Clopper-Pearson Algorithms for Efficient Statistical Model Checking Estimation

Hao Bu and Meng Sun

***Abstract*—Statistical model checking (SMC) is a simulation-based formal verification technique to deal with the scalability problem faced by traditional model checking. The main workflow of SMC is to perform iterative simulations. The number of simulations depends on users' requirement for the verification results, which can be very large if users require a high level of confidence and precision. Therefore, how to perform as fewer simulations as possible while achieving the same level of confidence and precision is one of the core problems of SMC. In this paper, we consider the estimation problem of SMC. Most existing statistical model checkers use the Okamoto bound to decide the simulation number. Although the Okamoto bound is sound, it is well known to be overly conservative. The simulation number decided by the Okamoto bound is usually much higher than it actually needs, which leads to a waste of time and computation resources. To tackle this problem, we propose an efficient, sound and lightweight estimation algorithm using the Clopper-Pearson confidence interval. We perform comprehensive numerical experiments and case studies to evaluate the performance of our algorithm, and the results show that our algorithm uses 40%-60% fewer simulations than the Okamoto bound. Our algorithm can be directly integrated into existing model checkers to reduce the verification time of SMC estimation problems.**

***Index Terms*—Statistical model checking, formal methods, quantitative verification.**

I. Introduction

THE verification of stochastic systems (e.g. discrete-time Markov chains and continuous-time Markov chains) is crucial for developing trustworthy software. Model checking [1] is a widely-used formal method to verify properties of stochastic systems. To formally verify a stochastic system, traditional model checking algorithms consider all states and use intensive numerical computation to obtain the probability that the given property is satisfied [2], [3], [4], so they usually face the scalability problem. Specifically, when the system contains a large number of states, traditional model checking algorithms usually require a large amount of space (hence of time too). To solve the scalability problem, in around 2004, researchers proposed the statistical model checking (SMC) [5], [6], [7], [8], [9], [10], which is a simulation-based formal verification technique. Instead of considering all states, SMC performs iterative simulations on the system to be verified. In each simulation, SMC generates a finite path of the given system and observes whether this path satisfies the given property. After a certain number of simulations, we can obtain enough information about the probability that the property is satisfied. As SMC relies on simulations, its time cost is usually acceptable (there is no need to consider all states). Moreover, as each simulation is independent with each other, its memory requirement is quite low (only the current path needs to be stored), and most SMC algorithms can be easily parallelized.

There are two types of SMC verification problems [11]: the *hypothesis testing problem* [5] and the *estimation problem* [7]. The hypothesis testing problem (also called "qualitative analysis") aims to answer whether the probability of the given property is higher or lower than a given bound. For example, we want to know whether the probability that a system will fail within T time units is greater than 0.01. The estimation problem (also called "quantitative analysis") aims to estimate the probability of the given property. For example, we want to know the probability that the system will fail within T time units. Due to the stochastic nature of SMC, it cannot provide "exact" results like traditional model checking. Therefore, users need to manually set the desired confidence and precision. For example, for estimation problems, users need to set the confidence δ and the precision ε, and the difference between the estimation result and the true result should be less than ε with more than $1-\delta$ probability. The foundation of an SMC algorithm is to decide the simulation number needed to achieve the desired confidence and precision. Although the hypothesis testing problem and the estimation problem can be transformed into each other, for computational efficiency, people design different algorithms to decide simulation numbers for these two problems. The hypothesis testing problem has been extensively studied in history, and all existing model checkers [12], [13], [14] use the *sequential probability ratio test* [15] to decide the simulation number. However, the research on the SMC estimation problem is quite immature. In the past 20 years, the SMC estimation algorithm is almost unchanged. Currently most model checkers [12], [16], [17] use the *Okamoto bound* [18] to decide the simulation number, which is well known to be

Manuscript received 10 September 2023; revised 25 February 2024; accepted 15 April 2024. Date of publication 23 April 2024; date of current version 18 July 2024. This work was supported in part by the National Key R&D Program of China under Grant 2022YFB2702200, in part by the NSFC under Grant 62172019, and in part by the CCF-Huawei Populus Grove Fund. Recommended for acceptance by D. Bianculli. *(Corresponding author: Meng Sun.)*

The authors are with the School of Mathematical Sciences, Peking University, Beijing 100871, China (e-mail: buhao@pku.edu.cn; sunm@pku.edu.cn).

Digital Object Identifier 10.1109/TSE.2024.3392720

overly conservative [13], [19], [20]. As the Okamoto bound is the foundation of SMC estimation algorithms, any improvement on it is valuable.

The SMC estimation problem can be formalized as the parameter estimation problem of Bernoulli variables (shown in Problem 1). Here p is the probability that the given property is satisfied, and $X = 1$ ($X = 0$) means the property is (not) satisfied in one simulation.

Problem 1: X is a Bernoulli variable with the following distribution: $Pr(X = 1) = p, Pr(X = 0) = 1 - p$. Given precision ε and confidence δ, we want to estimate the probability p. An estimation algorithm is *sound* if for any $0 \leq p \leq 1$, precision $0 < \varepsilon < 1$ and confidence $0 < \delta < 1$, its estimation result $\hat{p}$ satisfies:

$$Pr(|\hat{p} - p| > \varepsilon) \leq \delta$$

To solve the SMC estimation problem in Problem 1, most model checkers use the Okamoto bound to decide the simulation number. Specifically, given ε and δ, they perform $\lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$ independent simulations and use the frequency of the property being satisfied as the estimation result (see Section II-A for details). Although simple to implement, the Okamoto bound has two critical problems. First, the Okamoto bound is very conservative. For example, when $\varepsilon = \delta = 0.01$, the Okamoto bound is 26492, but using our proposed algorithm, we find that 16684 simulations can already achieve the given confidence and precision. Second, the Okamoto bound is independent with the probability p to be estimated. However, researchers find that when p is near 0 or 1, the number of simulations needed is much lower than that when p is near 0.5 [19], [21]. This is particularly important for SMC problems, because in many cases, we want to verify safety properties of systems. That is, we want to prove that the probability of the bad (good) event is very small (large). If we can reduce the simulation number when p is near 0 or 1, we can significantly reduce the cost for verifying safety properties. For the first problem, we propose the *interval-sensitive bound* using the Clopper-Pearson confidence interval [22]. Compared with the Okamoto bound, our interval-sensitive bound can reduce the simulation number by 35%-50%. Although our interval-sensitive bound does not have a closed form like the Okamoto bound, it can be efficiently calculated with very little time overhead. For the second problem, we propose the *adaptive estimation algorithm*, which uses a small number of pre-simulations to roughly estimate p and then chooses proper simulation strategies for different p. Our adaptive estimation algorithm is built on the interval-sensitive bound and can reduce the simulation number by an extra 5%-25% on average. For safety properties with p near 0 or 1, it can even reduce the simulation number by 97%. Our adaptive estimation algorithm is easy to implement and has little extra time cost. It can be directly integrated into existing model checkers to reduce the time cost of solving SMC estimation problems.

The rest of this paper is structured as follows. In Section II, we introduce some basic concepts that will be used later. In Section III, we investigate existing SMC estimation algorithms. In Section IV, we introduce our algorithm in detail and prove its soundness. In Section V, we conduct comprehensive evaluations to evaluate our algorithm. We make some further discussion in Section VI and finally conclude in Section VII.

II. Preliminary

A. Okamoto Bound and Massart Bound

Lemma 1 (Okamoto bound) [18]: For any Bernoulli parameter $0 \leq p \leq 1$ and precision $0 < \varepsilon < 1$, suppose we sample $n > 0$ independent Bernoulli variables $X_1, X_2, \cdots, X_n$ with parameter p (i.e. $Pr(X_i = 1) = p, Pr(X_i = 0) = 1 - p$), then:

$$Pr(|\hat{p_n} - p| > \varepsilon) \leq 2 \cdot \exp(-2n\varepsilon^2)$$

where $\hat{p_n} \triangleq \frac{\sum_{i=1}^{n} X_i}{n}$.

Therefore, we can simply perform $n = \lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$ (derived from $2 \cdot \exp(-2n\varepsilon^2) \leq \delta$) independent simulations and use $\hat{p_n}$ as the estimation result to solve Problem 1.

The Okamoto bound is actually a special case of the more general *Chernoff-Hoeddfing bound* [23], which is designed for all bounded random variables. When used for Bernoulli variables, the Chernoff-Hoeddfing bound can derive the *Massart bound* [24]:

$$Pr(|\hat{p_n} - p| > \varepsilon) \leq 2 \cdot \exp(-n\varepsilon^2 m(p, \varepsilon))$$

where

$$m(p, \varepsilon) = \begin{cases} \dfrac{9}{2(3p + \varepsilon)(3(1 - p) - \varepsilon)}, & 0 \leq p < \dfrac{1}{2} \\ \dfrac{9}{2(3(1 - p) + \varepsilon)(3p - \varepsilon)}, & \dfrac{1}{2} \leq p \leq 1. \end{cases}$$

It can be shown that $m(p, \varepsilon) \geq 2$ for all $p \in [0, 1]$, and if we replace $m(p, \varepsilon)$ by 2, the Massart bound becomes the Okamoto bound. One advantage of the Okamoto bound is that it is independent with p thus is easy to calculate and use. On the other hand, this can also be regarded as a drawback of the Okamoto bound, because $m(p, \varepsilon)$ is much larger than 2 when p is near 0 and 1, which makes the Okamoto bound very conservative in this situation.

Jegourel et al. [19] propose a sequential algorithm using the above Massart bound to solve Problem 1. Specifically, they perform simulations one by one. After each simulation, they calculate a confidence interval $[a, b]$ for p, then calculate the lower bound t of $m(p, \varepsilon)$ when $p \in [a, b]$ and use t to replace $m(p, \varepsilon)$ in the Massart bound. The algorithm returns the frequency $\hat{p_n}$ as the estimation result when $2 \cdot \exp(-n\varepsilon^2 \cdot t)$ is lower than a pre-calculated bound. The main idea of their algorithm is that when p is near 0 and 1, the aforementioned t will be much larger than 2, which makes the Massart bound much tighter than the Okamoto bound.

