\delta)}{n}$$

In conclusion, a $1 - \delta$ confidence interval for p is $[0, \frac{-\ln(\delta)}{n}]$. By symmetry, $[1 + \ln(\delta)/n, 1]$ is therefore a $1 - \delta$ confidence interval when considering successes instead of failures.

(If $\delta = 0.05$, then $-\ln(\delta) \simeq 2.99573 \simeq 3$. Hence the name of the theorem.)

□

With the rule of three, we can easily derive a formula for the number of scenarios we need to simulate without encountering a single case for which the property P is not verified before returning 1 for $Check(TN, P_k)$, in function of the user- or designer-specified values for the precision $\epsilon > 0$ and the confidence level δ of the SMC method. Indeed, if we want the lower bound

Algorithm 13 SMC algorithm to compute (estimate) $Check(TN, P_k)$

Input:
A Task Network TN
A property P_k
A precision ϵ and a confidence level δ

Output:
1 if there is not enough statistical evidence to reject the hypothesis that P_k is verified for TN, 0 otherwise

```
1: n ← ⌈−ln(δ)/ϵ⌉                                  ▷ Rule of three
2: S ← a set of n random possible scenarios for TN  ▷ Generate a set of random scenarios
3: for ξ ∈ S do                                    ▷ Check if there is at least one bad scenario
4:     if Φ(TN, P_k, ξ) == 0 then
5:         return 0
6: return 1
```

of the $1 - \delta$ confidence interval to be no more distant than ϵ from 1, we must have:

$$1 + \frac{\ln(\delta)}{n} \geq 1 - \epsilon$$
$$\ln(\delta)/n \geq -\epsilon$$
$$-\frac{\ln(\delta)}{\epsilon} \leq n$$

We note that this formula can be seen as a degenerate case of the Chernoff bound. Moreover, that formula is based on a (very rough) approximation (see proof of Theorem 6.1).

Algorithm 13 shows pseudocode for an implementation of a SMC algorithm which computes (estimates) $Check(TN, P_k)$. In addition to the flaws of the formula this algorithm is built upon, it suffers from all the drawbacks of the SMC methods which need to estimate their sample size *a priori* as discussed in Chapter 3. We suggest that adaptive stopping algorithms could provide a great alternative for the estimation of $Check(TN, P_k)$ in the context of temporal networks, but this is left for future work.

6.5 Application: Mars 2020 task networks

In this study, we analyze a set of task networks from Mars 2020 sol types and run the proposed property-based brittleness analysis specifically through the *ceteris paribus* analysis and the Monte Carlo approach. By using the analysis, mapping and visual inspection tools, we draw observations on the typical

characteristics found with respect to variations on activity's uncertain duration and the three target user-specified properties: dynamic controllability, the global minimal SOC, and handover minimum SOC. We then use these observations to guide a few cases in which we apply our proposed temporal relaxation system to reduce the brittleness level of certain problematic activities.

6.5.1 Setup

We run the property-based brittleness analysis using a set of nine M2020 sol type task networks. These networks are representative of what has been investigated and applied to develop an onboard scheduler for the M2020 rover [Chi+18]. Sol types generally contain between 20 and 50 activities, such as driving, conducting remote science, and taking images. Table 6.1 shows some of the main properties of each task network used in this work, including the number of activities and dependencies (the total number of activities that are listed in all the activity's dependency set D_i as described in Section 6.2.1), as well as the resulting number of nodes and edges when translating it to an STNU for dynamic controllability property check ($Check(TN, P_{DC})$).

At the time of this study, the M2020 rover had only been active for a few months. Therefore, accurate models of activity duration variance were not yet available. For the purpose of this study and to illustrate the benefits of the analysis, we use an estimate of the nominal durations of activities as their mean value (based on conservative estimates from scientists) and select a sigma for each activity randomly as a fraction of the mean, i.e. $\sigma_i = r_i \times \mu_i$ where $0 < r_i < 1$.

With respect to energy constraints, we use estimate values for activity's energy consumption rates (e_i), as well as estimate initial state of charge (SOC^{init}), global minimum and maximum state of charge (SOC^{min} and SOC^{max} respectively), and handover minimum state of charge ($SOC^{handover}$) based on target operation requirements. These estimate values for energy constraints are also based on conservative estimates from scientists and engineers.

In this work, we focus on the following set of user-specified properties P:

1. **Dynamic Controllability Property** (P_{DC}). For STNU dynamic controllability (also referred here as P_1) checks, $Check(TN, P_1)$ we used the open-source implementation derived from [Cui+15].

2. **Global Minimum State of Charge Property** ($P_{SOC^{min}}$). For this property (also referred as P_2) we consider each activity's energy consumption rate and its duration to compute the energy profile in the plan execution. In each analysis check, we use the duration variation upperbounds (delays) only to cover the worst case on energy consump-

tion. We also consider the rover's sleep mode [Agr+19] during which the rover's energy level increases during a sol; no activity is executed while in sleep mode. The set of sleep mode occurrences are not explicit in the task network, but are scheduled by the onboard scheduler depending on the energy demand for execution. We run the scheduler with all activities' nominal durations to determine the set of scheduled sleep operations, $sl_j \in Sl$, in the nominal case. We use that set of sleep mode operations (and their respective energy production rates e_j^{sl}) as a reference to compute the energy profile in all the deviations of activity duration. This estimate already takes into account shunting, in which the rover is put into an awake mode if the energy level reaches the Maximum State of Charge (SOC^{max}) to prevent long term battery performance degradation. Starting from the initial state of charge value SOC, we simulate the energy consumption after each activity execution (i.e. $dur(a_i) \times e_i$) and energy production after a sleep operations (i.e. $dur(sl_j) \times e_j^{sl}$, capped at SOC^{max}) – following the order of activities and sleep operations provided in the nominal plan – to compute the energy level at the start and end time point of each event (activity or sleep) in the plan. We then check if the energy level at the end of each activity a_i is greater than the global minimum state of charge SOC^{min}. Therefore, our $Check(TN, P_2)$ function boils down to checking if the following holds for each activity a_i:
$SOC^{init} - \sum_{n=0}^{i} dur(a_n) \times e_n + \sum_{n=0}^{j} dur(sl_n) \times e_n^{sl} \geq SOC^{min}$, where the first sum computes the energy consumption from start of the plan up to the end time point of activity a_i while the second sum computes the energy production of all sleep operations that were scheduled before a_i in the nominal plan.

Disclaimer: Varying activity durations in our analysis could potentially affect the timing of the sleep and shunting operations from the nominal case. However, we are ignoring that effect here and using the same exact set of sleep operations from the nominal plan in all the deviation scenarios of the analysis for a particular task network. A more precise computation of sleep and shunting modes is left for future work since the focus of this work is on the constraint agnostic analysis process itself rather than a precise computation of energy profile and constraint violation.

3. **Handover Minimum State of Charge Property** ($P_{SOC^{handover}}$). For this property (also referred as P_3) we consider each activity's energy consumption rate and its duration to compute the *total energy usage* in the plan execution. Similarly to the above property, we use the duration variation upperbounds (delays) only and the set of *sleep* operations Sl

Task Network	# Act	# Dep	# Nodes	# Edges	$z_{i,min}^{u,Y^0}$	$z_{i,max}^{u,Y^0}$	$1 - p_i^{u,Y^0}$	# P_1 viol.	# P_2 viol.	# P_3 viol.
TN 0	32	26	67	188	0.095	1336.77	0.46	29	3	0
TN 1	40	39	83	233	0.045	1117.31	0.48	37	3	0
TN 2	28	26	59	168	0.060	389.71	0.47	25	3	0
TN 3	18	21	39	113	0.030	3076.92	0.49	16	2	0
TN 4	42	42	87	251	0.150	681.20	0.44	39	3	0
TN 5	42	42	87	251	0.140	595.60	0.44	39	3	0
TN 6	35	42	73	215	0.030	5241.09	0.49	32	2	1
TN 7	46	52	95	282	0.095	980.40	0.46	43	3	0
TN 8	20	18	43	120	0.135	896.86	0.44	19	1	0

Table 6.1: Studied M2020 sol types task networks with their respective number of activities (# Act), total number of direct activity dependencies (# Dep), and the number of nodes and edges in the corresponding STNU. It also includes the metric for the most brittle activity ($z_{i,min}^{u,Y^0}$), the metric for the least brittle activity ($z_{i,max}^{u,Y^0}$), the probability of falling beyond the z_i^{u,Y^0} values in the activity delay case only (which is equal to $1 - p_i^{u,Y^0}$ for the most brittle activity), the number of activities that violated the controllability property (P_1), the global minimum state of charge property (P_2), and the handover minimum state of charge property (P_3), all from the *ceteris paribus* analysis.

from the nominal plan to compute energy gain/production. However, as opposed to looking at the energy profile and at the end time point of each activity, herein we instead look at the end of the plan execution only. The energy available at the end of the plan execution will be used as a reference for the energy that will be available in the next sol (handover energy). Violating that property will impact the rover's productivity and schedule for the following sol type. In this case we can focus on the total consumption and total production of energy. Therefore, our $Check(TN, P_3)$ function boils down to checking if the following holds:
$SOC^{init} - \sum_{a_i \in A} dur(a_i) \times e_i + \sum_{sl_j \in Sl} dur(sl_j) \times e_j^{sl} \geq SOC^{handover}$.

Disclaimer: The same issue related to the effect on sleep and shunting modes holds in this constraint. We chose to ignore that effect and leave a more precise computation of sleep and shutting operations for future work.

For each task network (*TN 1* through *8*), we perform the following: 1) run Algorithm 9 in the special case Y^0 to identify the upper bound of z_i^u and plot the brittleness ranking; 2) run Algorithm 11 to analyze the brittleness metric value variation in the Monte Carlo approach, where we set 20 simulation per activity a_i (i.e., $M = 20$); and 3) we use the mapping tool to generate graphical representation (STNU-like) of the task network with the added layer from the brittleness analysis data. Finally, we select a representative task network from the analysis results to run the temporal relaxation Algorithm 12 and generate a temporal relaxation solution to improve temporal brittleness - here

we consider property P_1 only.

Given that the analysis is meant to be an offline process, runtime analysis of the algorithms is not the focus of our research. As a reference, Algorithm 9 usually runs in about 2 minutes in a Ubuntu 18.04 Virtual Machine, 12GB Memory, 3.1 GHz Intel Core i7. Algorithm 11 varies significantly with the number of Monte Carlo simulations and the selected method to chose the starting scenarios Y^m (line 5) that satisfy all user-specified properties. Algorithm 12 runtime is directly related to the CDRU algorithm and varies mainly according to the size of the network and the number of relaxable temporal constraints. For the example provided in this section, the relaxation runtime is about 1 second.

6.5.2 Results

6.5.2.1 Property-based Brittleness Analysis

Table 6.1 shows the brittleness metrics for each task network from the *ceteris paribus* analysis, specifically from Algorithm 9. The table also presents the minimum values of the target sigma multiplier $z_{i,min}^{u,Y^0}$, i.e., for the most brittle activity, and the maximum values of the target sigma multiplier $z_{i,max}^{u,Y^0}$, i.e., the least brittle activity value, to illustrate the large range of brittleness in the set.

These results show a brittle set of task networks, given the low sigma multipliers $z_{i,min}^{u,Y^0}$, that is: with a small variation of one key activity's duration the task network violates at least one of the user-specified properties.

The low sigma multipliers reflects on the high value of $1 - p_i^{u,Y^0}$, showing a high risk of property violation. Table 6.1 also shows that dynamic controllability (P_1) is the dominant property with a large margin compared to both the global and handover minimum state of charge properties (P_2 and P_3 respectively). Looking closely at that aspect we noticed that, in all task networks studied, P_2 and P_3 usually show up being violated for the activities ranked in the middle or in the end of the ranking; never in the top 7 most brittle activities.

Figure 6.8 (1-2) and Figure 6.9 (1-2) illustrate clearly which activities are most brittle in *TN 2* and *TN 3*, respectively. In *TN 2*, *Activity 9* is the most brittle, and largely more brittle than the other activities. In *TN 3*, *Activity 3* is largely more brittle than any other activity in the task network. Figure 6.10 (1-2) illustrates an interesting case in *TN 7*, in which we can identify the most brittle activity, *Activity 8*; however, several other activities are also significantly brittle. This is due to the fact that temporal constraints in *TN 7* are much tighter, with less room for temporal variation.

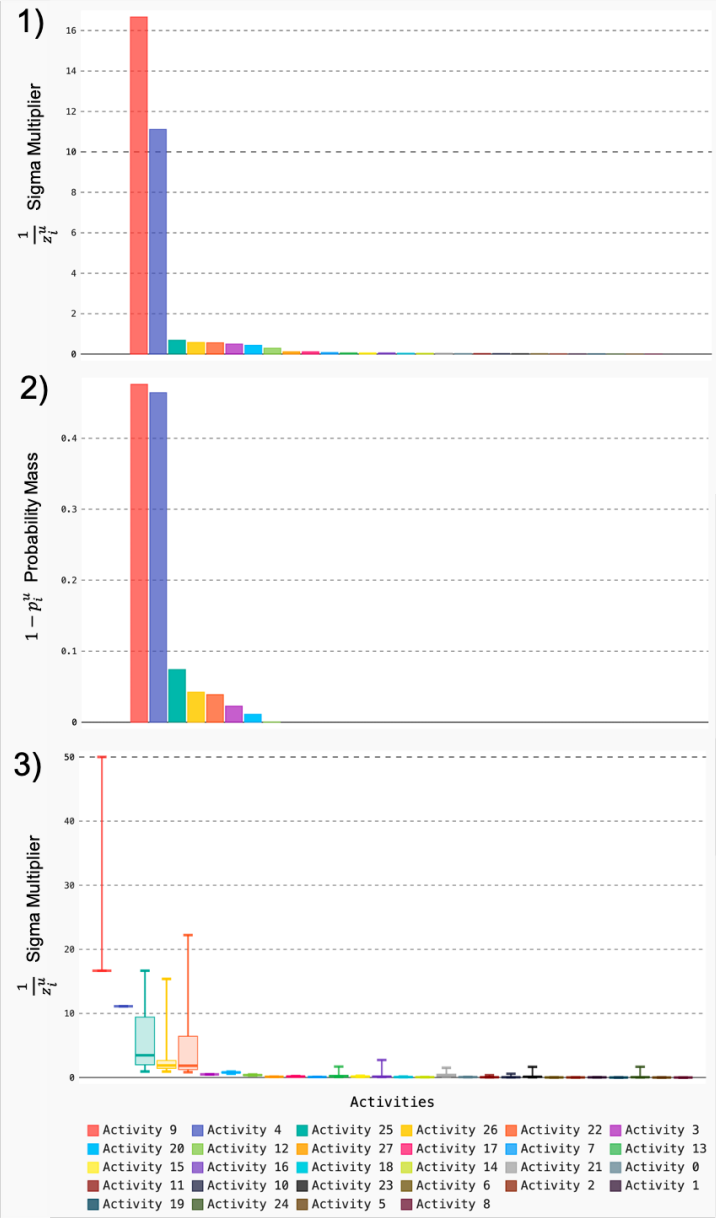


Figure 6.8: Activity brittleness ranking for TN 2: 1) Shows the brittleness ranking with respect to the sigma multiplier z_i^u where each activity a_i has a corresponding sigma multiplier z_i^u that represents the temporal deviation bounds from the mean beyond which the network violates one or more properties in P. We use Y^0 for all other contingent activities. The chart shows the value of $1/z_i^u$ for each activity; 2) Shows the same brittleness ranking with respect to the probability mass, $1 - p_i^u$, under the same assumptions; and 3) shows the same ranking based on the Monte Carlo approach with variation of sigma multipliers Z_i^u of each contingent activity a_i.

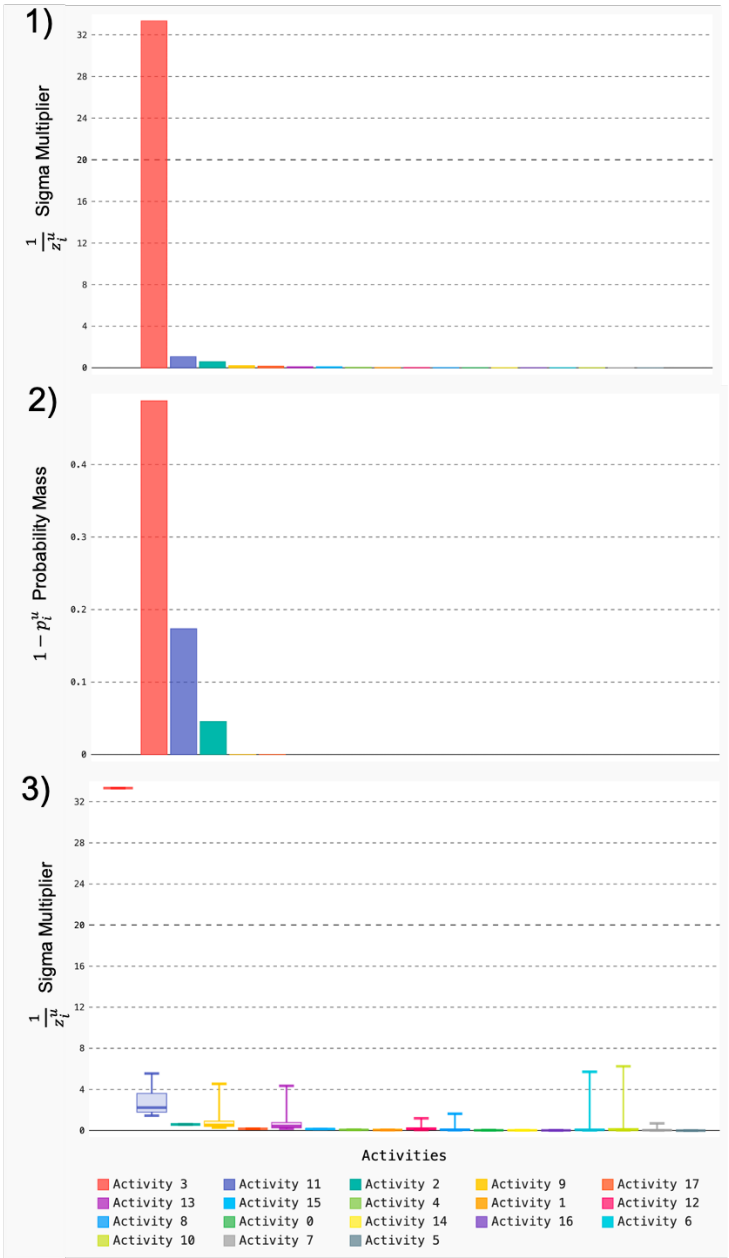


Figure 6.9: Activity brittleness ranking for TN 3: 1) Shows the brittleness ranking with respect to the sigma multiplier z_i^u where each activity a_i has a corresponding sigma multiplier z_i^u that represents the temporal deviation bounds from the mean beyond which the network violates one or more properties in P. We use Y^0 for all other contingent activities. The chart shows value of $1/z_i^u$ for each activity; 2) Shows the same brittleness ranking with respect to the probability mass, $1 - p_i^u$, under the same assumptions; and 3) shows the same ranking based on the Monte Carlo approach with variation of sigma multipliers Z_i^u of each contingent activity a_i.

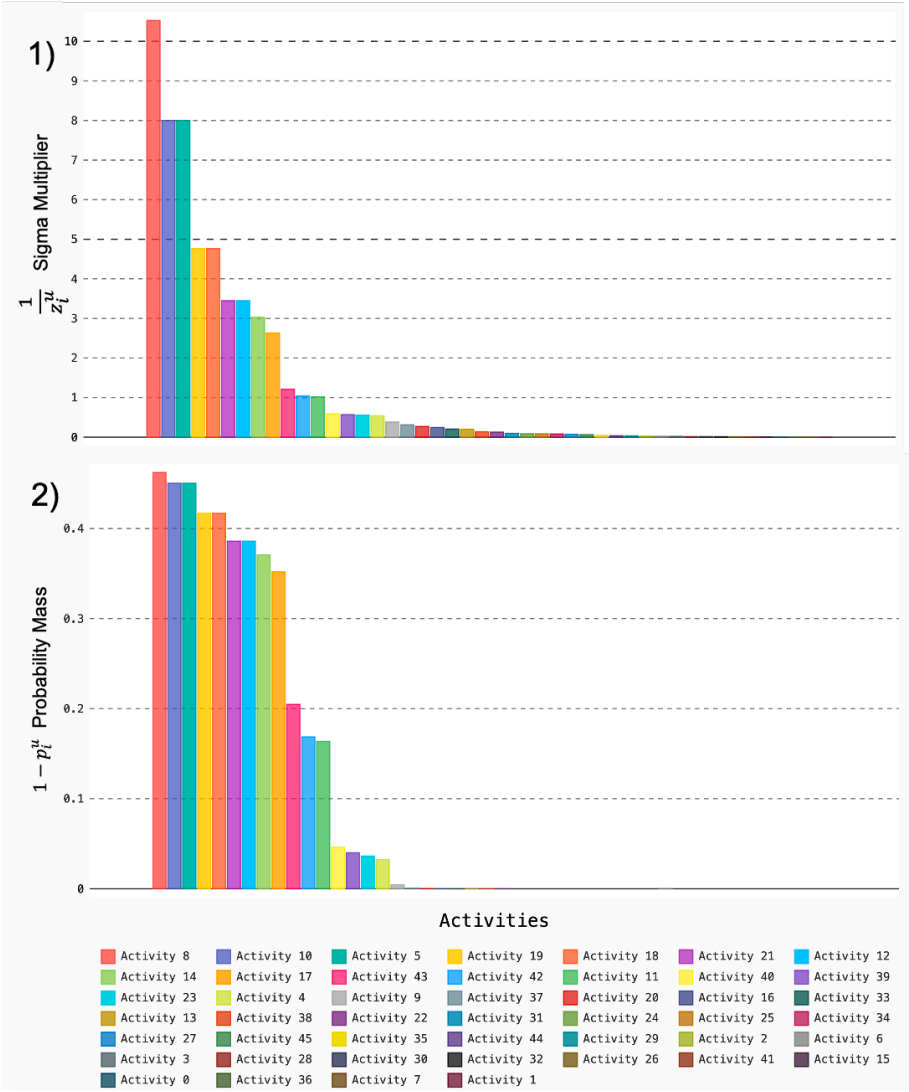


Figure 6.10: Activity brittleness ranking for TN 7: 1) Shows the brittleness ranking with respect to the sigma multiplier z_i^u; 2) Shows the same brittleness ranking with respect to the probability mass, $1 - p_i^u$.

The results from the Monte Carlo analysis and the mapping approach show two main sources of temporal brittleness in the studied set of M2020 task networks, which falls into two cases: ***Case 1*** represents brittleness due to a tight user-specified execution window; and ***Case 2*** represents brittleness due to a dependency chain.

Activities in Case 1 would usually have no (or insignificant) variation in the Monte Carlo analysis (e.g., activities 9 and 4 in Figure 6.8 (3) and activity 3 in Figure 6.9 (3)). That can also be inspected in the mapping tool where it would have no temporal dependency to other activities, but a tight window between the earliest start time and the latest end time. Case 2 manifests itself with variations to the brittleness value in the Monte Carlo approach and clear temporal/ordering dependencies in the graph representation (e.g., activities 25, 26 27 in Figure 6.8 (3)). In what follows we use task network *TN 2* as our representative example since it covers the two aforementioned cases.

The brittleness of *Activity 9* – a Long Drive activity – in Figure 6.11 (a) is primarily attributed to its tight execution window compared to its large nominal duration (*Case 1*). Each activity may have a cutoff time to ensure that the activity does not interfere with another activity. If the activity runs past its cutoff time then it is aborted and the activity fails to be scheduled. Even though the long drive has no dependencies or resource contention with any other activity, its nominal duration is 13044 sec (3 hr, 37 min) while the difference between its earliest allowed start time and its cutoff time is only slightly longer at 13563 sec (3 hr, 46 min). Then, assuming the activity starts as early as it can, if it runs longer than expected by even 9 minutes, it will violate its cutoff time and fail to be scheduled. Due to the lack of flexibility given by its execution window, even a small fluctuation from its nominal duration will result in an inconsistent schedule. Figure 6.8 (3) shows the long drive activity with no significant brittleness value variation in the Monte Carlo analysis, while Figure 6.11 (a) shows the portion of the STNU-like graph with the added analysis results and the key temporal constraints that represent the tight execution window case.

The most common example of an activity that is tied to a tight execution window in our study is one which has a lighting requirement from science which translates into a tight execution window. Another case would be cutoff times for activities: at cutoff time an activity would be aborted to prevent interference with a later, high priority activity. An example of this would be a science activity cutoff to ensure non interference with a communication activity – an activity which allows critical data to be sent to and from Earth and is constrained to start only at one particular time when an orbiter is overhead. We expect more illumination requirements from scientists in future sol types which would further constrain the execution windows, resulting in more cases such as this.

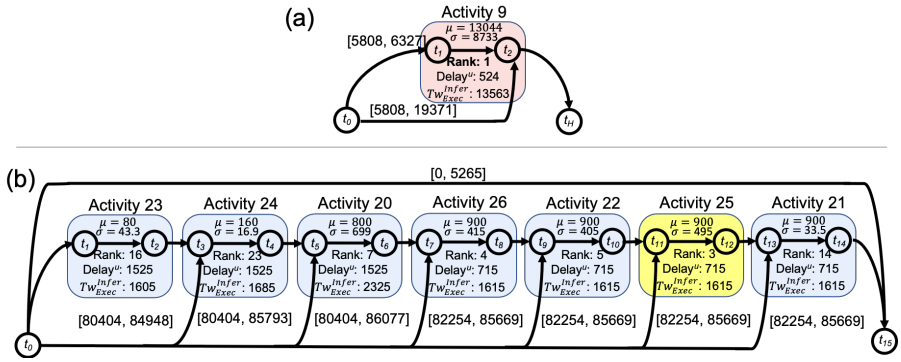


Figure 6.11: Brittleness due to (a) tight user-specified execution window and (b) to dependencies in *TN 2*. In all the activities shown here the dynamic controllability property is violated at deviation z_i^u.