B. Clopper-Pearson Confidence Interval

Given confidence $0 < \delta < 1$, suppose we sample $n > 0$ independent Bernoulli variables with success probability p and get

x successes ($X = 1$), then the corresponding Clopper-Pearson confidence interval [22] $CP\_int(n, x, \delta)$ for p is:

$$\begin{cases} \left[0, 1 - \sqrt[n]{\frac{\delta}{2}}\right], x = 0 \\ \left[B\left(x, n - x + 1, \frac{\delta}{2}\right), B\left(x + 1, n - x, 1 - \frac{\delta}{2}\right)\right], \\ \qquad\qquad\qquad\qquad 0 < x < n \\ \left[\sqrt[n]{\frac{\delta}{2}}, 1\right], x = n \end{cases}$$

where B is the beta distribution quantile function. The beta distribution quantile function is available in many programming languages (e.g. the `scipy.special.betaincinv` function in Python), so it is easy to calculate Clopper-Pearson confidence intervals with the help of a computer.

As an example, suppose we perform $n = 1000$ Bernoulli trials and find $x = 400$ successes. We want to use these results to estimate the unknown success probability p. Intuitively, p is most likely to be near $\frac{x}{n} = 0.4$. Apart from n and x, we also need to select the confidence δ before we calculate the confidence interval. The confidence δ means that p is in the calculated confidence interval with more than $1 - \delta$ probability. If we set $\delta = 0.05$ (or 0.01), then the Clopper-Pearson confidence interval is $[0.3695, 0.4311]$ (or $[0.3602, 0.4408]$). Therefore, we can conclude that p is in $[0.3695, 0.4311]$ (or $[0.3602, 0.4408]$) with more than 95% (or 99%) probability. Moreover, for the same n and δ, the length of a Clopper-Pearson confidence interval can vary with different x. For example, the confidence interval for $n = 1000, x = 300, \delta = 0.05$ is $[0.2717, 0.3295]$, and its length (0.3295−0.2717=0.0578) is smaller than that when $x = 400$ (0.4311−0.3695=0.0616).

The Clopper-Pearson confidence interval is specially designed for Bernoulli variables and is very accurate. On the other hand, the Okamoto bound is derived from the Chernoff-Hoeddfing bound [23], which is designed for all bounded random variables. Therefore, the Okamoto bound (and other variants of the Chernoff-Hoeddfing bound, e.g. the Massart bound [24]) is less accurate than the Clopper-Pearson confidence interval when used for Bernoulli variables. For example, if we set $n = 1000$ and $\delta = 0.05$ in the Okamoto bound $n = \lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$, we can derive $\varepsilon = 0.0429$. According to Lemma 1, we have $Pr(|\hat{p_n} - p| > 0.0429) \leq 0.05$, where $\hat{p_n} = \frac{x}{n}$. Therefore, when the success number $x = 400$, we know p is in $[\frac{400}{1000} - 0.0429, \frac{400}{1000} + 0.0429] = [0.3571, 0.4429]$ with more than 95% probability. The above interval (derived from the Okamoto bound) is wider than the corresponding Clopper-Pearson confidence interval ($[0.3695, 0.4311]$). In some SMC problems, we simply need a confidence interval for p. In this situation, most statistical model checkers [12], [14], [20] directly return the Clopper-Pearson confidence interval because it is more accurate.

However, sometimes we want to explicitly control the length of the confidence interval (estimation error). For example, in Problem 1, we hope $p \in [\hat{p} - \varepsilon, \hat{p} + \varepsilon]$. If we can guarantee that the length of the confidence interval for p is always lower than $2 \cdot \varepsilon$, then we can use the midpoint of the confidence interval as the estimation result $\hat{p}$. However, as we have mentioned before, the length of a Clopper-Pearson confidence interval is not fixed even using the same n and δ, so we will meet some difficulties if we want to use it for Problem 1. On the other hand, the Okamoto bound always produces a fixed error bound ε when n and δ are fixed (see the example in the previous paragraph). Therefore, most statistical model checkers use the Okamoto bound to deal with Problem 1. The main contribution of this paper is that we successfully use the more accurate Clopper-Pearson confidence interval to deal with Problem 1, thus significantly reduce the simulation number.

The Clopper-Pearson confidence interval has some properties that we will use later. Their proofs are straightforward and can be found in [25].

Lemma 2 (Soundness): For any Bernoulli parameter $0 \leq p \leq 1$, sampling number $n > 0$ and confidence $0 < \delta < 1$, we have:

$$Pr(p \in CP\_int(n, x, \delta)) \geq 1 - \delta$$

We use $CP\_int(n, x, \delta).left$ and $CP\_int(n, x, \delta).right$ to denote the left and right endpoint of the Clopper-Pearson confidence interval. We use $CP\_int(n, x, \delta).length \triangleq CP\_int(n, x, \delta).right - CP\_int(n, x, \delta).left$ to denote the length of the Clopper-Pearson confidence interval.

Lemma 3 (Symmetry): For any sampling number $n > 0$, success number $0 \leq x \leq n$ and confidence $0 < \delta < 1$, we have:

$$CP\_int(n, x, \delta).left = 1 - CP\_int(n, n - x, \delta).right$$

Therefore,

$$CP\_int(n, x, \delta).length = CP\_int(n, n - x, \delta).length$$

Lemma 4 (Monotonicity): For any sampling number $n > 0$ and confidence $0 < \delta < 1$, $CP\_int(n, x, \delta).left$ and $CP\_int(n, x, \delta).right$ are both monotonically increasing with the success number x.

III. Related Work

A. Estimation Algorithms in Existing Model Checkers

Since the introduction of SMC about 20 years ago, people have developed many statistical model checkers. Nowadays, most popular model checkers can solve Problem 1. We investigate some of the most famous model checkers and summarize their estimation algorithms for Problem 1 in Table I. Although many model checkers are closed-source, their implementation details of the estimation algorithm can usually be found in corresponding tool papers, manuals or tutorials (presented in the "source" column). We present the release time of the latest version in the "Last update" column.

PRISM [12] refers to the SMC estimation problem as "approximate probabilistic model checking". According to PRISM's online manual [26], it uses the Okamoto bound to solve the problem.

UPPAAL [27] contains a module for SMC called UPPAAL-SMC. According to Section 3 of [13], it implements a naive

TABLE I
SUMMARY OF POPULAR MODEL CHECKERS

Tool	Estimation Algorithm	Source	Last Update
PRISM	Okamoto	Online manual	2023
UPPAAL	Naive sequential	Tool tutorial	2023
COSMOS	Okamoto Chow-Robbins method	Tool paper	2023
MODES	Okamoto Naive sequential Chen's method	Tool paper	2023
APMC	Okamoto	Tool paper	2006
SBIP	Okamoto	Technical report	2018
Plasma Lab	Okamoto	Tool paper	2018

sequential estimation method. Specifically, it performs simulations one by one, and after each simulation, it calculates the Clopper-Pearson confidence interval using existing running results. Once the confidence interval is shorter than $2 \cdot \varepsilon$, it terminates and returns the midpoint of the interval as the estimation result. The developers did not prove its soundness. In fact, this algorithm is known to be unreliable (see Section 3 of [21], Section 4.1.1 of [19] and Section 2.3.1 of [20]).

COSMOS [28] contains two estimation algorithms: "static sample size estimation" and "dynamic sample size estimation". According to Section 4.1 of [14], static sample size estimation directly uses the Okamoto bound, and dynamic sample size estimation uses the Chow-Robbins method [29], which is only asymptotically sound when ε approaches 0.

MODES [20] refers to the SMC estimation problem as "query problem", and it contains three algorithms (see Section 2.3.1-2.3.3 of [20]). The first is the same sequential algorithm as that in UPPAAL-SMC, and they say it is unsound. The second is the Okamoto bound. The third is Chen's method [30], which can only guarantee the soundness on one side, and the soundness on the other side has not been proved (see Theorem 1 of [30] and Section 4.1.2 of [19]).

APMC [7] is proposed in 2004 and is the first statistical model checker that can deal with the estimation problem. In its initial version, it uses a bound that is even more conservative than the Okamoto bound. Later, it is replaced by the Okamoto bound (see Section 2 of [31]).

SBIP uses the PESTIM engine to solve the estimation problem. According to Section 4.3 of [17], it uses the Okamoto bound.

The estimation problem in Plasma Lab is called "quantitative SMC algorithms". According to Section 1 of [16], it uses the Okamoto bound.

In summary, most existing statistical model checkers use the Okamoto bound to deal with Problem 1. There are some other methods, but they all lack soundness proofs thus are unreliable.

B. Estimation Algorithms in the Literature

There are few researches trying to improve the Okamoto bound, and we only find two algorithms that have soundness proofs. They both have not been implemented in any statistical model checker.

Jegourel et al. [19] propose a sequential algorithm (see Section II-A for more details) using the Massart bound. They provide a soundness proof and their experimental results support the soundness of their algorithm. Their algorithm can achieve similar performance with those methods without soundness proofs (e.g. Chen's method in MODES), making it a very promising alternative for existing estimation methods in Section III-A. We will use it as a baseline in our evaluation part.

Frey [21] also proposes a sequential algorithm that is sound. However, Frey's algorithm needs to finetune a hyperparameter that is very hard to compute, especially when ε is small. The authors of [19] report that they meet the overflow problem for $\varepsilon < 0.1$ when calculating the hyperparameter (see Section 4.1.1 and Section 6.1 of [19]). We try to run the program provided by Frey, but we cannot obtain the hyperparameter even after seven days on an Intel Xeon E5-2650 v4 CPU. The developers of UPPAAL-SMC also take this algorithm into consideration (see their online documentation [32]) but they finally decide not to use it. To circumvent this problem, the authors of [19] also try to manually choose the hyperparameter, but they find it will break the soundness of the algorithm (see Section 6.1 of [19]). Therefore, although this algorithm has some theoretical values, it is not suitable for SMC estimation problems in practice.

C. Other Estimation Problems in SMC

In this paper we consider Problem 1, which is the original SMC estimation problem and is supported by most statistical model checkers. To be specific, it is the *absolute estimation problem for Bernoulli variables*. Nowadays, many statistical model checkers also support other estimation problems, and we discuss them in this subsection.