We also examine 7 activities in the sol type *TN 2* (Figure 6.11 (b)) that cannot occur concurrently due to hardware resource contention (all use the Remote Sensing Mast (RSM)), which causes brittleness. The first three MastcamZ activities (*Activities 23, 24* and *20*) are imaging activities. These activities have execution windows that overlap with those of the next four activities, which all use the Navcam (cameras) in addition to the RSM. Of these next four activities, the first two create atmospheric monitoring movies (*Activities 26* and *22*) while the second two (*Activities 25* and *21*) generate movies of dust devil activity. These four activities have the same execution window and must all end by the same time and also share part of their execution windows with the previous three activities. Because all of these activities use the same hardware and have highly overlapping execution windows, they must be placed sequentially. As a result, they become fairly constrained in where they can be scheduled, making the risk of failure if these activities run long quite high (*Case 2*).

Although the four activities have the same nominal duration, *Activity 25* has a larger sigma (uncertainty) than the others and, therefore, is the most brittle of the four. *Activity 21* on the other hand has less uncertainty and is lower in the ranks, but the variations in the Monte Carlo simulations are as significant as for the other three activities in that chain.

Understanding which activities are more brittle, why and with respect to what property has important use cases. For example, such an analysis can assist scientists in pinpointing activities that have a high risk of failure and help them understand which constraints might need to be reassessed, whether they be temporal constraints, dependencies or resources. It could also be the case that certain activity duration estimates and/or standard deviations have to be revisited or redesigned (e.g., to reduce uncertainty, sigma).

If a more brittle activity was not expected to have high risk of failure, then such an analysis can help scientists understand how to reconstruct the plan so that that it is not the case. It could for instance be the case that energy resources have to be looked at closely, from the initial energy available at the beginning of the sol type, to certain activities' energy consumption, minimum thresholds and sleep mode requirements. For example, one could use the proposed analysis to perform a tradeoff study on brittleness and the amount of energy available in the beginning of the sol type (SOC^{init}). For example, P_2 starts to be the dominant property violated when SOC^{init} is about 89.5% of the original value in task network *TN 2*. Those types of tradeoffs could inform mission design and/or mission planning for reduced undesirable brittleness levels and, therefore, increased mission execution robustness.

6.5.2.2 Temporal Relaxation to Reduce Brittleness

In order to illustrate the proposed temporal relaxation capability and advisory system, we use task network *TN 2* which contains the two most common brittleness cases (as shown in Figure 6.8 (1-3) and Figure 6.11). The property-based brittleness analysis highlights two highly brittle activities under Case 1: Activities 9 and 4. It also highlights three activities in Case 2 that present large variability in the Monte Carlo analysis: Activities 25, 26 and 22 in that order (see Figure 6.8 (3)). Let us focus on these five most brittle activities in the task network and on the dynamic controllability property P_{DC}.

If we analyze Figure 6.8 (2) we can extract the probability mass p_i^u that refers to a symmetric bound around the mean (confidence interval). Note that Figure 6.8 (2) is using a probability mass (we will denote it by p_i^{u*}) in which the lower bound on the activity duration for computing the probability mass is zero, and therefore asymmetric, due to the fact that our M2020 use case is only brittle with respect to delays (activities ending earlier has no effect on brittleness). Based on the bar chart, we have $p_9^{u*} = 0.52$, $p_4^{u*} = 0.53$, $p_{25}^{u*} = 0.92$, $p_{26}^{u*} = 0.95$ and $p_{22}^{u*} = 0.96$. That corresponds to the following symmetric p_i^u (confidence level) values: $p_9^u = 0.04$, $p_4^u = 0.06$, $p_{25}^u = 0.84$, $p_{26}^u = 0.90$ and $p_{22}^u = 0.92$. We regard those values as our *undesirable levels of brittleness*, and they are the starting point of our relaxation approach.

We then set the following task network temporal relaxation problem *RN* as the input to the *Brittleness Level Guided Task Network Relaxation* Algorithm:

- TN is our task network example, $TN2$.
- $A_{Brittle} = \{a_9, a_4, a_{25}, a_{26}, a_{22}\}$ is the set of five aforementioned most brittle activities in $TN2$ as informed by the property-based brittleness analysis.

- RA^{TC} is the set the earliest and latest start time of *all activities* in $TN2$, including the five most brittle activities. We could have selected a subset of activities but for the sake of simplicity we chose all of them.
- $BM^{T}H$ is the set to be the following target brittleness measure threshold, i.e. the target confidence levels for the five target activities: $bm_{9}^{TH} = 0.50$, $bm_{4}^{TH} = 0.50$, $bm_{25}^{TH} = 0.95$, $bm_{26}^{TH} = 0.95$ and $bm_{22}^{TH} = 0.95$ (as opposed to the current $p_{9}^{u} = 0.04$, $p_{4}^{u} = 0.06$, $p_{25}^{u} = 0.84$, $p_{26}^{u} = 0.90$ and $p_{22}^{u} = 0.92$). These values refer to the improvements we want to make with respect to the aforementioned undesirable levels of brittleness. These are arbitrary values to illustrate the proposed capability.
- No requirements are set for the confidence levels of all other activities ($bm_{default}^{TH}$) in the task network.

With that input the algorithm will suggest the following solution R^{TC}:

- $ra_{9,est}^{TC} \rightarrow -5371$
- $ra_{4,est}^{TC} \rightarrow -381$
- $ra_{26,est}^{TC} \rightarrow -1866$
- $ra_{25,est}^{TC} \rightarrow -976$
- $ra_{22,est}^{TC} \rightarrow -151$
- $ra_{20,est}^{TC} \rightarrow -816$
- $ra_{23,est}^{TC} \rightarrow -1056$

The above relaxation indicates that a designer should decrease the earliest start time of activities a_9, a_4, a_{26}, a_{24}, a_{22}, a_{20} and a_{23} by the indicated amount. It is interesting to see that the system is relaxing not only the temporal constraints associated with the five most brittle activities, but also two additional activities that play a role in the dependency chain shown in Figure 6.11 (b). Activities 20 and 23 are the most brittle activities on the left hand side time window [0, 4544], showing the option of making room for activities in the right hand side time window of Figure 6.11 (b).

If we apply the suggested relaxations to $TN2$, we can run the property-based brittleness analysis with P_{DC} only and analyze the charts to check the impact on the brittleness measures. Figure 6.12 shows the results of such a new run of the analysis, now using the relaxed $TN2$. In Figure 6.12 (1-3) the reduced scale of the y-axis is noticeable. Figure 6.12 (2) in particular shows that the target confidence levels are met and that the brittleness of both the

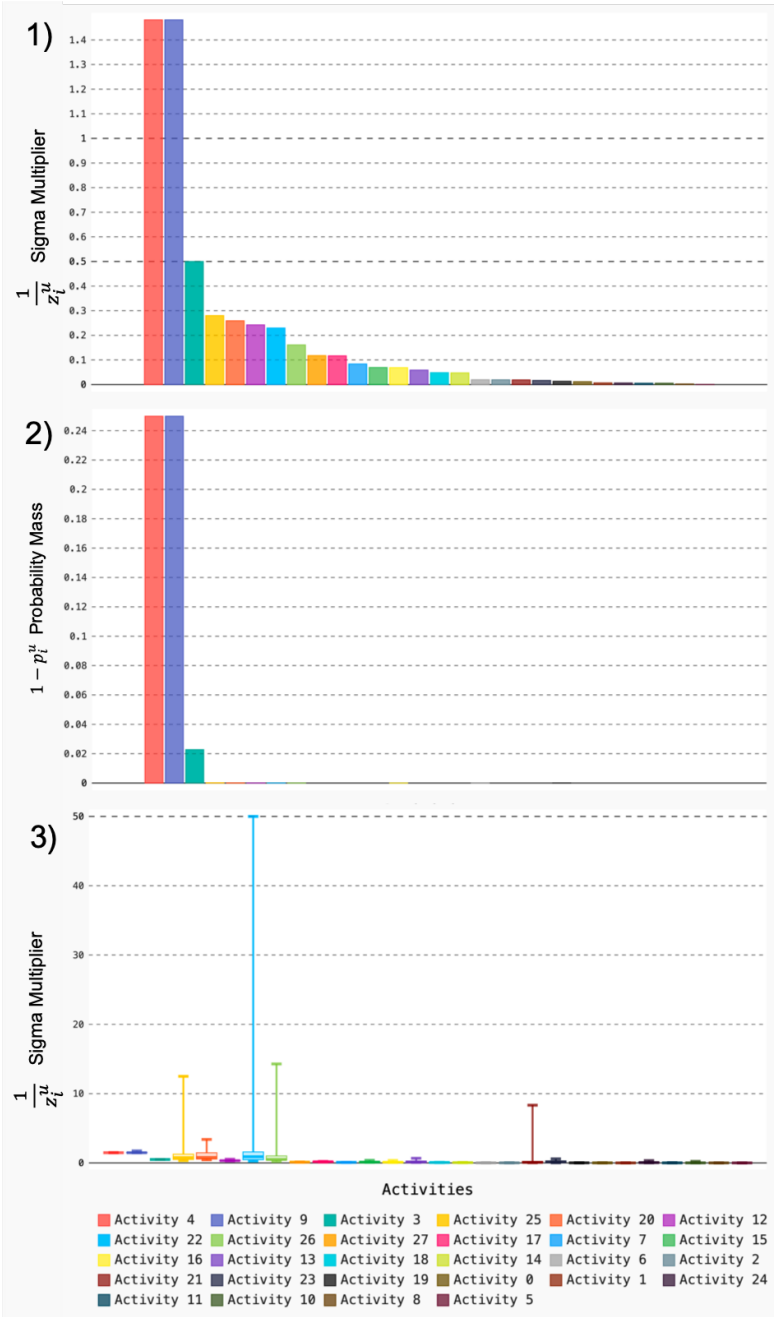


Figure 6.12: Temporally relaxed task network *TN 2* after analysis: 1) Shows the brittleness ranking with respect to the sigma multiplier z_i^u for $P = \{P_{DC}\}$; 2) Shows the same brittleness ranking with respect to the probability mass, $1 - p_i^u$; and 3) Shows the same ranking based on the Monte Carlo approach with variation of sigma multipliers Z_i^u of each contingent activity a_i.

target five activities and the whole task network is reduced to the acceptable values. The activity ranking is different: 26, 25 and 22 are pushed back to lower levels. The resulting relaxed task network can provide a more robust execution, assuming that the suggestions are implemented and agreed on before deployment. We believe that this capability can be quite powerful and useful not only for human planners but also, ultimately, for (offline) automated planners in a generated and test/analyze setting.

6.6 Application: Manufacturing in biotechnology

We now apply our tools to tackle a significantly different operational context. Project management accounts for around 30% of the world gross product [Tur+10]. However, most existing studies have solely looked at machine scheduling environments [HL05]. We consider the real world problem of biotechnology product manufacturing. Our data come from the Belgian biotech company *Zentech*, and concerns the full manufacturing process of one their most popular products.

6.6.1 STNUs versus DTNUs

Temporal networks constitute expressive tools for modeling operational plans, from human operated projects to planetary rover operations. And yet, a STNU aims at formally stating only one of the many possible structural configurations of the network. From the STNU of Fig. 6.11 for example, a different STNU could be obtained by considering a different ordering of activities 21 to 26.

In this context, one may come up with as many different PSTNs as there are possible orderings of the activities, or alternatively use the Disjunctive Temporal Network (DTN) formalism to consider explicit representation of choices or alternatives. DTN extends the formalism of STN by enabling, with the use of disjunctive temporal constraints, the encoding of exponentially many interpretations, in the form of STNs or STNUs, within one single DTN(U). A recent study of DTNs in the context dynamic control can be found in [CMR16].

From operations management to STNUs The manufacturing of a biotech products involves many different operations that must be scheduled amongst human operators, whilst fulfiling a set of constraints. In particular, the manufacturing process considered in this case study is depicted in Figure 6.13. Operations management eventually requires evaluating STNUs. The problem faced by the *Zentech* biotech company can not be fully encoded with a simple temporal network. In fact, it should be noticed that Figure 6.13 does not look like a STN. Instead, it gives a visual representation of some complex *job-shop*

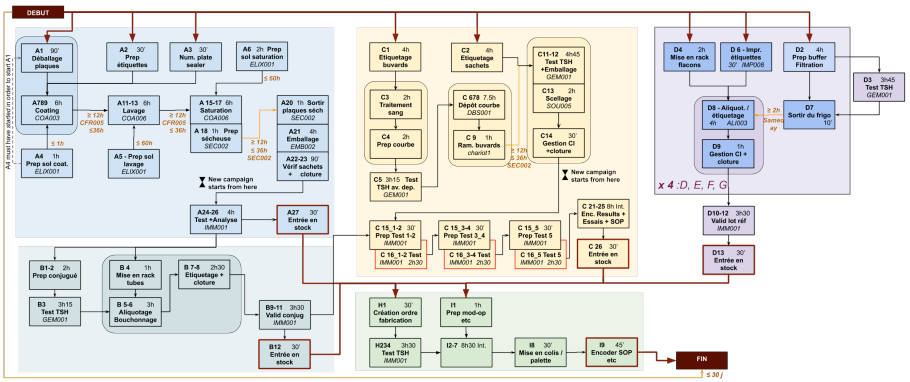


Figure 6.13: Activities and constraints involved in the manufacturing process of the biotech product at stake. Each box is a specific activity and each arrow a temporal constraint.

scheduling problem [LK79] with resources and temporal constraints as well as temporal uncertainty. Given a limited set of human operators and machine resources, the goal is to distribute and order all the activities amongst them. But whereas a job-shop scheduling problem only amounts to solving the assignment and ordering problem, in this case each operator's schedule must also state the specific times at which each activity must be started, in order to best avoid operational issues such as resource collision. In other words, the operations management problem should be formulated as a Disjunctive Temporal Network under Uncertainty (DTNU).