1) Relative Error: In Problem 1, we hope the difference between our estimation result $\hat{p}$ and the probability p is small with high confidence ($Pr(|\hat{p} - p| > \varepsilon) \leq \delta$). The estimation error $|\hat{p} - p| > \varepsilon$ is called *absolute error*. However, sometimes people may hope to control the *relative error*. Specifically, people hope $Pr(|(1-\varepsilon) \cdot p \leq \hat{p} \leq (1+\varepsilon) \cdot p|) \geq 1 - \delta$ (or equivalently, $Pr(|\hat{p} - p| > \varepsilon \cdot p) \leq \delta$).

There are many algorithms to deal with the *relative estimation problem for Bernoulli variables*, for example [19], [30], [33]. The algorithm in [30] has been implemented in MODES to control the relative error of estimation (see Section 2.3.3 of [20]).

There are also some powerful algorithms that can deal with the *relative estimation problem for general random variables*. Dagum et al. [34] propose the $\mathcal{AA}$ algorithm that can control the relative error for any random variable distributed in $[0, 1]$. They prove that the average simulation number of $\mathcal{AA}$ algorithm is optimal within a constant factor. Later, Mnih et al. [35] propose the EBStop algorithm that can deal with any bounded random variable. Currently, the relative estimation problem for general random variables has not been supported in popular statistical model checkers.

2) General Random Variables: In Problem 1, we consider the absolute estimation problem for Bernoulli variables. However, in modern statistical model checkers, we sometimes need to estimate the expectation of more general random variables. For example, in PRISM, models can be augmented with *rewards*, and we may want to estimate the expected rewards of paths that satisfy a given property (e.g. reach a specific state). The distribution of rewards is usually unknown.

There are many algorithms to solve the *absolute estimation problem for general random variables*, and many of them have been implemented in popular statistical model checkers. The Chernoff-Hoeffding bound [23] only requires the random variable to be bounded, and it has been implemented in PRISM (see APMC Method in [26]) and COSMOS (see Section 4.1 of [14]). The Chow-Robbins method [29], although only asymptotically sound, can be used for any random variable with finite variance. It has been implemented in COSMOS (see Section 4.1 of [14]) and MODES (see Section 2.3.1 of [20]). The central limit theorem can be used for general random variables, and the result is reliable if the simulation number is large enough. It is used by PRISM (see ACI Method in [26]), COSMOS (see Section 4.1 of [14]) and MODES (see Section 2.3.1 of [20]) when nothing is known about the distribution of variables.

3) Rare Events: In safety-critical systems, the probability p of certain properties (e.g. failure) can be very small (e.g. 10^{-8}). In this situation, people may hope the estimation error is also very small. Although this is still an estimation problem for Bernoulli variables (absolute or relative), aforementioned methods (e.g. the Okamoto bound) typically cannot work well. Importance sampling [36] and importance splitting [37] are two popular techniques for rare events simulation. They have been implemented in MODES (see Section 3 of [20]), Plasma Lab (see Section 3.2 of [16]) and SBIP (see Section 4.5 of [17]).

IV. Methodology

We have mentioned two main problems for the Okamoto bound in Section I. The first problem is that it is too conservative: the given confidence and precision can be achieved using much fewer simulations than the Okamoto bound. It is because the Okamoto bound is a special case of the Chernoff-Hoeffding bound [23], which is designed for all bounded random variables. To tackle this problem, we consider the Clopper-Pearson confidence interval [22] that is designed for Bernoulli variables only, thus is more accurate. As we have discussed in Section II-B, the main obstacle of using the Clopper-Pearson confidence interval for Problem 1 is that its length varies with different x. To solve this problem, we develop an algorithm that can efficiently calculate the maximum length of Clopper-Pearson confidence intervals for given n and δ. Based on this, we propose the *interval-sensitive bound* that is much better than the Okamoto bound and develop a simple estimation algorithm (Section IV-A). The second problem is that the Okamoto bound is independent with the probability p to be estimated, so it cannot utilize the information of p to further reduce the simulation number. As we have discussed in Section II-A, the estimation problem is usually easier when p is near 0 or 1. Therefore, we further propose the *adaptive estimation algorithm* that is optimized for p near 0 or 1 (Section IV-B). Finally, we perform complexity analysis for the extra cost of our algorithms (Section IV-C).

A. Interval-Sensitive Bound

We consider the interval-sensitive version of Problem 1, in which the probability p is known to be within an interval $[a, b]$.

Problem 2: X is a Bernoulli variable with the following distribution: $Pr(X = 1) = p, Pr(X = 0) = 1 - p$. Given precision ε and confidence δ, we want to estimate the probability p. An estimation algorithm is *sound* if for any $0 \leq a \leq p \leq b \leq 1$, precision $0 < \varepsilon < 1$ and confidence $0 < \delta < 1$, its estimation result $\hat{p}$ satisfies:

$$Pr(|\hat{p} - p| > \varepsilon) \leq \delta$$

Note that Problem 2 is identical to Problem 1 if we set $a = 0$ and $b = 1$.

We use the Clopper-Pearson confidence interval to deal with the above problem. The main idea is to find a simulation number n first, such that the length of $CP\_int(n, x, \delta) \cap [a, b]$ is no larger than $2 \cdot \varepsilon$ for all $x = 0, 1, \cdots, n$. Then we use the midpoint of $CP\_int(n, x, \delta) \cap [a, b]$ as the estimator $\hat{p}$. The simulation number n is called the *interval-sensitive bound* because it is sensitive to the given interval $[a, b]$. The calculation method of the interval-sensitive bound is shown in Algorithm 1, which is a binary search algorithm with the initial lower bound of 0 and the initial upper bound of the Okamoto bound. In line 7 of Algorithm 1, we use the max_length function (shown in Algorithm 2) to calculate the max length of $CP\_int(n, x, \delta) \cap [a, b]$ $(x = 0, 1, \cdots, n)$ and compare it with $2 \cdot \varepsilon$.

When n and δ are given, there are $n + 1$ possible Clopper-Pearson confidence intervals in total (corresponding to $x = 0, 1, \cdots, n$). Therefore, a straightforward method is to enumerate all $n + 1$ possible confidence intervals and calculate the length of $CP\_int(n, x, \delta) \cap [a, b]$. Although it takes little time to calculate the Clopper-Pearson confidence interval, the max_length function is frequently invoked in Algorithm 1, so we develop an efficient algorithm (shown in Algorithm 2) that only needs to consider 5 intervals rather than all $n + 1$ intervals. It first calculates x_1 such that $CP\_int(n, x_1, \delta).left \leq a$ and $CP\_int(n, x_1 + 1, \delta).left > a$ using binary search (line 3, Algorithm 2). Then it calculates x_2 such that $CP\_int(n, x_2, \delta).right < b$ and $CP\_int(n, x_2 + 1, \delta).right \geq b$ (line 4, Algorithm 2). The positions of x_1 and x_2 can be illustrated by Fig. 1. After obtaining x_1 and x_2,

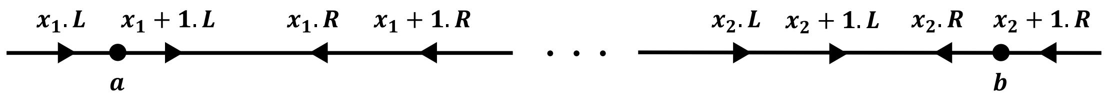

Fig. 1. Illustration for x_1, x_2 in Algorithm 2 and 3. $CP\_int(n,x,\delta).left$ and $CP\_int(n,x,\delta).right$ are abbreviated as $x.L$ and $x.R$. Note that according to Lemma 4, $x.L$ and $x.R$ are both monotonically increasing with x.

Algorithm 1 Function $interval\_sensitive\_bound(\varepsilon, \delta, a, b)$

```
Input: Precision $\varepsilon$, confidence $\delta$, interval $[a,b]$
Output: Simulation number $n$
1:  if $b - a \leq 2 \cdot \varepsilon$ then
2:      return 0
3:  $low = 0$
4:  $high = \lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$ # Okamoto bound
5:  while $high - low > 1$ do
6:      $mid = \lfloor (high + low)/2 \rfloor$
7:      if $max\_length(mid, \delta, a, b) \leq 2 \cdot \varepsilon$ then
8:          $high = mid$
9:      else
10:         $low = mid$
11: return $high$
```

Algorithm 2 Function $max\_length(n, \delta, a, b)$

```
Input: The simulation number $n$, confidence $\delta$, interval $[a,b]$
Output: The max length of $CP\_int(n,x,\delta) \cap [a,b]$ $(x = 0, 1, \cdots, n)$
1: if $CP\_int(n,0,\delta).right \geq b \vee CP\_int(n,n,\delta).left \leq a$ then
2:     return $b - a$
3: Find $x_1$ such that $CP\_int(n,x_1,\delta).left \leq a$ and $CP\_int(n,x_1+1,\delta).left > a$ # using binary search
4: Find $x_2$ such that $CP\_int(n,x_2,\delta).right < b$ and $CP\_int(n,x_2+1,\delta).right \geq b$ # using binary search
5: return $max(CP\_int(n,x_1,\delta) \cap [a,b], CP\_int(n,x_1+1,\delta) \cap [a,b], CP\_int(n,x_2,\delta) \cap [a,b], CP\_int(n,x_2+1,\delta) \cap [a,b], CP\_int(n,\lfloor \frac{n}{2} \rfloor,\delta) \cap [a,b])$
```

we calculate the length of $CP\_int(n,x,\delta) \cap [a,b]$ for $x = x_1, x_1+1, x_2, x_2+1, \lfloor \frac{n}{2} \rfloor$ and return the max length (line 5, Algorithm 2). To prove the soundness of Algorithm 2, we need to answer why we only need to consider these five x rather than all $x = 0, 1, \cdots, n$.

The soundness of Algorithm 2 relies on the fact that $CP\_int(n,x,\delta).left$ and $CP\_int(n,x,\delta).right$ are monotonically increasing with x (see Lemma 4 in Section II-B). Moreover, we find experimentally a monotonicity property about the confidence interval length, which is presented in the following hypothesis.

Hypothesis 1: For any confidence $0 < \delta < 1$ and simulation number $n > 0$, the length of the Clopper-Pearson confidence interval (i.e. $CP\_int(n,x,\delta).right - CP\_int(n,x,\delta).left$) is monotonically increasing with the success number x when $x \leq \frac{n}{2}$.

Using Lemma 3, if Hypothesis 1 holds, then the length of the Clopper-Pearson confidence interval is monotonically decreasing with x when $x \geq \frac{n}{2}$.

With the help of Fig. 1, it is intuitive to figure out the soundness of Algorithm 2. When $x < x_1$, the length of $CP\_int(n,x,\delta) \cap [a,b]$ is smaller than $CP\_int(n,x_1,\delta) \cap [a,b]$ (because $x.L < x_1.L$ and $x.R < x_1.R$ in Fig. 1). Similarly, when $x > x_2 + 1$, the length of $CP\_int(n,x,\delta) \cap [a,b]$ is smaller than $CP\_int(n,x_2+1,\delta) \cap [a,b]$. Therefore, we only need to consider $x_1 \leq x \leq x_2 + 1$. Note that when $x_1 + 1 \leq x \leq x_2$, $CP\_int(n,x,\delta)$ is included in $[a,b]$ (see Fig. 1). According to Hypothesis 1, the max length of $CP\_int(n,x,\delta)$ for $x_1 + 1 \leq x \leq x_2$ is achieved when x is $x_1 + 1$ (if $x_1 + 1 \geq \frac{n}{2}$), x_2 (if $x_2 \leq \frac{n}{2}$) or $\lfloor \frac{n}{2} \rfloor$ (if $x_1 + 1 < \frac{n}{2} < x_2$). Therefore, the max length of $CP\_int(n,x,\delta) \cap [a,b]$ can only occur when x is one of the following five values: $x_1, x_1+1, x_2, x_2+1, \lfloor \frac{n}{2} \rfloor$ (line 5, Algorithm 2). Formally, we have the following theorem.

Theorem 1 (Soundness of Algorithm 2): Under the condition that Hypothesis 1 holds, for any simulation number $n > 0$, confidence $0 < \delta < 1$ and $0 \leq a < b \leq 1$, Algorithm 2 can correctly calculate the max length of $CP\_int(n,x,\delta) \cap [a,b]$ $(x = 0, 1, \cdots, n)$.

Proof: It is obvious that the max length is no larger than $b - a$. Note that $CP\_int(n,0,\delta).left = 0$ (see Section II-B), if $CP\_int(n,0,\delta).right \geq b$, then $CP\_int(n,0,\delta) \cap [a,b] = [a,b]$, so the max length is $b - a$. The case when $CP\_int(n,n,\delta).left \leq a$ is similar.

Now we consider the case when $CP\_int(n,0,\delta).right < b$ and $CP\_int(n,n,\delta).left > a$. As $CP\_int(n,0,\delta).left = 0 \leq a$, $CP\_int(n,n,\delta).left > a$ and $CP\_int(n,x,\delta).left$ is monotonically increasing with x (see Lemma 4), we can successfully obtain x_1 such that $CP\_int(n,x_1,\delta).left \leq a$ and $CP\_int(n,x_1+1,\delta).left > a$ by binary search in line 3 of Algorithm 2. Similarly, we can successfully obtain x_2 in line 4 of Algorithm 2. If $x_1 > x_2 + 1$, then $CP\_int(n,x_1,\delta) \cap [a,b] = [a,b]$, so the max length of $CP\_int(n,x,\delta) \cap [a,b]$ is $b - a$ (achieved when $x = x_1$). We assume $x_1 \leq x_2 + 1$ from now on.

We want to prove the max length of $CP\_int(n,x,\delta) \cap [a,b]$ is achieved when x is one of the following five values: $x_1, x_1+1, x_2, x_2+1, \lfloor \frac{n}{2} \rfloor$.

Using Lemma 4, we only need to consider $x_1 \leq x \leq x_2 + 1$ because when $x < x_1$ (or $x > x_2 + 1$), the length of $CP\_int(n,x,\delta) \cap [a,b]$ is smaller than $CP\_int(n,x_1,\delta) \cap [a,b]$ (or $CP\_int(n,x_2+1,\delta) \cap [a,b]$).

(1) If $CP\_int(n,x_1,\delta).right \geq b$, then $CP\_int(n,x_1,\delta) \cap [a,b] = [a,b]$, so the max length is $b - a$ and it is achieved when $x = x_1$.

Algorithm 3 Function $validate(n, \varepsilon, \delta, a, b)$

```
Input: The simulation number $n$, precision $\varepsilon$, confidence $\delta$, interval $[a, b]$
Output: "True" ("False") if validation passed (failed)
1: if $CP\_int(n, 0, \delta).right \geq b \vee CP\_int(n, n, \delta).left \leq a$ then
2:     return $b - a \leq 2 \cdot \varepsilon$
3: Find $x_1$ such that $CP\_int(n, x_1, \delta).left \leq a$ and $CP\_int(n, x_1 + 1, \delta).left > a$   # using binary search
4: Find $x_2$ such that $CP\_int(n, x_2, \delta).right < b$ and $CP\_int(n, x_2 + 1, \delta).right \geq b$   # using binary search
5: if $x_1 > x_2 + 1$ then
6:     return $b - a \leq 2 \cdot \varepsilon$
7: for $x$ from $x_1$ to $x_2 + 1$ do
8:     if $CP\_int(n, x, \delta) \cap [a, b] > 2 \cdot \varepsilon$ then
9:         return False
10: return True
```

Algorithm 4 Function $estimate(n, \varepsilon, \delta, a, b)$

```
Input: The simulation number $n$, precision $\varepsilon$, confidence $\delta$, interval $[a, b]$
Output: The estimation result $\hat{p}$
1: if $n == 0$ then
2:     return $\frac{a+b}{2}$
3: if $validate(n, \varepsilon, \delta, a, b)$ then
4:     # validation passed
5:     Perform $n$ simulations and find $x$ successes
6:     $[lb, ub] = CP\_int(n, x, \delta)$
7:     if $ub \leq a$ then
8:         return $a$
9:     else if $lb \geq b$ then
10:        return $b$
11:    else
12:        return $\frac{max(lb,a)+min(ub,b)}{2}$
13: else
14:    # validation failed
15:    $n_1 = \lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$
16:    Perform $n_1$ simulations and find $x$ successes
17:    return $\frac{x}{n_1}$
```

(2.1) If $CP\_int(n, x_1, \delta).right < b$ and $CP\_int(n, x_1 + 1, \delta).right \geq b$, then $x_1 = x_2$ and the max length is achieved when x is x_1 or $x_1 + 1$.

(2.2) If $CP\_int(n, x_1, \delta).right < b$ and $CP\_int(n, x_1 + 1, \delta).right < b$, then $CP\_int(n, x_2, \delta).left > a$. Therefore, when $x_1 + 1 \leq x \leq x_2$, $CP\_int(n, x, \delta) \subseteq [a, b]$ so $CP\_int(n, x, \delta) \cap [a, b] = CP\_int(n, x, \delta)$. Using Hypothesis 1, the max length when $x_1 + 1 \leq x \leq x_2$ is achieved when x is $x_1 + 1$ (if $x_1 + 1 \geq \frac{n}{2}$), x_2 (if $x_2 \leq \frac{n}{2}$) or $\lfloor \frac{n}{2} \rfloor$ (if $x_1 + 1 < \frac{n}{2} < x_2$). As we have mentioned before, we only need to consider $x_1 \leq x \leq x_2 + 1$, so the max length is achieved when x is one of the following five values: $x_1, x_1 + 1, x_2, x_2 + 1, \lfloor \frac{n}{2} \rfloor$. □

We have verified Hypothesis 1 for commonly-used confidence δ ($0.001, 0.002, \cdots, 0.050$ and $0.01, 0.02, \cdots, 0.99$) and all simulation numbers $n \leq 20000$. Moreover, we find that the confidence interval length's derivative with respect to x is 0 at $\frac{n}{2}$ [38]. Therefore, we believe that Hypothesis 1 is correct. However, we cannot find an existing proof for it, nor can we formally prove it (the derivative being 0 is a necessary but not sufficient condition). To ensure the soundness of our algorithms, we need a validation procedure (shown in Algorithm 3) to double-check whether the length of $CP\_int(n, x, \delta) \cap [a, b]$ is indeed no larger than $2 \cdot \varepsilon$ for all $x = 0, 1, \cdots, n$.

Algorithm 3 is actually a heavyweight version of Algorithm 2. It uses the same method to calculate x_1 and x_2 (lines 3-4 of Algorithm 3 are the same with lines 3-4 of Algorithm 2). The only difference is that Algorithm 3 enumerates all $x_1 \leq x \leq x_2 + 1$ (lines 7-9, Algorithm 3) while Algorithm 2 only considers five of them (line 5, Algorithm 2). Therefore, Algorithm 3 does not rely on Hypothesis 1, and its soundness proof is a small subset of the proof for Algorithm 2.

Theorem 2 (Soundness of Algorithm 3): For any simulation number $n > 0$, any confidence and precision $0 < \delta, \varepsilon < 1$ and $0 \leq a < b \leq 1$, Algorithm 3 can return the correct result (i.e. return "True" if the length of $CP\_int(n, x, \delta) \cap [a, b]$ is no larger than $2 \cdot \varepsilon$ for all $x = 0, 1, \cdots, n$ and return "False" otherwise).

Proof: If $CP\_int(n, 0, \delta).right \geq b$, then the max length of $CP\_int(n, x, \delta) \cap [a, b]$ $(x = 0, 1, \cdots, n)$ is $b - a$, so we only need to check if $b - a \leq 2 \cdot \varepsilon$ holds. The case when $CP\_int(n, n, \delta).left \leq a$ is similar.

If $CP\_int(n, 0, \delta).right < b$ and $CP\_int(n, n, \delta).left > a$, the proof is the same with the second and fourth paragraphs in the proof of Theorem 1. □

After obtaining the simulation number n through Algorithm 1, the estimation method is shown in Algorithm 4.