Solving a DTNU amounts to enumerating (most of) its possible interpretations, each being a particular STNU. Namely, each possible activities to operators assignment and ordering can be represented as a STNU, in which the orderings are predetermined but time values are still to be decided. This is why, in any operations management problem in which temporal values are of interest, such as human operated manufacturing processes for instance, one always eventually faces the problem of evaluating STNUs. Figure 6.14 shows an example of an excerpt of a possible full planning (operators' assignment and starting times) for the manufacturing problem described in Fig. 6.13. Any such planning can be expressed as a STNU in order to exploit techniques from the simple temporal network domain, such as the proposed brittleness analysis.

Scope and considered STNUs In the context of this case study, we are interested in assessing the reliability of our operational plans. We are interested in plans with high probability that *"everything happens as expected"*, which we define here as the probability that every task gets operated the day it has been planned to, in addition to all the constraints (such as the one stating a

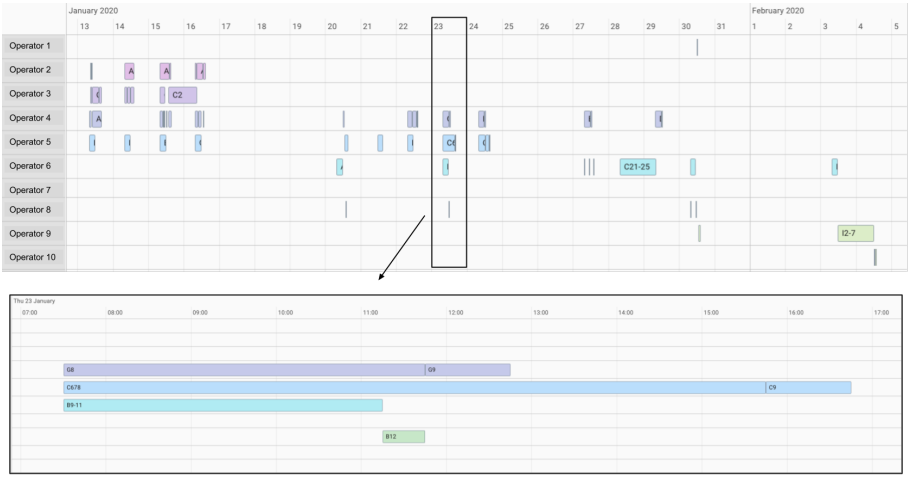


Figure 6.14: An example of operational planning in bio-manufacturing, which corresponds to a particular solution to the problem depicted in Figure 6.13.

maximum of 32 hours between A18 and A20 in Fig. 6.13) being respected. Therefore, a STNU is derived from each particular planning, each planning being a particular solution to the scheduling (*i.e.* DTNU) problem. Here we selected five different solutions, obtained by using a job-shop robust optimization algorithm [Sai19], leading to five different STNUs. Within each STNU, in addition to all the constraints already listed in Fig. 6.13, a time window is imposed on every activity, corresponding to the planned operating time (start of time window) and the end of the related working day (end of time window). For example, in Fig. 6.14 activity C678 would be imposed a time window spanning from its *a priori* planned starting time, 7:30 am, to the end of the working day, defined as 5 pm.

Task Network	# Act	# Dep	# Nodes	# Edges	$z_{i,min}^{u,Y^0}$	$z_{i,max}^{u,Y^0}$	$1-p_i^{u,Y^0}$
BioTN 1	70	117	143	469	1.035	175.02	0.15
BioTN 2	70	119	143	471	1.035	265.02	0.15
BioTN 3	70	119	143	471	1.035	565.02	0.15
BioTN 4	70	118	143	470	1.035	302.52	0.15
BioTN 5	70	122	143	470	1.035	265.02	0.15

Table 6.2: Studied biotechnology task networks with their respective number of activities (# Act), total number of activity dependencies (# Dep) (i.e. the total number of direct activity dependencies specified in the manufacturing problem), and the number of nodes and edges in the corresponding STNU. It also includes the metric value for the most brittle activity ($z_{i,min}^{u,Y^0}$) and for the least brittle activity ($z_{i,max}^{u,Y^0}$), and the probability of falling beyond the z_i^{u,Y^0} values in the activity delay case (with $1-p_i^{u,Y^0}$ for the most brittle activity).

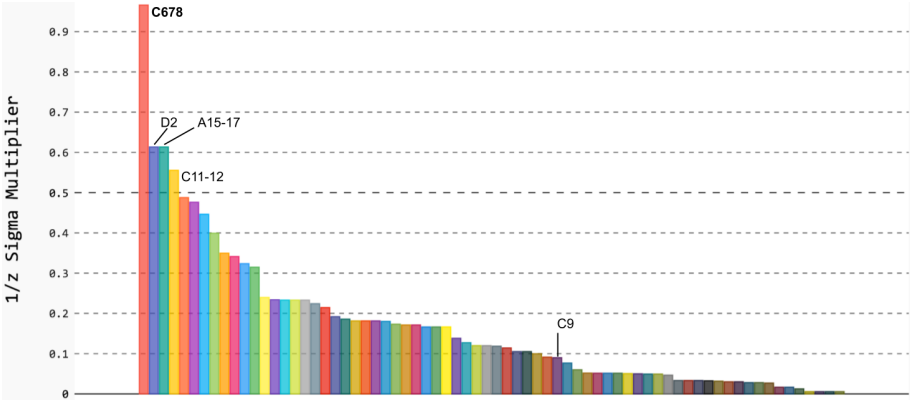


Figure 6.15: Ranking of brittle activities from the STNU based on the production planning of Figure 6.14.

6.6.2 Results

6.6.2.1 Property-based Brittleness Analysis

Figure 6.15 shows the overall brittleness ranking obtained from Algorithm 9, on the bio-manufacturing operational planning of Figure 6.14. According to the results, activity C678 is the most brittle. Contrary to *Activity 9* identified in the M2020 application case, the brittleness here is due to a structural bottleneck, rather than a tight time window with respect to the activity's nominal duration. In fact, by inspection of Figure 6.14, we remark that activity C9 is planned for execution directly after C678, on the same day. As a consequence, the uncertainty around C678's duration has a direct impact on the probability of completing C9 before the end of the working day. Note how C9 comes far away in the brittleness ranking; the bottleneck is not on C9's uncertainty, but clearly on that of C678.

For this biomanufacturing context, the purpose of the brittleness analysis, which identifies the activities being particularly brittle to temporal uncertainty, is to suggest either an improvement of the operators' skills or a rethinking of the activities. For example, the operators could be trained to conduct activity C678 more efficiently, leading to a decrease of both its nominal duration and standard deviation. Alternatively, C678 could be redesigned to be split into two separate activities.

6.6.2.2 Temporal Relaxation to Reduce Brittleness

Suppose activity C678 in BioTN 1 task network cannot be redesigned, being doomed to be processed without any interruption, and that significantly decreasing its total nominal duration or uncertainty is not an option. Then, a

temporal relaxation could provide alternative solutions, such as starting that particular working day earlier, in order to anticipate C678's challenging duration.

An an exercise of the relaxation capability in the activity C678 case, we want to increase the confidence level of activity C678 from 0.70 (computed in the same way as the aforementioned M2020 application for the symmetric confidence level value p_i^u) to 0.90. In this case, we then set the following task network temporal relaxation problem *RN* as the input to the *Brittleness Level Guided Task Network Relaxation* Algorithm:

- TN is our task network example, $BioTN1$.
- $A_{Brittle} = \{a_{C678}\}$ is the target set of brittle activities in $BioTN1$ as informed by the property-based brittleness analysis.
- RA^{TC} is the set the earliest and latest start time of *all activities* in $BioTN1$, including activity C678.
- BM^TH is the set to be the following target brittleness measure threshold, i.e. the target confidence levels: $bm^{TH}_{C678} = 0.90$ (as opposed to the 0.70).
- No requirements are set for the confidence levels of all other activities ($bm^{TH}_{default}$) in the task network.

With that input the algorithm will suggest the following solution R^{TC}: $ra^{TC}_{C678,est} \rightarrow -3600$, which means that if we change the earliest start time of activity C678 to be earlier by 3600 we would reach a less brittle activity C678 with confidence level of 0.9. In practice, this means that the activity should be started one hour earlier, by asking the operators to adapt the corresponding day in order to start one hour sooner, in order to reach the 0.9 confidence level. Note that if we slightly increase the target brittleness measure threshold further, so that $bm^{TH}_{C678} = 0.99$, then the solution R^{TC} becomes $ra^{TC}_{C678,est} \rightarrow -9000$, which would imply starting that working day 2.5 hours sooner than usual!

Towards a better scheduling Our brittleness analysis highlights a strong brittleness in the system which is due to activity C678. By further analysing the schedule shown in Fig. 6.14, it appears that a solution can be found with an alternative schedule, one that does not plan activity C9 directly after C678 on the same day, but postpones it to the day after. This illustrates the usefulness of the proposed brittleness analysis when comparing solutions during the process of solving the underlying scheduling (*i.e.* DTNU) problem. Additionally, the analysis of the five considered STNUs (all being significantly

different near-optimal solutions) all identify C678 as the dominantly brittle activity, suggesting further that the solution is unlikely to lie in the displacement of C9.

6.7 Conclusion

The analysis of temporal networks is a whole field in itself which encompasses many settings. We only looked at the problem of task scheduling, with a focus on the question of activity brittleness. By introducing SMC, the proposed methods for the verification of temporal networks are not necessarily limited to specific classes of properties for which there already exists an exhaustive and complete algorithm.

More precisely, we presented in this chapter a new approach for measuring and analyzing the brittleness of temporal networks with respect to a set of user-specified properties, as well its application in two distinct domains. We introduced a new metric to measure the brittleness of activities and algorithms to help designers/planners identify the most critical activities that cause issues. We also described an advisory system that is capable of suggesting temporal constraint relaxation to adjust the brittleness measure to an acceptable level. We showed results from a selected set of Mars 2020 planetary rover task networks in which two general cases for activity brittleness emerged, highlighting different ways to address them. Lastly, we applied our tools to a job scheduling problem originating from the industry of biomanufacturing.

Future work While we personally do not have the intent on expanding on this work, there are plans for the following:

- explore new sets of brittleness properties dedicated to additional critical available resources
- expand the constraint relaxation mechanisms [YFW15; MH04] to reason about non-temporal properties constraints
- suggest task network relaxation for both temporal and resource constraints
- handle asymmetric and non-parametric distributions in addition to unimodal distributions to represent activity duration models

As far as we are concerned, a continuation of this work would take the form of a reiteration of the transposition of SMC principles, but in yet another setting. However, we also note that the idea of minimum sigma multipliers itself could potentially be transposed to other contexts, maybe to analyze the robustness of CPSs.

Conclusion 7

In this thesis, we studied how SMC can benefit the verification of cyber-physical systems. While SMC offers a powerful alternative to naive methods such as unit testing, one that does not suffer from the flaws of pure and exhaustive formal methods, many challenges remain. We have tried to tackle some of those challenges, such as the issue of scalability, the rare event problem or the difficulty to not get stuck around local extrema with optimization algorithms, both from a theoretical perspective and from a practical perspective. While we have proposed novel solutions and analyzed various promising strategies for those challenges, we have also highlighted the inherent limitations of the field. In particular, it is clear that the tension between the effective usability and the scalability of SMC will remain an important topic of research for many years to come, as real-world use cases ideally demand extreme levels of precision and confidence while simultaneously requiring models of huge size and daunting complexity. Fortunately, there seems to be no shortage of new ideas to try to improve the discipline further. Just in this thesis alone, we may have opened enough doors to keep at least one researcher busy for a few years.

In Chapter 3, we generalized existing adaptive stopping algorithms to provide a universal alternative to the Monte Carlo paradigm. With statistical considerations alone, we showed how smarter sampling strategies can have a huge impact on the performance, efficiency and scalability of estimation algorithms designed in the context of SMC. We successfully applied our research to the rare event problem and discussed the optimality of our results. We genuinely believe that the content of this chapter could have a significant impact on the discipline if it led to a widespread adoption of adaptive stopping algorithms as a replacement for the basic Monte Carlo subroutines present in many of the existing SMC methods.

In Chapter 4, we revisited and improved an optimization algorithm for Markov decision processes. We not only corrected some of its design flaws, but also analyzed its structure and parameters to propose a new version of the algorithm which is remarkably faster without any sacrifice in terms of performance. Moreover, we exploited this opportunity to explore new strategies to solve the local extremum problem, and we described the framework of a genetic algorithm for SMC.