The $n = 0$ case (lines 1-2, Algorithm 4) is only used when $b - a \leq 2 \cdot \varepsilon$ (see lines 1-2 of Algorithm 1). When $n > 0$, we first run the validation procedure (Algorithm 3) to check if for all $x = 0, 1, \cdots, n$, the length of $CP\_int(n, x, \delta) \cap [a, b]$ is no larger than $2 \cdot \varepsilon$ (line 3, Algorithm 4). If the validation is passed, then we perform n simulations and calculate the Clopper-Pearson confidence interval using the simulation results (lines 5-6, Algorithm 4). If the intersection of the confidence interval and $[a, b]$ is not empty, then we use the midpoint of the intersection as the estimation result (line 12, Algorithm 4). We use a (b) as the estimation result if the confidence interval is totally on the left (right) of $[a, b]$ (lines 7-10, Algorithm 4). If the validation is failed, we simply use the Okamoto bound for estimation (lines 15-17, Algorithm 4).

The main idea of the soundness proof for Algorithm 4 is as follows. According to soundness of the Clopper-Pearson confidence interval (Lemma 2 in Section II-B), we have $Pr(p \in CP\_int(n, x, \delta) \cap [a, b]) \geq 1 - \delta$ (we already know $p \in [a, b]$ in Problem 2). Moreover, the validation procedure (Algorithm 3) ensures that the length of $CP\_int(n, x, \delta) \cap [a, b]$ is no larger than $2 \cdot \varepsilon$ (Theorem 2). Because we choose the

Algorithm 5 Simple estimation algorithm

```
Input: User-defined precision $\varepsilon$ and confidence $\delta$, interval $[a,b]$
Output: The estimation result $\hat{p}$
1: $n = interval\_sensitive\_bound(\varepsilon, \delta, a, b)$
2: return $estimate(n, \varepsilon, \delta, a, b)$
```

midpoint of $CP\_int(n,x,\delta) \cap [a,b]$ as the estimation result $\hat{p}$, we have $CP\_int(n,x,\delta) \cap [a,b] \subseteq [\hat{p}-\varepsilon, \hat{p}+\varepsilon]$. Therefore, we have $Pr(p \in [\hat{p}-\varepsilon,\hat{p}+\varepsilon]) \geq Pr(p \in CP\_int(n,x,\delta) \cap [a,b]) \geq 1-\delta$. Formally, we have the following theorem.

Theorem 3 (Soundness of Algorithm 4): Denote p as the probability to be estimated. In Algorithm 4, for any confidence and precision $0 < \delta, \varepsilon < 1$, simulation number $n > 0$ and $0 \leq a < b \leq 1$, if $p \in [a,b]$, then $Pr(|\hat{p} - p| > \varepsilon) \leq \delta$.

Proof: If the validation is passed, then using Theorem 2, the length of $CP\_int(n,x,\delta) \cap [a,b]$ is no larger than $2 \cdot \varepsilon$ for all $x = 0, 1, \cdots, n$. Using Lemma 2, we have $Pr(p \in CP\_int(n,x,\delta)) \geq 1-\delta$. As $p \in [a,b]$, we have $Pr(p \in CP\_int(n,x,\delta) \cap [a,b]) \geq 1-\delta$. If $CP\_int(n,x,\delta) \cap [a,b] \neq \emptyset$, as the length of $CP\_int(n,x,\delta) \cap [a,b]$ is no larger than $2 \cdot \varepsilon$, it is included in $[\hat{p}-\varepsilon, \hat{p}+\varepsilon]$ (as we choose $\hat{p}$ as the midpoint of the intersection, see line 12 of Algorithm 4). If $CP\_int(n,x,\delta) \cap [a,b] = \emptyset$, it is also included in $[\hat{p}-\varepsilon,\hat{p}+\varepsilon]$ (in our algorithm we choose $\hat{p}$ to be a or b, but actually it can be chosen arbitrarily as the empty set is included in any set). Therefore, $CP\_int(n,x,\delta) \cap [a,b] \subseteq [\hat{p}-\varepsilon,\hat{p}+\varepsilon]$, thus $Pr(p \in [\hat{p}-\varepsilon,\hat{p}+\varepsilon]) \geq 1-\delta$, which is equivalent to $Pr(|\hat{p}-p| > \varepsilon) \leq \delta$.

If the validation is failed, we directly use the Okamoto bound $\lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$ for estimation. Using Lemma 1, we also have $Pr(|\hat{p}-p| > \varepsilon) \leq \delta$. □

Finally, as shown in Algorithm 5, to solve Problem 2, we first use Algorithm 1 to calculate the simulation number n and then use Algorithm 4 to make the estimation. As a special case, if we set $[a,b] = [0,1]$, we can obtain a simple estimation algorithm to solve Problem 1.

Theorem 4 (Soundness of Algorithm 5): Denote p as the probability to be estimated. In Algorithm 5, for any confidence and precision $0 < \delta, \varepsilon < 1$ and $0 \leq a < b \leq 1$, if $p \in [a,b]$, then $Pr(|\hat{p}-p| > \varepsilon) \leq \delta$.

Proof: In line 1 of Algorithm 5, if $n = 0$, according to the $interval\_sensitive\_bound$ function (Algorithm 1), it can only happen if $b - a \leq 2 \cdot \varepsilon$. As $p \in [a,b]$, we have $|p - \frac{a+b}{2}| \leq \frac{b-a}{2} \leq \varepsilon$. Therefore, we can directly estimate $\hat{p} = \frac{a+b}{2}$ without any simulation (line 2, Algorithm 4).

If $n > 0$, using Theorem 3, Algorithm 5 is sound. □

As a special case of Theorem 4, when $[a,b] = [0,1]$, Algorithm 5 is sound (because we always have $p \in [0,1]$).

Thanks to the validation procedure (Algorithm 3), the soundness of Theorem 2-4 does not rely on Hypothesis 1 or Theorem 1. If we can prove Hypothesis 1 in the future, then we can remove the validation procedure (i.e. lines 3 and 13-17 of Algorithm 4), and the soundness proof of the new algorithm is almost the same with the current one (use Theorem 1 to replace Theorem 2). The running time will not change much because in the current algorithm, the validation procedure is invoked only once and takes very little time.

In the worst case, the simulation number of Algorithm 5 is equal to the Okamoto bound (if the validation is failed). However, although Hypothesis 1 is unproven, we have not found a single experiment in which the validation failed. Therefore, the performance of Algorithm 5 is much better than the Okamoto bound in practice.

Example 1: Suppose $\varepsilon = 0.01, \delta = 0.05, p = 0.21, [a,b] = [0.2, 0.3]$. We use Algorithm 5 to solve Problem 2. In line 1 of Algorithm 5, we call Algorithm 1 by $interval\_sensitive\_bound(0.01, 0.05, 0.2, 0.3)$. Algorithm 1 performs binary search with the initial lower bound of 0 and the initial upper bound of $\lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil = 18445$. In the first search, $mid = \lfloor 18445/2 \rfloor = 9222$, and we call Algorithm 2 by $max\_length(9222, 0.05, 0.2, 0.3)$ to calculate the max length of $CP\_int(9222, x, 0.05) \cap [0.2, 0.3]$. In lines 3 and 4 of Algorithm 2, we obtain $x_1 = 1920$ and $x_2 = 2680$ using binary search. In line 5 of Algorithm 2, we calculate the length of $CP\_int(9222, x, 0.05) \cap [0.2, 0.3]$ for five different x: $x_1 = 1920, x_1 + 1 = 1921, x_2 = 2680, x_2 + 1 = 2681, \lfloor \frac{9222}{2} \rfloor = 4611$. The max length 0.01864 (achieved when $x = x_2 = 2680$) is returned. In line 7 of Algorithm 1, we find $0.01864 \leq 2 \cdot \varepsilon = 0.02$, therefore, the new upper bound $high$ becomes 9222, and in the second search, $mid = 4611$. After 15 searches, Algorithm 1 returns $high = 8005$ as the simulation number n in line 1 of Algorithm 5. Then we call Algorithm 4 by $estimate(8005, 0.01, 0.05, 0.2, 0.3)$ in line 2 of Algorithm 5 to make the final estimation. In line 3 of Algorithm 4, we call Algorithm 3 by $validate(8005, 0.01, 0.05, 0.2, 0.3)$. In lines 3 and 4 of Algorithm 3, we obtain $x_1 = 1671$ and $x_2 = 2320$ using binary search. In lines 7-9 of Algorithm 3, we consider all $1671 \leq x \leq 2321$ and check whether the length of $CP\_int(8005, x, 0.05) \cap [0.2, 0.3]$ is no larger than $2 \cdot \varepsilon = 0.02$. The validation is passed, so Algorithm 3 returns "True". In line 5 of Algorithm 4, we perform 8005 simulations and find $x = 1634$ successes, therefore, the corresponding confidence interval in line 6 is $[lb, ub] = [0.19534, 0.21312]$. Finally, in line 12 of Algorithm 4, we return the midpoint of $[0.19534, 0.21312] \cap [0.2, 0.3] = [0.2, 0.21312]$, that is, 0.20656, as the final estimation result $\hat{p}$. The estimation error $|\hat{p} - p| = 0.00344 \leq \varepsilon = 0.01$.

Example 2: Suppose $\varepsilon = \delta = 0.01$, then the Okamoto bound is $\lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil = 26492$ regardless of $[a,b]$. In Algorithm 5, if we set $[a,b] = [0,1]$, then the simulation number n is 16684, which is much lower than the Okamoto bound. If we set $[a,b]$ to be $[0, 0.1]$, $[0, 0.05]$, $[0.05, 0.1]$, $[0.4, 0.5]$, $[0.45, 0.55]$, $[0.2, 0.3]$ and $[0.9, 1]$, the simulation number is 5508, 2592, 5508, 16677, 16684, 13755 and 5508 respectively.

B. Adaptive Estimation Algorithm

As shown in Example 2 above, if we know in advance that p is within some interval $[a,b]$, we may further reduce the simulation number compared with setting $[a,b] = [0,1]$ in Algorithm 5. Specifically, the farther away $[a,b]$ is from 0.5, the lower the simulation number is. When $[a,b]$ is near 0.5 or includes 0.5, the simulation number is almost the same with that when $[a,b]=[0,1]$. However, usually we cannot know the range of p before we actually perform some simulations. Therefore, a straightforward idea is to perform some simulations first to obtain a confidence interval of p.

Example 3: Suppose $\varepsilon=\delta=0.01$, we need to perform 16684 simulations if we directly use Algorithm 5 with $[a,b]=[0,1]$. Here we consider another method: *we perform $n=200$ simulations first to get a confidence interval of p, then use the obtained interval as $[a,b]$ in Algorithm 5.*

Case 1: Suppose $p=0.1$, we perform 200 simulations and find 24 successes, then the Clopper-Pearson confidence interval is $[0.057, 0.212]$ (99.9% confidence level, $\delta'=0.001$). Set $[a,b]=[0.057,0.212]$ and use Algorithm 5 (precision is $\varepsilon=0.01$, confidence is $\delta-\delta'=0.009$), only $n=11101$ simulations are needed. In total, we need 200+11101=11301 simulations, which is 32% less than directly using Algorithm 5 with $[a,b]=[0,1]$.

Case 2: Suppose $p=0.4$, we perform 200 simulations and find 76 successes, then the Clopper-Pearson confidence interval is $[0.271, 0.498]$ (99.9% confidence level, $\delta'=0.001$). Set $[a,b]=[0.271,0.498]$ and use Algorithm 5 (precision is $\varepsilon=0.01$, confidence is $\delta-\delta'=0.009$), another $n=17144$ simulations are needed. In total, we need 200+17144=17344 simulations, which is 4% more than directly using Algorithm 5 with $[a,b]=[0,1]$.

From Example 3 we know that when p is near 0 or 1, we can significantly reduce the simulation number (compared with directly using Algorithm 5 with $[a,b]=[0,1]$) by performing some simulations first to obtain a confidence interval of p. However, if p is near 0.5, the obtained confidence interval is also near 0.5 (or includes 0.5), and the loss outweighs the gain. Specifically, we need to answer the following two questions.

Question 1: How to choose between Algorithm 5 (with $[a,b]=[0,1]$) and the method in Example 3?

Question 2: When using the method in Example 3, how to choose the simulation number n?

As mentioned before, the method in Example 3 is more suitable for p near 0 and 1, and Algorithm 5 with $[a,b]=[0,1]$ is more suitable for p near 0.5. Therefore, the main challenge for Question 1 is that we cannot know the value of p in advance. For Question 2, although using $n=200$ simulations works well in Case 1 of Example 3, it is somewhat arbitrary and may not work well for other ε,δ,p. Therefore, the main challenge is to automatically select a proper simulation number.

We propose the *adaptive estimation algorithm* as shown in Algorithm 6. We first calculate the simulation number n_1 needed if we directly use Algorithm 5 with $[a,b]=[0,1]$ (line 1, Algorithm 6). Then we perform about $1\%\cdot n_1$ *pre-simulations* to obtain a rough estimation p_1 (lines 3-6, Algorithm 6). In the strategy analysis part (lines 7-15, Algorithm 6), we answer the above two questions. Specifically, we assume we perform $\lceil 0.01\cdot n_1\rceil, \lceil 0.02\cdot n_1\rceil, \cdots, \lceil 0.20\cdot n_1\rceil$ simulations to calculate the confidence interval respectively (line 8, Algorithm 6). For each n_3 in *n_list*, we suppose we obtain about $n_3\cdot p_1$ successes after n_3 simulations, and we calculate the corresponding Clopper-Pearson confidence interval (lines 11-13, Algorithm 6). Then we use the obtained interval as $[a,b]$ in Algorithm 5 to determine the final simulation number. The total cost is the sum of these two simulation numbers (line 14, Algorithm 6). To summary, we estimate the total cost of choosing different simulation numbers n_3 for calculating the confidence interval. Then we compare these estimated costs with n_1 (line 15, Algorithm 6), if all of them are higher than n_1, we consider it is better to directly use the Algorithm 5 with $[a,b]=[0,1]$ (lines 16-17, Algorithm 6). Otherwise, we select the simulation number n_4 with the smallest cost (line 20, Algorithm 6) and use the method in Example 3 (lines 21-25, Algorithm 6). Specifically, we perform n_4 simulations and find x successes, then we compute the Clopper-Pearson confidence interval of p (lines 21-22, Algorithm 6). Finally, we use the obtained interval as $[a,b]$ in Algorithm 5 (lines 24-25, Algorithm 6).

Algorithm 6 Adaptive estimation algorithm

```
Input: User-defined precision $\varepsilon$ and confidence $\delta$
Output: The estimation result $\hat{p}$
1:  $n_1 = interval\_sensitive\_bound(\varepsilon, \delta, 0, 1)$
2:  $\delta' = 0.05\cdot\delta$
3:  # pre-simulation
4:  $n_2 = max(min(\lceil 0.01\cdot n_1\rceil, 100), 10)$
5:  Perform $n_2$ simulations and find $x$ successes
6:  $p_1 = \frac{x}{n_2}$
7:  # strategy analysis
8:  $n\_list = [\lceil 0.01\cdot n_1\rceil, \lceil 0.02\cdot n_1\rceil, \cdots, \lceil 0.20\cdot n_1\rceil]$
9:  $cost = []$
10: for $i$ in $range(20)$ do
11:     $n_3 = n\_list[i]$
12:     $x = round(n_3\cdot p_1)$
13:     $[lb, ub] = CP\_int(n_3, x, \delta')$
14:     $cost.append(n_3 +$
        $interval\_sensitive\_bound(\varepsilon, \frac{\delta-\delta'}{1-\delta'}, lb, ub))$
15: if $min(cost) \geq n_1$ then
16:     # directly use Algorithm 5
17:     return $estimate(n_1, \varepsilon, \delta, 0, 1)$
18: else
19:     # interval estimation
20:     $n_4 = n\_list[cost.min\_index]$
21:     Perform $n_4$ simulations and find $x$ successes
22:     $[lb, ub] = CP\_int(n_4, x, \delta')$
23:     # calculate interval-sensitive bound
24:     $n_5 = interval\_sensitive\_bound(\varepsilon, \frac{\delta-\delta'}{1-\delta'}, lb, ub)$
25:     return $estimate(n_5, \varepsilon, \frac{\delta-\delta'}{1-\delta'}, lb, ub)$
```

Example 4: Suppose $\varepsilon=\delta=0.01$, we need to perform $n_1=16684$ simulations if we directly use Algorithm 5 with $[a,b]=[0,1]$.

Case 1: Suppose $p=0.1$. In the pre-simulation part, we perform $n_2=100$ simulations and find $x=8$ successes, so $p_1=0.08$. In the strategy analysis part, we find $n_3=\lceil 0.06\cdot n_1\rceil=1002$ has the smallest cost 1002+6335=7337 (6335 is obtained by calling the *interval_sensitive_bound* function in line 14 of Algorithm 6), which is smaller than $n_1=$

16684. Therefore, we choose the second estimation method (lines 19-25, Algorithm 6) and set $n_4 = 1002$. We perform 1002 simulations and find 96 successes, so the Clopper-Pearson confidence interval is $[0.066, 0.132]$. Then we use Algorithm 5 with $[a, b] = [0.066, 0.132]$ and obtain the simulation number $n_5 = 7292$. Finally, we perform 7292 simulations and obtain the estimation result 0.1024. We perform 100+1002+7292=8394 simulations in total, which is 50% less than n_1. Compared with Case 1 in Example 3, here we use 1002 simulations instead of 200 simulations to compute the confidence interval, thus obtaining a narrower interval and leading to a lower total simulation number (8394 compared to 11301).

Case 2: Suppose $p = 0.4$. In the pre-simulation part, we perform $n_2 = 100$ simulations and find $x = 43$ successes, so $p_1 = 0.43$. In the strategy analysis part, we find all n_3 have higher cost than $n_1 = 16684$. Therefore, we choose the first estimation method (lines 16-17, Algorithm 6). We perform $n_1 = 16684$ simulations and obtain the estimation result 0.4010. We perform 100+16684=16784 simulations in total, which is 0.6% higher than n_1. Compared with Case 2 in Example 3, here we correctly choose the first estimation method (lines 16-17, Algorithm 6) rather than the second estimation method (lines 19-25, Algorithm 6), thus avoiding many unnecessary simulations and leading to a lower total simulation number (16784 compared to 17344).

In summary, Algorithm 6 is the combination of two estimation methods. The first method in lines 16-17 is suitable for p near 0.5, and the second method in lines 19-25 is suitable for p near 0 or 1. We use the pre-simulation part and the strategy analysis part (lines 3-15, Algorithm 6) to decide which method to use. In the pre-simulation part, we perform a small number of simulations to obtain a rough estimation p_1. In the strategy analysis part, we conduct numerical calculation based on p_1 without actually performing simulations. Usually p_1 is close to p, so we can choose the correct estimation method in most cases.

To prove the soundness of Algorithm 6, we only need to prove the soundness of the two estimation methods used. The first method (lines 16-17, Algorithm 6) is directly using Algorithm 5 (n_1 is calculated in line 1 of Algorithm 6), so its soundness is guaranteed by Theorem 4. For the second method (lines 19-25, Algorithm 6), we first perform n_4 simulations to obtain a confidence interval $[lb, ub]$, then we call Algorithm 5 with $[a, b] = [lb, ub]$. According to soundness of the Clopper-Pearson confidence interval (Lemma 2 in Section II-B), we have $Pr(p \in [lb, ub]) \geq 1 - \delta'$. Moreover, according to Theorem 4, if $p \in [lb, ub]$, we have $Pr(|p - \hat{p}| > \varepsilon) \leq \frac{\delta-\delta'}{1-\delta'}$. Using the above two inequalities, we can prove that the overall error rate is no larger than δ by the law of total probability: $Pr(|p - \hat{p}| > \varepsilon) = Pr(|p - \hat{p}| > \varepsilon | p \in [lb, ub]) \cdot Pr(p \in [lb, ub]) + Pr(|p - \hat{p}| > \varepsilon | p \notin [lb, ub]) \cdot Pr(p \notin [lb, ub])$. Formally, we have the following theorem.