In Chapter 5, we investigated the parallelization of model-based verification techniques. We highlighted how parallelization can be two-fold. On one hand, with careful design, SMC algorithms can be combined and run in parallel at a minimal cost. We hope this realization will encourage collaborations in the field as the future, as it shows that merging the innovations and tools of different researchers can easily lead to practical improvements for the users. On the other hand, the distribution of the simulation process, most often the most costly step when verifying industry-sized systems, can allow existing SMC methods to scale with the use of high-performance computing infrastructures. We demonstrated that the scaling benefits can be expected to be directly linear with the size of the parallel architecture, making it a worthwhile effort.

In Chapter 6, we showed how the ideas of SMC can provide new and unexpected solutions when transposed to settings which are usually not considered by the field. We developed a new framework relying on stochastic methods for the verification of temporal networks, allowing us to present a rich toolbox to help decision-making when working with potentially brittle task networks. We exercised it on two concrete problems: the scheduling of the Perseverance rover's activities and biotech manufacturing. We regard this chapter as a good illustration of another seemingly modest yet consequential way to improve the impact of SMC: connecting it to other areas of research which are related to the development and verification of cyber-physical systems, and making its foundational principles known to more people.

Future work We have previously listed some possible direct continuations of what has been achieved in this thesis at the end of each chapter. To avoid repetition, we focus here on what we think are the most propitious and appealing themes.

From the theoretical point of view, SMC needs to evolve. The randomness of the algorithms must become *guided* randomness rather than *pure* randomness. In the case of SMC, we generally have a lot more control on the generation of samples than with generic stochastic processes since the simulation function is rarely a fully uninterpretable and uncontrollable black-box. In a similar fashion to what stopping algorithms do for estimation and what our genetic algorithm could potentially do for optimization, we think that SMC methods cannot afford anymore to ignore the information they gradually gain about the system as they accumulate simulations. Otherwise, most of the simulations end up having low utility, which means they ideally could or should be skipped. This could significantly improve both the speed and the scalability of many SMC algorithms. Hopefully, guided SMC will be able to close the gap further between SMC and full MC (in terms of effectiveness).

Note that this idea of guiding randomness and smart sample generation

is already present in the literature, albeit not necessarily explicitly. In some sense, shielding [Blo+20; BLX21; Bro+23; Bro+24] is a good example.

From the practical point of view, parallelization should become a reflex in the field. While it will not make SMC applicable in any setting just yet, parallelization can already be enough to bridge the gap for important real-world problems. Furthermore, while brute-forcing with sheer computing power, even when done intelligently, will never revolutionize the scaling capabilities of SMC, distributed algorithms and parallelization should always be considered and implemented by default, regardless of the current developments of the discipline.

Revisiting some of the state-of-the-art SMC algorithms and tools to refactor them to integrate parallelization as systematically as possible could be a very prolific and rewarding endeavour.

Investigating the compatibility of guided strategies for SMC and parallelization further also appears to be an inevitable step.

Additionally, we want to give the reader some spoilers for three possible future papers, two of which have already been worked on over the past half-year. First, a second and definitive paper on the subject of adaptive stopping algorithms revolving around the theme of optimality. The considerations at the end of Chapter 3 were not included in [PL24] but we think they are worth publishing. Second, we think that more can be said in terms of improving the sampling process of SMC methods under the assumption that it can be done without repetition, which sounds like a reasonable assumption for many contexts. Third, we want to concretize and test the idea of the genetic algorithm with a third, complete and successful implementation.

Finally, we mention a few topics that we have spotted in the literature of the recent years that have caught our attention and that we would like to investigate as well if given the opportunity. First, shields [Blo+20; BLX21; Bro+23; Bro+24]. They make it possible to deactivate some actions in state-transition models to limit and guide *a priori* the exploration of possible traces but also to offer safeguards with *a posteriori* modifications of schedulers. We want to determine how well and how far they could properly guide the sample generation process of SMC methods. Second, considering system properties whose truth values depend on multiple traces, which are called hyperproperties [Wan+19; Wan+21; HSB21; Aro+22], is tempting. In particular, we are interested in the associated logic(s): they can be regarded as an (unequivalent) alternative to branching logics. Third, we would like to have a more holistic view of Markov decision processes as system models and the associated SMC algorithms. [Har+25], the extremely recent and revised version of [Har+23], would be a good start.

Bibliography

[Abb+13] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta. "Probabilistic temporal logic falsification of cyber-physical systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 12.2s (2013), pp. 1–30.

[ABD14] E. G. Amparore, M. Beccuti, and S. Donatelli. "(Stochastic) model checking in GreatSPN". In: *Application and Theory of Petri Nets and Concurrency: 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings 35*. Springer. 2014, pp. 354–363.

[Abr+19] J. R. Abrahams, D. A. Chu, G. Diehl, M. Knittel, J. Lin, W. Lloyd, J. C. Boerkoel Jr, and F. Jeremy. "Dream: An algorithm for mitigating the overhead of robust rescheduling". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 3–12.

[AFH96] R. Alur, T. Feder, and T. A. Henzinger. "The benefits of relaxing punctuality". In: *Journal of the ACM (JACM)* 43.1 (1996), pp. 116–146.

[AGP19] V. Atlidakis, P. Godefroid, and M. Polishchuk. "RESTler: stateful REST API fuzzing". In: *ICSE*. IEEE / ACM, 2019, pp. 748–758.

[Agr+19] J. Agrawal, W. Chi, S. Chien, G. Rabideau, S. Kuhn, and D. Gaines. "Enabling Limited Resource-Bounded Disjunction in Scheduling". In: *11th International Workshop on Planning and Scheduling for Space (IWPSS 2019)*. Also appears at the 29th International Conference on Automated Planning and Scheduling (ICAPS) Workshop PlanRob 2019 and ICAPS SPARK 2019 and ICAPS IntEx 2019. Berkeley, California, USA, July 2019, pp. 7–15.

[Akm+19] S. Akmal, S. Ammons, H. Li, and J. C. Boerkoel. "Quantifying Degrees of Controllability in Temporal Networks with Uncertainty". In: *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS-19)*. 2019.

[Ali+03] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. "Definition of Robustness Metric for Resource Allocation". In: *Proc. of the 17th International Parallel and Distributed Processing Symposium (IPDPS)*. France: IEEE, Apr. 2003. ISBN: 0-7695-1926-1.

[Alp] U. G. Alpes. *SBIP*. `https://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html?lang=en` [Accessed: 2025].

[AM11] M. Alturki and J. Meseguer. "PVeStA: A parallel statistical model checking and quantitative analysis tool". In: *International Conference on Algebra and Coalgebra in Computer Science*. 2011, pp. 386–392.

[AMS06] G. Agha, J. Meseguer, and K. Sen. "PMaude: Rewrite-based specification language for probabilistic object systems". In: *Electronic Notes in Theoretical Computer Science* 153.2 (2006), pp. 213–239.

[AMS07a] J.-Y. Audibert, R. Munos, and C. Szepesvári. "Tuning bandit algorithms in stochastic environments". In: *International conference on algorithmic learning theory*. Springer. 2007, pp. 150–165.

[AMS07b] J.-Y. Audibert, R. Munos, and C. Szepesvári. "Variance estimates and exploration function in multi-armed bandit". In: *CERTIS Research Report 07–31*. Citeseer, 2007.

[AP18] G. Agha and K. Palmskog. "A survey of statistical model checking". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 28.1 (2018), pp. 1–39.

[Aro+22] S. Arora, R. R. Hansen, K. G. Larsen, A. Legay, and D. B. Poulsen. "Statistical model checking for probabilistic hyperproperties of real-valued signals". In: *International Symposium on Model Checking Software*. Springer. 2022, pp. 61–78.

[AS65] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. Vol. 55. Courier Corporation, 1965.

[ASB20] S. Amin, T. Salahuddin, and A. Bouras. "Cyber physical systems and smart homes in healthcare: Current state and challenges". In: *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. IEEE. 2020, pp. 302–309.

[Bai+03] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. "Model-checking algorithms for continuous-time Markov chains". In: *IEEE Transactions on software engineering* 29.6 (2003), pp. 524–541.

[Bai98] C. Baier. "On algorithmic verification methods for probabilistic systems". PhD thesis. Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.

[Bak+17] M. E. Bakir, M. Gheorghe, S. Konur, and M. Stannett. "Comparative analysis of statistical model checking tools". In: *Membrane Computing: 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers 17*. Springer. 2017, pp. 119–135.

[Bal+15] P. Ballarini, B. Barbot, M. Duflot, S. Haddad, and N. Pekergin. "HASL: A new approach for performance evaluation and model checking from concepts to experimentation". In: *Performance Evaluation* 90 (2015), pp. 53–77.

[Bar+20] B. Barbot, N. Basset, T. Dang, A. Donzé, J. Kapinski, and T. Yamaguchi. "Falsification of cyber-physical systems with constrained signal spaces". In: *NASA Formal Methods: 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings 12*. Springer. 2020, pp. 420–439.

[Bar+21a] E. Baranov, J. Bowles, T. Given-Wilson, A. Legay, and T. Webber. "A Secure User-Centred Healthcare System: Design and Verification". In: *DATAMOD*. Vol. 13268. LNCS. Springer, 2021, pp. 44–60.

[Bar+21b] E. Baranov, J. Bowles, T. Given-Wilson, A. Legay, and T. Webber. "A Secure User-Centred Healthcare System: Design and Verification". In: *International Symposium: From Data to Models and Back*. Springer. 2021, pp. 44–60.

[Bas+22a] D. Basile, M. H. ter Beek, A. Ferrari, and A. Legay. "Exploring the ERTMS/ETCS full moving block specification: an experience with formal methods". In: *STTT* 24.3 (2022), pp. 351–370.

[Bas+22b] D. Basile, M. H. ter Beek, A. Ferrari, and A. Legay. "Exploring the ERTMS/ETCS full moving block specification: an experience with formal methods". In: *International Journal on Software Tools for Technology Transfer* 24.3 (2022), pp. 351–370.

[Ben+96] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. *UPPAAL—a tool suite for automatic verification of real-time systems*. Springer, 1996.

[BG11] R. Baheti and H. Gill. "Cyber-physical systems". In: *The impact of control technology* 12.1 (2011), pp. 161–166.

[BHH12] J. Bogdoll, A. Hartmanns, and H. Hermanns. "Simulation and statistical model checking for modestly nondeterministic models". In: *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Springer. 2012, pp. 249–252.

[BHP12] B. Barbot, S. Haddad, and C. Picaronny. "Coupling and importance sampling for statistical model checking". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2012, pp. 331–346.

[Bit16] A. Bit-Monnot. "Temporal and Hierarchical Models for Planning and Acting in Robotics". PhD thesis. LAAS-CNRS, Dec. 2016.

[BK08] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[BLB03] S. Boucheron, G. Lugosi, and O. Bousquet. "Concentration inequalities". In: *Summer school on machine learning*. Springer, 2003, pp. 208–240.

[Blo+20] R. Bloem, P. G. Jensen, B. Könighofer, K. G. Larsen, F. Lorber, and A. Palmisano. "It's Time to Play Safe: Shield Synthesis for Timed Systems". In: *arXiv preprint arXiv:2006.16688* (2020).

[BLX21] O. Bastani, S. Li, and A. Xu. "Safe Reinforcement Learning via Statistical Model Predictive Shielding." In: *Robotics: Science and Systems*. 2021, pp. 1–13.

[Bor+14] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. "Maximizing social influence in nearly optimal time". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 946–957.

[Boy+13] B. Boyer, K. Corre, A. Legay, and S. Sedwards. "PLASMA-lab: A flexible, distributable statistical model checking library". In: *International Conference on Quantitative Evaluation of Systems*. Springer. 2013, pp. 160–164.

[Bro+05] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner. "Model-based testing of reactive systems". In: *Volume 3472 of Springer LNCS*. Springer. 2005.

[Bro+15] J. Brooks, E. Reed, A. Gruver, and J. C. Boerkoel. "Robustness in Probabilistic Temporal Planning". In: *Proc. of the 29th AAAI Conf. on Artificial Intelligence*. 2015, pp. 3239–3246.

[Bro+23] A. H. Brorholt, P. G. Jensen, K. G. Larsen, F. Lorber, and C. Schilling. "Shielded reinforcement learning for hybrid systems". In: *International Conference on Bridging the Gap between AI and Reality*. Springer. 2023, pp. 33–54.

[Bro+24] A. H. Brorholt, A. H. Høeg-Petersen, K. G. Larsen, and C. Schilling. "Efficient shield synthesis via state-space transformation". In: *International Conference on Bridging the Gap between AI and Reality*. Springer. 2024, pp. 206–224.

[BS07] J. K. Bradley and R. E. Schapire. "Filterboost: Regression and classification on large datasets". In: *Advances in neural information processing systems* 20 (2007).

[Bul+11] P. Bulychev, A. David, K. G. Larsen, M. Mikučionis, and A. Legay. "Distributed parametric and statistical model checking". In: *arXiv preprint arXiv:1111.0370* (2011).

[Bus+13] R. Busa-Fekete, B. Szorenyi, W. Cheng, P. Weng, and E. Hüllermeier. "Top-k selection based on adaptive sampling of noisy preferences". In: *International Conference on Machine Learning*. PMLR. 2013, pp. 1094–1102.

[CB21] G. Casella and R. L. Berger. *Statistical inference*. Cengage Learning, 2021.

[CGL83] T. F. Chan, G. H. Golub, and R. J. LeVeque. "Algorithms for computing the sample variance: Analysis and recommendations". In: *The American Statistician* 37.3 (1983), pp. 242–247.

[Che+16] M. Chen, S. Huang, X. Fu, X. Liu, and J. He. "Statistical model checking-based evaluation and optimization for cloud workflow resource allocation". In: *IEEE Transactions on Cloud Computing* 8.2 (2016), pp. 443–458.

[Che+17] B. Chen, Z. Yang, S. Huang, X. Du, Z. Cui, J. Bhimani, X. Xie, and N. Mi. "Cyber-physical system enabled nearby traffic flow modelling for autonomous vehicles". In: *2017 IEEE 36th international performance computing and communications conference (IPCCC)*. IEEE. 2017, pp. 1–6.

[Chi+18] W. Chi, S. Chien, J. Agrawal, G. Rabideau, E. Benowitz, D. Gaines, E. Fosse, S. Kuhn, and J. Biehl. "Embedding a Scheduler in Execution for a Planetary Rover". In: *Proc. of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*. Delft, Netherlands, June 2018.

[CL06] F. Chung Graham and L. Lu. *Complex Graphs and Networks American Mathematical Society*. 2006.

[Cla+11] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani. "Model checking and the state explosion problem". In: *LASER Summer School on Software Engineering*. Springer, 2011, pp. 1–30.

[Cla+12] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani. "Model checking and the state explosion problem". In: *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures* (2012), pp. 1–30.

[Cla+18a] K. Claessen, N. Smallbone, J. Eddeland, Z. Ramezani, and K. Åkesson. "Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems". In: *IFAC-PapersOnLine* 51.7 (2018), pp. 408–415.

[Cla+18b] E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith. *Model checking, 2nd Edition*. MIT Press, 2018.

[Cla+18c] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, et al. *Handbook of model checking*. Vol. 10. Springer, 2018.

[CMR16] A. Cimatti, A. Micheli, and M. Roveri. "Dynamic Controllability of Disjunctive Temporal Networks : Validation and Synthesis of Executable Strategies". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2016, pp. 3116–3122.

[Col+15] A. Colombo, D. Fontanelli, A. Legay, L. Palopoli, and S. Sedwards. "Efficient customisable dynamic motion planning for assistive robots in complex human environments". In: *JAISE* 7.5 (2015), pp. 617–634.

[Com+19] C. Combi, R. Posenato, L. Viganò, and M. Zavatteri. "Conditional Simple Temporal Networks with Uncertainty and Resources". In: *J. Artif. Int. Res.* 64.1 (Jan. 2019), pp. 931–985. ISSN: 1076-9757. DOI: 10.1613/jair.1.11453.

[COS98] A. Cesta, A. Oddi, and S. F. Smith. "Profile-based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems". In: *Proc. of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS)*. Pittsburgh, Pennsylvania, USA, 1998, pp. 214–223. ISBN: 1-57735-052-9.

[CRS02] S. Chien, L. Rasmussen, and A. Sinclair. "Clifford algebras and approximating the permanent". In: *Proceedings of the thiryfourth annual ACM symposium on Theory of computing*. 2002, pp. 222–231.

[Cui+15] J. Cui, P. Yu, C. Fang, P. Haslum, and B. C. Williams. "Optimising Bounds in Simple Temporal Networks with Uncertainty under Dynamic Controllability Constraints". In: *Proc. of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*. 2015, pp. 52–60.

[CW96] E. M. Clarke and J. M. Wing. "Formal methods: State of the art and future directions". In: *ACM Computing Surveys (CSUR)* 28.4 (1996), pp. 626–643.

[CZ11] E. M. Clarke and P. Zuliani. "Statistical model checking for cyber-physical systems". In: *International symposium on automated technology for verification and analysis*. Springer. 2011, pp. 1–12.

[Dag+00] P. Dagum, R. Karp, M. Luby, and S. Ross. "An optimal algorithm for Monte Carlo estimation". In: *SIAM Journal on computing* 29.5 (2000), pp. 1484–1496.

[DAr+15a] P. R. D'Argenio, A. Legay, S. Sedwards, and L. Traonouez. "Smart sampling for lightweight verification of Markov decision processes". In: *STTT* 17.4 (2015), pp. 469–484.

[DAr+15b] P. D'Argenio, A. Legay, S. Sedwards, and L.-M. Traonouez. "Smart sampling for lightweight verification of Markov decision processes". In: *International Journal on Software Tools for Technology Transfer* 17 (2015), pp. 469–484.

[Dav+12] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, and S. Sedwards. "Statistical model checking for stochastic hybrid systems". In: *arXiv preprint arXiv:1208.3856* (2012).

[Dav+15] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen. "Uppaal SMC tutorial". In: *International journal on software tools for technology transfer* 17.4 (2015), pp. 397–415.

[De +07] R. De Nicola, J.-P. Katoen, D. Latella, M. Loreti, and M. Massink. "Model checking mobile stochastic logic". In: *Theoretical Computer Science* 382.1 (2007), pp. 42–70.

[Dey+18] N. Dey, A. S. Ashour, F. Shi, S. J. Fong, and J. M. R. Tavares. "Medical cyber-physical systems: A survey". In: *Journal of medical systems* 42 (2018), pp. 1–13.

[DGW02] C. Domingo, R. Gavalda, and O. Watanabe. "Adaptive sampling methods for scaling up knowledge discovery algorithms". In: *Data Mining and Knowledge Discovery* 6 (2002), pp. 131–152.

[DHS08] S. Donatelli, S. Haddad, and J. Sproston. "Model checking timed and stochastic properties with CSL^{TA}". In: *IEEE Transactions on Software Engineering* 35.2 (2008), pp. 224–240.

[DMC18] Q. Do, B. Martini, and K.-K. R. Choo. "Cyber-physical systems information gathering: A smart home case study". In: *Computer Networks* 138 (2018), pp. 1–12.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. "Temporal Constraint Networks". In: *Artificial Intelligence* 49.1-3 (May 1991), pp. 61–95. ISSN: 0004-3702. DOI: 10.1016/0004-3702(91)90006-6.

[Dob+23] O. Dobe, S. Schupp, E. Bartocci, B. Bonakdarpour, A. Legay, M. Pajic, and Y. Wang. "Lightweight Verification of Hyperproperties". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2023, pp. 3–25.

[DOK-ING] DOK-ING. *DOK-ING d.o.o.* 2024. URL: https://dok-ing.hr/.

[Dok+15] A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan, and G. Fainekos. "Requirements driven falsification with coverage metrics". In: *2015 International Conference on Embedded Software (EMSOFT)*. IEEE. 2015, pp. 31–40.

[Don+20] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma. "A survey on ensemble learning". In: *Frontiers of Computer Science* 14 (2020), pp. 241–258.

[Dup+22] S. Dupont, A. Yautsiukhin, G. Ginis, G. Iadarola, S. Fagnano, F. Martinelli, C. Ponsard, A. Legay, and P. Massonet. "Product Incremental Security Risk Assessment Using DevSecOps Practices". In: *SECPRE*. Vol. 13785. LNCS. 2022.

[Dvo+14] F. Dvorák, R. Barták, A. Bit-Monnot, F. Ingrand, and M. Ghallab. "Planning and Acting with Temporal and Hierarchical Decomposition Models". In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. 2014, pp. 115–121. DOI: 10.1109/ICTAI.2014.27.

[DW+00] C. Domingo, O. Watanabe, et al. "MadaBoost: A modification of AdaBoost". In: *COLT*. 2000, pp. 180–189.

[EMM02] E. Even-Dar, S. Mannor, and Y. Mansour. "PAC bounds for multi-armed bandit and Markov decision processes". In: *International Conference on Computational Learning Theory*. Springer. 2002, pp. 255–270.

[EMS04] J.-P. Eckmann, E. Moses, and D. Sergi. "Entropy of dialogues creates coherent structures in e-mail traffic". In: *Proceedings of the National Academy of Sciences* 101.40 (2004), pp. 14333–14337.

[EMT23] M. Esposito, T. Mancini, and E. Tronci. "Optimizing Fault-Tolerant Quality-Guaranteed Sensor Deployments for UAV Localization in Critical Areas via Computational Geometry". In: *IEEE Transactions on Systems, Man and Cybernetics: Systems* (2023). DOI: 10.1109/TSMC.2023.3327432.

[EP+22a] M. Esposito, L. Picchiami, et al. "Estimation-Based Verification of Cyber-Physical Systems via Statistical Model Checking." In: *HYDRA/RCRA@ LPNMR*. 2022, pp. 51–63.

[EP+22b] M. Esposito, L. Picchiami, et al. "Formal Certification of Surrogate Models for Cyber-Physical Systems Verification". In: *CEUR WORKSHOP PROCEEDINGS*. Vol. 3311. 2022, pp. 63–71.

[Ern+22] G. Ernst, P. Arcaini, G. Fainekos, F. Formica, J. Inoue, T. Khandait, M. M. Mahboob, C. Menghi, G. Pedrielli, M. Waga, et al. "ARCH-COMP 2022 Category Report: Falsification with Ubounded Resources". In: *Proceedings of 9th International Workshop on Applied*. Vol. 90. 2022, pp. 204–221.

[Eurostat] Eurostat. *Road safety: 20,640 people died in a road crash last year – progress remains too slow*. 2023. URL: https://transport.ec.europa.eu/news-events/news/road-safety-20640-people-died-road-crash-last-year-progress-remains-too-slow-2023-10-19%5C_en.

[Fan+17] C. Fan, B. Qi, S. Mitra, and M. Viswanathan. "DryVR: Data-driven verification and compositional reasoning for automotive systems". In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. Springer. 2017, pp. 441–461.

[FK05] M. Fürer and S. P. Kasiviswanathan. "An almost linear time approximation algorithm for the permanent of a random (0-1) matrix". In: *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science: 24th International Conference, Chennai, India, December 16-18, 2004. Proceedings 24*. Springer. 2005, pp. 263–274.

[Für00] M. Fürer. "Approximating permanents of complex matrices". In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 667–669.

[FYW14] C. Fang, P. Yu, and B. C. Williams. “Chance-constrained Probabilistic Simple Temporal Problems”. In: *Proc. of the 28th AAAI Conf. on Artificial Intelligence*. Québec City, Québec, Canada, 2014, pp. 2264–2270.

[Gai+16] D. Gaines, G. Doran, H. Justice, G. Rabideau, S. Schaffer, V. Verma, K. Wagstaff, V. Vasavada, W. Huffman, R. Anderson, R. Mackey, and T. Estlin. “Productivity challenges for Mars rover operations: A case study of Mars Science Laboratory operations”. In: *Technical Report D-97908, Jet Propulsion Laboratory* (Jan. 2016).

[Gil+19] M. Gil, M. Albert, J. Fons, and V. Pelechano. “Designing human-in-the-loop autonomous cyber-physical systems”. In: *International journal of human-computer studies* 130 (2019), pp. 21–39.

[GLS99] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.

[God20] P. Godefroid. “Fuzzing: hack, art, and science”. In: *Commun. ACM* 63.2 (2020), pp. 70–76.

[GS04] R. Grosu and S. A. Smolka. “Quantitative Model checking.” In: *ISoLA (Preliminary proceedings)* 2004 (2004), p. 6.

[GS05] R. Grosu and S. A. Smolka. “Monte carlo model checking”. In: *Tools and Algorithms for the Construction and Analysis of Systems: 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005. Proceedings 11*. Springer. 2005, pp. 271–286.

[HAF14] B. Hoxha, H. Abbas, and G. Fainekos. “Benchmarks for Temporal Logic Requirements for Automotive Systems.” In: *ARCH@ CPSWeek* 34 (2014), pp. 25–30.

[Har+23] A. Hartmanns, S. Junges, T. Quatmann, and M. Weininger. “A practitioner’s guide to MDP model checking algorithms”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2023, pp. 469–488.

[Har+25] A. Hartmanns, S. Junges, T. Quatmann, and M. Weininger. “The revised practitioner’s guide to MDP model checking algorithms”. In: *International Journal on Software Tools for Technology Transfer* (2025).

[HDF18] B. Hoxha, A. Dokhanchi, and G. Fainekos. "Mining parametric temporal logic properties in model-based design for cyber-physical systems". In: *International Journal on Software Tools for Technology Transfer* 20 (2018), pp. 79–93.

[Hér+04] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. "Approximate probabilistic model checking". In: *Verification, Model Checking, and Abstract Interpretation: 5th International Conference, VMCAI 2004 Venice, Italy, January 11-13, 2004 Proceedings 5*. Springer. 2004, pp. 73–84.

[HH14] A. Hartmanns and H. Hermanns. "The Modest Toolset: An integrated environment for quantitative modelling and verification". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2014, pp. 593–598.

[HI09a] V. Heidrich-Meisner and C. Igel. "Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 401–408.