Theorem 5 (Soundness of Algorithm 6): Denote p as the probability to be estimated. In Algorithm 6, for any confidence and precision $0 < \delta, \varepsilon < 1$, $Pr(|\hat{p} - p| > \varepsilon) \leq \delta$.

Proof: There are two estimation methods in Algorithm 6.

The first method is directly using Algorithm 5 with $[a, b] = [0, 1]$ (lines 16-17, Algorithm 6), which has been proved to be sound in Theorem 4.

The second method is performing interval estimation first to obtain a confidence interval $[lb, ub]$ (lines 19-22, Algorithm 6), then calling Algorithm 5 with $[a, b] = [lb, ub]$ (lines 23-25, Algorithm 6). Using Lemma 2, we have $Pr(p \in [lb, ub]) \geq 1 - \delta'$. Using Theorem 4, we have $Pr(|p - \hat{p}| > \varepsilon | p \in [lb, ub]) \leq \frac{\delta-\delta'}{1-\delta'}$. Note that

$$
\begin{aligned}
&Pr(|p - \hat{p}| > \varepsilon) \\
&\quad = Pr(|p - \hat{p}| > \varepsilon, p \in [lb, ub]) \\
&\qquad + Pr(|p - \hat{p}| > \varepsilon, p \notin [lb, ub]) \\
&\quad = Pr(|p - \hat{p}| > \varepsilon | p \in [lb, ub]) \cdot Pr(p \in [lb, ub]) \\
&\qquad + Pr(|p - \hat{p}| > \varepsilon | p \notin [lb, ub]) \cdot Pr(p \notin [lb, ub]) \\
&\quad \leq \frac{\delta - \delta'}{1 - \delta'} \cdot Pr(p \in [lb, ub]) + 1 \cdot (1 - Pr(p \in [lb, ub])) \\
&\quad = 1 - \frac{1 - \delta}{1 - \delta'} \cdot Pr(p \in [lb, ub]) \\
&\quad \leq 1 - \frac{1 - \delta}{1 - \delta'} \cdot (1 - \delta') = \delta
\end{aligned}
$$

Therefore, the second method is also sound.

In Algorithm 6, we use the pre-simulation part and the strategy analysis part to select the proper method for estimation. As we have proved that both methods are sound, Algorithm 6 is also sound. □

Note that in the second method, both the interval estimation part (lines 19-22, Algorithm 6) and the final estimation part (lines 23-25, Algorithm 6) introduce some uncertainty. As we require that the total error rate is no larger than δ, we allocate δ' for the interval estimation part and $\delta - \delta'$ (further refined to $\frac{\delta-\delta'}{1-\delta'}$ as shown in the above proof) for the final estimation part. We discuss the selection of δ' (and other hyperparameters) in Section VI-A.

C. Complexity Analysis of Extra Cost

Apart from simulations, there is some extra cost in our algorithms, mainly due to calculating Clopper-Pearson confidence intervals. In practice, the time cost of calculating a Clopper-Pearson confidence interval can be regarded as a constant. For example, on our laptop with an Intel Core i5-13500H CPU, calculating one Clopper-Pearson confidence interval takes about $4 \cdot 10^{-6}$s. Therefore, we only focus on the number of Clopper-Pearson confidence intervals calculated. Note that calculating a confidence interval involves calculating two sides (i.e. the left endpoint and right endpoint, see Section II-B). However, in our algorithms, sometimes we only need one side of the confidence interval (e.g. $CP\_int(n, 0, \delta).right$ in line 1 of Algorithm 2). In this situation, we count it as calculating 0.5 confidence interval. For better readability, in this subsection we use M to denote the Okamoto bound $\lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$.

In line 1 of Algorithm 2, we calculate at least 0.5 confidence interval (if $CP\_int(n, 0, \delta).right \geq b$ is satisfied then there is no need to check $CP\_int(n, n, \delta).left \leq a$) and at most 1 confidence interval. In line 3 of Algorithm 2, we use binary search to find x_1 with the initial lower bound of 0 and the initial upper bound of $n-1$. Note that there must exist one x_1 satisfying the given property, therefore, it involves at least 1 search (0 search if $n=1$) and at most $\lfloor \log_2 n \rfloor$ searches. Each search calculates 0.5 or 1 confidence interval (if $CP\_int(n, x_1, \delta).left \leq a$ is not satisfied then there is no need to check $CP\_int(n, x_1+1, \delta).left > a$). The situation for line 4 of Algorithm 2 is the same thus is omitted here. In line 5 of Algorithm 2, we calculate 5 confidence intervals. Therefore, in Algorithm 2, we calculate at least 0.5 confidence interval (if $CP\_int(n, 0, \delta).right \geq b$ is satisfied in line 1) and at most $2 \cdot \lfloor \log_2 n \rfloor + 6$ confidence intervals.

In Algorithm 1, if $b - a \leq 2 \cdot \varepsilon$, then we can terminate immediately without calculating any confidence interval. Otherwise, we perform binary search to calculate the simulation number needed. The initial lower bound is 0, and the initial upper bound is the Okamoto bound M. In the binary search, Algorithm 2 is invoked at least $\lfloor \log_2 M \rfloor$ times and at most $\lceil \log_2 M \rceil$ times, each with different mid as the parameter n in Algorithm 2. Note that $mid \leq M-1$, therefore, in Algorithm 1, we calculates at most $\lceil \log_2 M \rceil \cdot (2 \cdot \lfloor \log_2 (M-1) \rfloor + 6)$ confidence intervals.

The analysis of Algorithm 3 is very similar to Algorithm 2. In line 1 of Algorithm 3, we calculate at least 0.5 confidence interval and at most 1 confidence interval. In line 3 (or 4) of Algorithm 3, we perform at least 1 search (0 search if $n=1$) and at most $\lfloor \log_2 n \rfloor$ searches, and each search calculates 0.5 or 1 confidence interval. In lines 7-9 of Algorithm 3, we calculate at most $x_2 + 2 - x_1$ confidence intervals. Note that $x_1 \geq 0$ and $x_2 + 1 \leq n$, so $x_2 + 2 - x_1 \leq n + 1$. Therefore, in Algorithm 3, we calculate at least 0.5 confidence interval and at most $2 \cdot \lfloor \log_2 n \rfloor + n + 2$ confidence intervals.

In Algorithm 4, if $n = 0$, then we can terminate immediately without calculating any confidence interval. Otherwise, in line 3 of Algorithm 4, we invoke Algorithm 3 once. In line 6 of Algorithm 4, we calculate 1 confidence interval. Therefore, in Algorithm 4, we calculate at most $2 \cdot \lfloor \log_2 n \rfloor + n + 3$ confidence intervals.

In Algorithm 5 (the simple estimation algorithm), we first invoke Algorithm 1 to obtain the simulation number n, then invoke Algorithm 4 using n. According to Algorithm 1, we have $n \leq M$, therefore, in Algorithm 5, we calculates at most $\lceil \log_2 M \rceil \cdot (2 \cdot \lfloor \log_2 (M-1) \rfloor + 6) + 2 \cdot \lfloor \log_2 M \rfloor + M + 3$ confidence intervals. When $b - a \leq 2 \cdot \varepsilon$, Algorithm 1 directly returns $n = 0$, and Algorithm 4 directly returns $\frac{a+b}{2}$, so Algorithm 5 does not calculate any confidence interval in this situation.

In Algorithm 6 (the adaptive estimation algorithm), we invoke Algorithm 1 once in line 1. In the strategy analysis part, line 13 calculates 1 confidence interval, and line 14 invokes Algorithm 1 once. Note that lines 13 and 14 are both executed 20 times. If $min(cost) \geq n_1$, we invoke Algorithm 4 once in line 17. Otherwise, line 22 calculates 1 confidence interval, line 24 invokes Algorithm 1 once, and line 25 invokes Algorithm 4 once. Note that in lines 14 and 24, we invoke Algorithm 1 with $\frac{\delta - \delta'}{1 - \delta'} = \frac{0.95\delta}{1 - 0.05\delta}$ rather than δ. We denote the corresponding Okamoto bound $\lceil \frac{1}{2\varepsilon^2} \ln(\frac{2 \cdot (1 - 0.05\delta)}{0.95\delta}) \rceil$ by M_1, which is slightly larger than M. Note that $n_1 \leq M$ and $n_5 \leq M_1$, therefore, in Algorithm 6, we calculate at most $21 \cdot \lceil \log_2 M_1 \rceil \cdot (2 \cdot \lfloor \log_2 (M_1 - 1) \rfloor + 6) + \lceil \log_2 M \rceil \cdot (2 \cdot \lfloor \log_2 (M-1) \rfloor + 6) + 2 \cdot \lfloor \log_2 M_1 \rfloor + M_1 + 24$ confidence intervals.

In practice, the simulation number obtained through Algorithm 1 is much smaller than the Okamoto bound, so the actual number of confidence intervals calculated will be much smaller than the above theoretical analysis. In Section V-C, we will evaluate the extra time cost of our algorithms experimentally.

V. Evaluation

In this section we evaluate our simple estimation algorithm (Algorithm 5 with $[a, b] = [0, 1]$) and our adaptive estimation algorithm (Algorithm 6). We use the Okamoto bound and the Massart bound-based sequential algorithm [19] (see Section III-B) as the baseline methods. Other existing methods are either unreliable (i.e. methods mentioned in Section III-A other than the Okamoto bound) or infeasible to run (i.e. [21] in Section III-B).

Our evaluations include *numerical experiments* and *case studies*. In numerical experiments (Section V-A), we do not consider concrete models, thus can efficiently and comprehensively evaluate the theoretical performance of algorithms (e.g. average simulation numbers). In case studies (Section V-B), we consider concrete models and different properties for verification, thus can evaluate the efficiency of algorithms in real-world scenarios. Finally, we analyze the extra time cost of our algorithms in Section V-C.

For all experiments, we consider the following combinations of precision and confidence (ε, δ): (0.1, 0.1), (0.05, 0.05), (0.05, 0.01), (0.01, 0.05), (0.01, 0.01), (0.005, 0.005). These combinations can cover most SMC estimation scenarios in practice.