[HI09b] V. Heidrich-Meisner and C. Igel. "Neuroevolution strategies for episodic reinforcement learning". In: *Journal of Algorithms* 64.4 (2009), pp. 152–168.

[HJ94] H. Hansson and B. Jonsson. "A logic for reasoning about time and reliability". In: *Formal aspects of computing* 6 (1994), pp. 512–535.

[HL05] W. Herroelen and R. Leus. "Project scheduling under uncertainty: Survey and research potentials". In: *European Journal of Operational Research* 165.2 (2005), pp. 289–306. ISSN: 03772217. DOI: 10.1016/j.ejor.2004.04.002.

[HLV02] M. Huguet, P. Lopez, and T. Vidal. "Dynamic task sequencing in temporal problems with uncertainty". In: *Workshop on On-Line Planning and Scheduling in Sixth International Conference on AI Planning & Scheduling (AIPS'02)*. Toulouse, France, Apr. 2002, pp. 41–48.

[Hoe94] W. Hoeffding. "Probability inequalities for sums of bounded random variables". In: *The collected works of Wassily Hoeffding* (1994), pp. 409–426.

[HP04] N. G. Hall and M. E. Posner. "Sensitivity Analysis for Scheduling Problems". In: *Journal of Scheduling* 7.1 (Jan. 2004), pp. 49–83. ISSN: 1094-6136. DOI: 10.1023/B:JOSH.0000013055.31639.f6.

[HR02] K. Havelund and G. Roşu. "Synthesizing monitors for safety properties". In: *Tools and Algorithms for the Construction and Analysis of Systems: 8th International Conference, TACAS 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8–12, 2002 Proceedings 8*. Springer. 2002, pp. 342–356.

[HRS13] M. Heiner, C. Rohr, and M. Schwarick. "MARCIE–model checking and reachability analysis done efficiently". In: *Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings 34*. Springer. 2013, pp. 389–399.

[HSB21] T.-H. Hsu, C. Sánchez, and B. Bonakdarpour. "Bounded model checking for hyperproperties". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2021, pp. 94–112.

[Hua+18] A. Huang, L. Lloyd, M. Omar, and J. C. Boerkoel. "New Perspectives on Flexibility in Simple Temporal Planning". In: *Proc. of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*. Delft, Netherlands, June 2018, pp. 123–131.

[Hub06] M. Huber. "Exact sampling from perfect matchings of dense regular bipartite graphs". In: *Algorithmica* 44 (2006), pp. 183–193.

[IEA] I. E. Agency. *World Energy Outlook 2020*. 2020. URL: https://www.iea.org/reports/world-energy-outlook-2020.

[Jae+19] M. Jaeger, P. Jensen, K. G. Larsen, A. Legay, S. Sedwards, and J. H. Taankvist. "Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs". In: *ATVA*. Vol. 11781. LNCS. Springer, 2019.

[Jet18] Jet Propulsion Laboratory. *Mars 2020 rover mission*. https://mars.nasa.gov/mars2020. Accessed: 2019-03-04. 2018.

[Jha+09] S. K. Jha, E. M. Clarke, C. J. Langmead, A. Legay, A. Platzer, and P. Zuliani. "A bayesian approach to model checking biological systems". In: *Computational Methods in Systems Biology:*

7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings 7. Springer. 2009, pp. 218–234.

[JLS12a] C. Jegourel, A. Legay, and S. Sedwards. "A platform for high performance statistical model checking–PLASMA". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2012, pp. 498–503.

[JLS12b] C. Jegourel, A. Legay, and S. Sedwards. "Cross-entropy optimisation of importance sampling parameters for statistical model checking". In: *International Conference on Computer Aided Verification*. Springer. 2012, pp. 327–342.

[JLS13] C. Jegourel, A. Legay, and S. Sedwards. "Importance splitting for statistical model checking rare properties". In: *International Conference on Computer Aided Verification*. Springer. 2013, pp. 576–591.

[JLS16] C. Jegourel, A. Legay, and S. Sedwards. "Command-based importance sampling for statistical model checking". In: *Theoretical Computer Science* 649 (2016), pp. 1–24.

[Kah50] H. Kahn. "Random sampling (Monte Carlo) techniques in neutron attenuation problems. I". In: *Nucleonics (US) Ceased publication* 6.See also NSA 3-990 (1950).

[Kam12] N. Kamide. "Bounded linear-time temporal logic: A proof-theoretic investigation". In: *Annals of Pure and Applied Logic* 163.4 (2012), pp. 439–466.

[KH51] H. Kahn and T. E. Harris. "Estimation of particle transmission by random sampling". In: *National Bureau of Standards applied mathematics series* 12 (1951), pp. 27–30.

[Kim+13] J. Kim, H. Kim, K. Lakshmanan, and R. Rajkumar. "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car". In: *Proceedings of the ACM/IEEE 4th international conference on cyber-physical systems*. 2013, pp. 31–40.

[KM53] H. Kahn and A. W. Marshall. "Methods of reducing sample size in Monte Carlo computations". In: *Journal of the Operations Research Society of America* 1.5 (1953), pp. 263–278.

[KNP] M. Kwiatkowska, G. Norman, and D. Parker. "Analysis of a Gossip Protocol in PRISM". In: *ACM SIGMETRICS Performance Evaluation* 36 (), pp. 17–22.

[KNP08] M. Kwiatkowska, G. Norman, and D. Parker. "Analysis of a gossip protocol in PRISM". In: *ACM SIGMETRICS Performance Evaluation Review* 36.3 (2008), pp. 17–22.

[KNP11a] M. Z. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *CAV*. LNCS. 2011.

[KNP11b] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of probabilistic real-time systems". In: *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*. Springer. 2011, pp. 585–591.

[KNS02a] M. Kwiatkowska, G. Norman, and J. Sproston. "Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol". In: *Proc. PAPM/PROBMIV'02*. Vol. 2399. LNCS. 2002.

[KNS02b] M. Kwiatkowska, G. Norman, and J. Sproston. "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol". In: *Joint International Workshop von Process Algebra and Probabilistic Methods, Performance Modeling and Verification*. Springer. 2002, pp. 169–187.

[Knu14] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison-Wesley Professional, 2014.

[KPV14] M. Karsai, N. Perra, and A. Vespignani. "Time varying networks and the weakness of strong ties". In: *Scientific reports* 4.1 (2014), p. 4001.

[Kum+18] T. K. S. Kumar, Z. Wang, A. Kumar, C. M. Rogers, and C. A. Knoblock. "Load scheduling of simple temporal networks under dynamic resource pricing". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[Lab03] P. Laborie. "Resource Temporal Networks: Definition and Complexity". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 948–953.

[Lan+18] J. Lanet, H. L. Bouder, M. Benattou, and A. Legay. "When time meets test". In: *Int. J. Inf. Sec.* 17.4 (2018), pp. 395–409.

[Lar+22] K. Larsen, A. Legay, G. Nolte, M. Schlüter, M. Stoelinga, and B. Steffen. "Formal methods meet machine learning (F3ML)". In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2022, pp. 393–405.

[LDB10] A. Legay, B. Delahaye, and S. Bensalem. "Statistical model checking: An overview". In: *International conference on runtime verification*. Springer. 2010, pp. 122–135.

[Leg+19a] A. Legay, A. Lukina, L. Traonouez, J. Yang, S. A. Smolka, and R. Grosu. "Statistical Model Checking". In: *Computing and Software Science - State of the Art and Perspectives*. Vol. 10000. LNCS. 2019.

[Leg+19b] A. Legay, A. Lukina, L. M. Traonouez, J. Yang, S. A. Smolka, and R. Grosu. "Statistical model checking". In: *Computing and software science: state of the art and perspectives*. Springer, 2019, pp. 478–504.

[Let+17] A. Letichevsky, O. Letychevskyi, V. Skobelev, and V. Volkov. "Cyber-physical systems". In: *Cybernetics and Systems Analysis* 53 (2017), pp. 821–834.

[LGC19] A. Lambora, K. Gupta, and K. Chopra. "Genetic algorithm-A literature review". In: *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE. 2019, pp. 380–384.

[Lia+07] H. Liang, L. Shi, F. Bai, and X. Liu. "Random path method with pivoting for computing permanents of matrices". In: *Applied mathematics and computation* 185.1 (2007), pp. 59–71.

[Lin+12] Y. Lin, C. Chou, Y. Lai, T. Huang, S. Chung, J. Hung, and F. C. Lin. "Test coverage optimization for large code problems". In: *J. Syst. Softw.* 85.1 (2012), pp. 16–27.

[Lin74] R. F. Ling. "Comparison of several algorithms for computing sample means and variances". In: *Journal of the American Statistical Association* 69.348 (1974), pp. 859–866.

[LK79] J. K. Lenstra and A. H. G. R. Kan. "Computational complexity of discrete optimization problems". In: *Annals of Discrete Mathematics* 4 (1979), pp. 121–140.

[LL14] K. G. Larsen and A. Legay. "Statistical Model Checking Past, Present, and Future: (Track Introduction)". In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer. 2014, pp. 135–142.

[LL16] K. G. Larsen and A. Legay. "Statistical model checking: past, present, and future". In: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, Oc-*

tober 10–14, 2016, Proceedings, Part I 7. Springer. 2016, pp. 3–15.

[LL20] K. G. Larsen and A. Legay. "30 Years of Statistical Model Checking". In: *ISOLA*. Vol. 12476. LNCS. Springer, 2020, pp. 325–330.

[LOB19] J. Y. Lee, V. Ojha, and J. C. Boerkoel. "Measuring and Optimizing Durability Against Scheduling Disturbances". In: *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*. Berkeley, CA, July 2019.

[LST14] A. Legay, S. Sedwards, and L.-M. Traonouez. "Scalable verification of Markov decision processes". In: *International conference on software engineering and formal methods*. Springer. 2014, pp. 350–362.

[LST16a] A. Legay, S. Sedwards, and L. Traonouez. "Plasma Lab: A Modular Statistical Model Checking Platform". In: *ISOLA*. Vol. 9952. LNCS. 2016, pp. 77–93.

[LST16b] A. Legay, S. Sedwards, and L.-M. Traonouez. "Plasma lab: a modular statistical model checking platform". In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2016, pp. 77–93.

[Lui+17] L. Luisa Vissat, M. Loreti, L. Nenzi, J. Hillston, and G. Marion. "Three-Valued Spatio-Temporal Logic: a further analysis on spatio-temporal properties of stochastic systems". In: *Quantitative Evaluation of Systems: 14th International Conference, QEST 2017, Berlin, Germany, September 5-7, 2017, Proceedings 14*. Springer. 2017, pp. 317–332.

[Lun+17] K. Lund, S. Dietrich, S. Chow, and J. C. Boerkoel. "Robust Execution of Probabilistic Temporal Plans". In: *Proc. of the 21st AAAI Conf. on Artificial Intelligence*. 2017, pp. 3597–3604.

[Man+13] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, and E. Tronci. "System level formal verification via model checking driven simulation". In: *International Conference on Computer Aided Verification*. Springer. 2013, pp. 296–312.

[Man+14] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. "Anytime system level verification via random exhaustive hardware in the loop simulation". In: *2014 17th Euromicro Conference on Digital System Design*. IEEE. 2014, pp. 236–245.

[Man+16a] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. "Anytime system level verification via parallel random exhaustive hardware in the loop simulation". In: *Microprocessors and Microsystems* 41 (2016), pp. 12–28.

[Man+16b] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. "SyLVaaS: System level formal verification as a service". In: *Fundamenta Informaticae* 149.1-2 (2016), pp. 101–132.

[Man+17] T. Mancini, F. Mari, A. Massini, I. Melatti, I. Salvo, and E. Tronci. "On minimising the maximum expected verification time". In: *Information Processing Letters* 122 (2017), pp. 8–16.

[Man+21] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. "On checking equivalence of simulation scripts". In: *Journal of Logical and Algebraic Methods in Programming* 120 (2021), p. 100640.

[Mat23a] Mathworks. *Developing the Apollo Lunar Module Digital Autopilot, https://nl.mathworks.com/help/simulink/slref/developing-the-apollo-lunar-module-digital-autopilot.html, accessed: 25/01/2023.* 2023.

[Mat23b] Mathworks. *Model an Automatic Transmission Controllert, https://it.mathworks.com/help/simulink/slref/modeling-an-automatic-transmission-controller.html, accessed: 25/01/2023.* 2023.

[Mat23c] Mathworks. *Modeling a Fault-Tolerant Fuel Control System, https://nl.mathworks.com/help/simulink/slref/modeling-a-fault-tolerant-fuel-control-system.html, accessed: 25/01/2023.* 2023.

[Med+18] B. L. Mediouni, A. Nouri, M. Bozga, M. Dellabani, A. Legay, and S. Bensalem. "Bip 2.0: Statistical model checking stochastic real-time systems". In: *International symposium on automated technology for verification and analysis*. Springer. 2018, pp. 536–542.

[MH04] L. Michel and P. V. Hentenryck. "Iterative Relaxations for Iterative Flattening in Cumulative Scheduling". In: *Proc. of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*. Whistler, British Columbia, Canada, June 2004, pp. 200–208.

[MMT21] T. Mancini, I. Melatti, and E. Tronci. "Any-horizon uniform random sampling and enumeration of constrained scenarios for simulation-based formal verification". In: *IEEE Transactions on Software Engineering* (2021).

[MMV01] P. Morris, N. Muscettola, and T. Vidal. "Dynamic Control of Plans with Temporal Uncertainty". In: *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI) - Volume 1*. Seattle, WA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 494–499. ISBN: 1-55860-812-5, 978-1-558-60812-2.

[MN00] M. Matsumoto and T. Nishimura. "Dynamic creation of pseudorandom number generators". In: *Monte-Carlo and Quasi-Monte Carlo Methods 1998: Proceedings of a Conference held at the Claremont Graduate University, Claremont, California, USA, June 22–26, 1998*. Springer. 2000, pp. 56–69.

[MN04] O. Maler and D. Nickovic. "Monitoring temporal properties of continuous signals". In: *International symposium on formal techniques in real-time and fault-tolerant systems*. Springer. 2004, pp. 152–166.

[Mor14] P. Morris. "Dynamic Controllability and Dispatchability Relationships". In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR*. Vol. 8451. Lecture Notes in Computer Science. Springer, 2014, pp. 464–479.

[MSA08] V. Mnih, C. Szepesvári, and J.-Y. Audibert. "Empirical Bernstein stopping". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 672–679.

[MU49] N. Metropolis and S. Ulam. "The monte carlo method". In: *Journal of the American statistical association* 44.247 (1949), pp. 335–341.

[NKD18] M. Nilsson, J. Kvarnström, and P. Doherty. "Planning with Temporal Uncertainty, Resources and Non-Linear Control Parameters". In: *ICAPS*. 2018.

[Nou+18] A. Nouri, B. L. Mediouni, M. Bozga, J. Combaz, S. Bensalem, and A. Legay. "Performance evaluation of stochastic real-time systems with the SBIP framework". In: *International Journal of Critical Computer-Based Systems* 8.3-4 (2018), pp. 340–370.

[NPS13] G. Norman, D. Parker, and J. Sproston. "Model checking for probabilistic timed automata". In: *Formal methods in system design* 43 (2013), pp. 164–190.

[NTD16] H. T. Nguyen, M. T. Thai, and T. N. Dinh. "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks". In: *Proceedings of the 2016 international conference on management of data*. 2016, pp. 695–710.

[Oka59a] M. Okamoto. "Some Inequalities Relating to the Partial Sum of Binomial Probabilities". In: *Annals of the institute of Statistical Mathematics* 10 (1959).

[Oka59b] M. Okamoto. "Some inequalities relating to the partial sum of binomial probabilities". In: *Annals of the institute of Statistical Mathematics* 10 (1959), pp. 29–35.

[Oxf] U. of Oxford. *PRISM*. `https://www.prismmodelchecker.org/manual/RunningPRISM/StatisticalModelChecking` [Accessed: 2025].

[Pai+20] A. Paigwar, E. Baranov, A. Renzaglia, C. Laugier, and A.Legay. "Probabilistic Collision Risk Estimation for Autonomous Driving: Validation via Statistical Model Checking". In: *IV*. IEEE, 2020.

[Pap+20] A. Pappagallo et al. "Estimation of distribution parameters as a tool for model-based system engineering and model identification". In: (2020).

[PMT20a] A. Pappagallo, A. Massini, and E. Tronci. "Monte carlo based statistical model checking of cyber-physical systems: A review". In: *Information*. Vol. 11. 2020, p. 588.

[PMT20b] A. Pappagallo, A. Massini, and E. Tronci. "Monte carlo based statistical model checking of cyber-physical systems: A review". In: *Information* 11.12 (2020), p. 588.

[Pol+17] E. Polgreen, V. B. Wijesuriya, S. Haesaert, and A. Abate. "Automated experiment design for data-efficient verification of parametric Markov decision processes". In: *Quantitative Evaluation of Systems: 14th International Conference, QEST 2017, Berlin, Germany, September 5-7, 2017, Proceedings 14*. Springer. 2017, pp. 259–274.

[PRISM] PRISM. *PRISM Case Studies*. 2024. URL: `https://www.prismmodelchecker.org/casestudies/index.php`.

[Put14] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[RB17] G. Rabideau and E. Benowitz. "Prototyping an Onboard Scheduler for the Mars 2020 Rover". In: *International Workshop on Planning and Scheduling for Space (IWPSS)*. Pittsburgh, PA, June 2017.

[RDK16] R. Rajkumar, D. De Niz, and M. Klein. *Cyber-physical systems*. Addison-Wesley Professional, 2016.

[Rei+15] D. Reijsbergen, P.-T. De Boer, W. Scheinhardt, and B. Haverkort. "On hypothesis testing for statistical model checking". In: *International journal on software tools for technology transfer* 17 (2015), pp. 377–395.

[Roz11] K. Y. Rozier. "Linear temporal logic symbolic model checking". In: *Computer Science Review* 5.2 (2011), pp. 163–203.

[RR55] M. N. Rosenbluth and A. W. Rosenbluth. "Monte Carlo calculation of the average extension of molecular chains". In: *The Journal of Chemical Physics* 23.2 (1955), pp. 356–359.

[Rui+16] E. Ruijters, D. Guck, P. Drolenga, M. Peters, and M. Stoelinga. "Maintenance analysis and optimization via statistical model checking: Evaluating a train pneumatic compressor". In: *Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings 13*. Springer. 2016, pp. 331–347.

[Sai+20] M. Saint-Guillain, T. Stegun Vaquero, J. Agrawal, and S. Chien. "Robustness Computation of Dynamic Controllability in Probabilistic Temporal Networks with Ordinary Distributions". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by C. Bessiere. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 4168–4175. DOI: 10.24963/ijcai.2020/576.

[Sai+21] M. Saint-Guillain, T. Stegun Vaquero, S. Chien, and J. Abrahams. "Probabilistic Temporal Networks with Ordinary Distributions: Theory, Robustness and Expected Utility". In: *Journal of Artificial Intelligence Research* (2021).

[Sai19] M. Saint-Guillain. "Robust Operations Management on Mars". In: *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*. Berkeley, CA, July 2019.

[San+16a] P. Santana, T. Vaquero, C. Toledo, A. Wang, C. Fang, and B. Williams. "PARIS: A Polynomial-Time, Risk-Sensitive Scheduling Algorithm for Probabilistic Simple Temporal Networks with Uncertainty". In: *Proc. of the 26th International Conference on Automated Planning and Scheduling*. 2016.

[San+16b] P. Santana, T. Vaquero, C. Toledo, A. Wang, C. Fang, and B. Williams. "Paris: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 26. 2016, pp. 267–275.

[Sid+16] S. Sidor, P. Yu, C. Fang, and B. C. Williams. "Time Resource Networks". In: *CoRR* abs/1602.03203 (2016).

[SM14] A. S. Singh and M. B. Masuku. "Sampling techniques & determination of sample size in applied statistics research: An overview". In: *International Journal of economics, commerce and management* 2.11 (2014), pp. 1–22.

[Son13] E. D. Sontag. *Mathematical control theory: deterministic finite dimensional systems*. Vol. 6. Springer Science & Business Media, 2013.

[SR18] O. Sagi and L. Rokach. "Ensemble learning: A survey". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (2018), e1249.

[Ste+11] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, V. Colizza, L. Isella, C. Régis, J.-F. Pinton, N. Khanafer, W. Van den Broeck, et al. "Simulation of an SEIR infectious disease model on the dynamic contact network of conference attendees". In: *BMC medicine* 9 (2011), pp. 1–15.

[Sto91] Y. Stoskov. "Stability of an optimal schedule". In: *European Journal of Operational Research* 55.1 (1991), pp. 91–102. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(91)90194-Z.

[SV13] S. Sebastio and A. Vandin. "MultiVeStA: Statistical model checking for discrete event simulators". In: (2013).

[SVA04] K. Sen, M. Viswanathan, and G. Agha. "Statistical model checking of black-box probabilistic systems". In: *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings 16*. Springer. 2004, pp. 202–215.

[SVA05] K. Sen, M. Viswanathan, and G. Agha. "On statistical model checking of stochastic systems". In: *International conference on computer aided verification*. Springer. 2005, pp. 266–280.

[Tem10] N. M. Temme. *Error functions, Dawson's and Fresnel integrals*. 2010.

[Tsa02] I. Tsamardinos. "A Probabilistic Approach to Robust Execution of Temporal Plans with Uncertainty". In: *Methods and Applications of Artificial Intelligence*. Ed. by I. P. Vlahavas and C. D. Spyropoulos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 97–108. ISBN: 978-3-540-46014-5.

[TSX15] Y. Tang, Y. Shi, and X. Xiao. "Influence maximization in near-linear time: A martingale approach". In: *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 2015, pp. 1539–1554.

[TU] U. of Twente and S. University. *MODEST*. `https://www.modestchecker.net/` [Accessed: 2025].

[Tur+10] R. J. Turner, M. Huemann, F. T. Anbari, and C. N. Bredillet. *Perspectives on projects*. Routledge, 2010.

[TXS14] Y. Tang, X. Xiao, and Y. Shi. "Influence maximization: Near-optimal time complexity meets practical efficiency". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 75–86.

[Umb+18] A. Umbrico, A. Cesta, M. C. Mayer, and A. Orlandini. "Integrating Resource Management and Timeline-Based Planning". In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*. Ed. by M. de Weerdt, S. Koenig, G. Röger, and M. T. J. Spaan. AAAI Press, 2018, pp. 264–272.

[Vaq+19] T. Vaquero, S. Chien, J. Agrawal, W. Chi, and T. Huntsberger. "Temporal Brittleness Analysis of Task Networks for Planetary Rovers". In: *29th International Conference on Automated Planning and Scheduling (ICAPS'19)*. 2019.

[VF99] T. Vidal and H. Fargier. "Handling contingency in temporal constraint networks: from consistency to controllabilities". In: *Journal of Experimental and Theoretical Artificial Intelligence* 11 (1999), pp. 23–45.

[VG02] T. Vidal and M. Ghallab. "Dealing with Uncertain Durations In Temporal Constraint Networks dedicated to Planning". In: *European Conference on Artificial Intelligence (ECAI)*. Ed. by W. Wahlster. Jan. 3, 2002, pp. 48–54.

[Wan+19] Y. Wang, M. Zarei, B. Bonakdarpour, and M. Pajic. "Statistical verification of hyperproperties for cyber-physical systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 18.5s (2019), pp. 1–23.

[Wan+21] Y. Wang, S. Nalluri, B. Bonakdarpour, and M. Pajic. "Statistical model checking for hyperproperties". In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE. 2021, pp. 1–16.

[Was04] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2004.

[Wel62] B. P. Welford. "Note on a method for calculating corrected sums of squares and products". In: *Technometrics* 4.3 (1962), pp. 419–420.

[WF00] R. J. Wallace and E. C. Freuder. "Dispatchable Execution of Schedules Involving Consumable Resources". In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*. AIPS'00. Breckenridge, CO, USA: AAAI Press, 2000, pp. 283–290. ISBN: 1577351118.

[Whi80] W. Whitt. "Continuity of generalized semi-Markov processes". In: *Mathematics of operations research* 5.4 (1980), pp. 494–501.

[Wil+14] M. Wilson, T. Klos, C. Witteveen, and B. Huisman. "Flexibility and decoupling in Simple Temporal Networks". In: *Artificial Intelligence* 214 (2014), pp. 26–44. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.05.003.

[Wol09] W. Wolf. "Cyber-physical systems". In: *Computer* 42.03 (2009), pp. 88–89.

[YFW15] P. Yu, C. Fang, and B. Williams. "Resolving Over-constrained Probabilistic Temporal Problems Through Chance Constraint Relaxation". In: *Proc. of the 29th AAAI Conference on Artificial Intelligence*. Austin, Texas, 2015, pp. 3425–3431. ISBN: 0-262-51129-0.

[You04] H. L. S. Younes. *Verification and planning for stochastic processes with asynchronous events*. Carnegie Mellon University, 2004.

[You05] H. L. Younes. "Ymer: A statistical model checker". In: *International Conference on Computer Aided Verification*. Springer. 2005, pp. 429–433.

[YS02] H. L. Younes and R. G. Simmons. "Probabilistic verification of discrete event systems using acceptance sampling". In: *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*. Springer. 2002, pp. 223–235.

[YS06] H. S. Younes and R. G. Simmons. "Statistical probabilistic model checking with a focus on time-bounded properties". In: *Inf. Comput.* 204 (2006).

[Zan17] S. Zanero. "Cyber-physical systems". In: *Computer* 50.4 (2017), pp. 14–16.

[Zhe+15] X. Zheng, C. Julien, M. Kim, and S. Khurshid. "Perceptions on the state of the art in verification and validation in cyber-physical systems". In: *IEEE Systems Journal* 11.4 (2015), pp. 2614–2627.

[ZPC10] P. Zuliani, A. Platzer, and E. M. Clarke. "Bayesian statistical model checking with application to simulink/stateflow verification". In: *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. 2010, pp. 243–252.

[ZPC13] P. Zuliani, A. Platzer, and E. M. Clarke. "Bayesian statistical model checking with application to Stateflow/Simulink verification". In: *Formal Methods in System Design* 43 (2013), pp. 338–367.