We implement all algorithms in Python and release the code in [38]. Each single experiment is performed on an Intel Xeon E5-2650 v4 CPU.

A. Numerical Experiments

In this subsection, we perform numerical experiments to evaluate our algorithms. We do not consider a concrete model or property to be verified, instead, we manually set the probability p and use the Bernoulli variables generator to replace the simulations. For example, for line 5 of Algorithm 6, we simply generate n_2 independent Bernoulli variables (with parameter p) and use their sum as the success number x. This simplification does not change the distribution of the simulation number. Specifically, if an SMC model has probability p to satisfy the given property, then we can safely replace it by a Bernoulli variable (with parameter p) without changing the expectation simulation number. Therefore, numerical experiments are especially suitable for comparing simulation numbers of different algorithms. Compared with running a concrete SMC model, the time cost of generating Bernoulli variables is much lower, so we can perform large-scale evaluations within a reasonable cost. Moreover, as we can manually set the value of p, we can systematically test the performance of algorithms on different p.

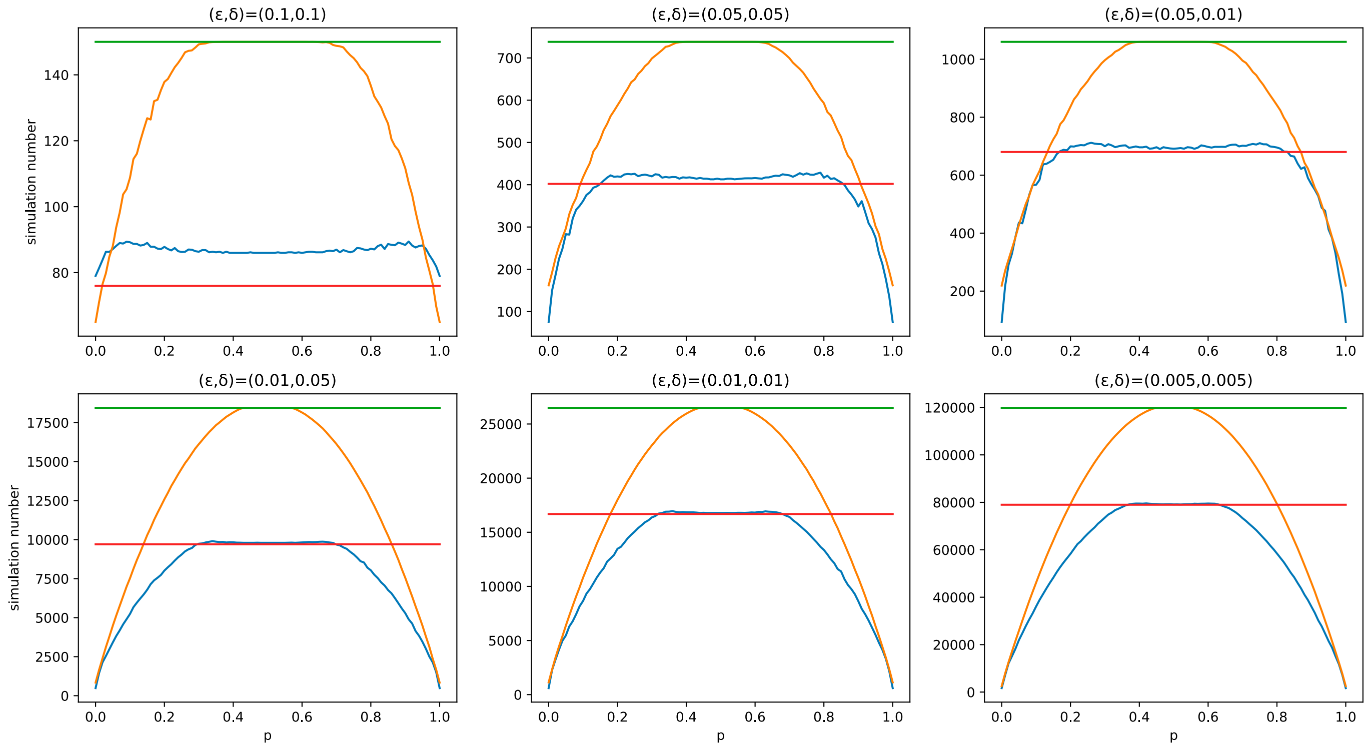

Fig. 2. Average simulation numbers of our simple estimation algorithm (red), our adaptive estimation algorithm (blue), the Okamoto bound (green) and the Massart bound-based sequential algorithm (orange).

1) Average Simulation Number: In this experiment, we consider $p = 0, 0.01, 0.02, \cdots, 0.99, 1$. For each combination of (ε, δ) and p, we repeat each algorithm 100 times and report the average simulation number. The results are shown in Fig. 2.

The Okamoto bound and our simple estimation algorithm are independent with p, so in Fig. 2 they are both horizontal straight lines. In all cases, our simple estimation algorithm uses much fewer simulations than the Okamoto bound. Our adaptive estimation algorithm and the Massart bound-based sequential algorithm are sensitive to p. When p is near 0 or 1, the simulation numbers of both algorithms are significantly lower than when p is near 0.5. When p is near 0.5, the simulation number of the Massart bound-based sequential algorithm is similar to the Okamoto bound, and our adaptive estimation algorithm is similar to our simple estimation algorithm. When p is near 0.5, our adaptive estimation algorithm is much better than the Massart bound-based sequential algorithm, and when p is near 0 or 1, our adaptive estimation algorithm is still consistently better (except for $(\varepsilon, \delta) = (0.1, 0.1)$).

When $(\varepsilon, \delta) = (0.1, 0.1)$, our adaptive estimation algorithm performs worse than our simple estimation algorithm. The reason is that the simulation number is already very small, so the loss ($n_2 = 10$ simulations in line 4 of Algorithm 6) outweighs the gain. In other cases, when p is near 0 or 1, our adaptive estimation algorithm selects the second estimation method (lines 19-25, Algorithm 6) thus can significantly reduce the simulation number compared with our simple estimation algorithm. When p is near 0.5, our adaptive estimation algorithm selects the first estimation method (lines 16-17, Algorithm 6) thus uses n_2 simulations more than our simple estimation algorithm (due to pre-simulations). Between 0 and 0.5, there is a *switch point*, and it is better to use the second (first) estimation method when p is smaller (larger) than it. Similarly, there is a symmetric switch point between 0.5 and 1. We notice that there are two small bounces near the switch points. For example, in the $(\varepsilon, \delta) = (0.05, 0.05)$ case, when p is near 0.2 or 0.8, our adaptive estimation algorithm even needs more simulations than when p is near 0.5. This is because when p is near switch points, any slight underestimation or overestimation of p in the pre-simulation part (lines 3-6, Algorithm 6) may mislead the strategy analysis part (lines 7-15, Algorithm 6) to select the wrong estimation method. For different combinations of (ε, δ), the switch points are also different. For example, the bounces in the $(\varepsilon, \delta) = (0.01, 0.01)$ case occur when p is near 0.35 or 0.65, which is different from the $(\varepsilon, \delta) = (0.05, 0.05)$ case. Therefore, we cannot replace the strategy analysis part by a single threshold.

We present some statistics in Table II. For each algorithm, we calculate its average simulation number under 101 different p (i.e. $p = 0, 0.01, 0.02, \cdots, 0.99, 1$), and the smallest number is in bold. Our adaptive estimation algorithm and the Massart bound-based sequential algorithm are sensitive to p, so we also present the maximum and minimum simulation number among 101 different p. All numbers are rounded to the nearest integer. For ease of comparison, we present the ratio to the Okamoto bound under each simulation number. Because we use the Bernoulli variables generator which takes very little time for each simulation, the average running time in Table II

TABLE II
Simulation Numbers and Average Running Time of Four Algorithms

ε	δ	Okamoto		Simple		Massart				Adaptive			
		Average	Time(s)	Average	Time(s)	Average	Max	Min	Time(s)	Average	Max	Min	Time(s)
0.1	0.1	150 100.0%	0.0005	**76 50.7%**	0.0015	131 87.4%	150 100.0%	65 43.3%	0.0024	87 57.8%	90 59.9%	79 52.7%	0.0266
0.05	0.05	738 100.0%	0.0024	402 54.5%	0.0031	580 78.5%	738 100.0%	162 22.0%	0.0102	**380 51.5%**	428 58.0%	75 10.2%	0.0370
0.05	0.01	1060 100.0%	0.0035	680 64.2%	0.0051	827 78.0%	1060 100.0%	219 20.7%	0.0144	**625 58.9%**	710 67.0%	93 8.8%	0.0386
0.01	0.05	18445 100.0%	0.0647	9701 52.6%	0.0586	12825 69.5%	18445 100.0%	836 4.5%	0.2364	**7662 41.5%**	9903 53.7%	487 2.6%	0.1206
0.01	0.01	26492 100.0%	0.0889	16684 63.0%	0.0921	18342 69.2%	26492 100.0%	1129 4.3%	0.3655	**12881 48.6%**	16908 63.8%	602 2.3%	0.1570
0.005	0.005	119830 100.0%	0.3955	78990 65.9%	0.4395	81325 67.9%	119830 100.0%	2517 2.1%	1.4911	**57893 48.3%**	79553 66.4%	1681 1.4%	0.4237

(the “Time(s)” column) does not reflect the performance of algorithms in the real world, where performing one simulation takes much more time. These results for running time will be used to calculate the *extra time cost* in Section V-C later, and here we only discuss the results for simulation numbers.

In all cases, our simple estimation algorithm is much better than the Okamoto bound. The simulation number is only 50.7% to 65.9% of the Okamoto bound. Although our simple estimation algorithm is not optimized for p near 0 or 1, its average performance is still better than the Massart bound-based sequential algorithm, especially when the simulation number is not very large. In the $(\varepsilon, \delta) = (0.1, 0.1)$ case, our simple estimation algorithm is even the best among all four algorithms.

In most cases, our adaptive estimation algorithm has the best average performance, using only 41.5% to 58.9% of the Okamoto bound simulations. Compared with the Massart bound-based sequential algorithm, our adaptive estimation algorithm is consistently better in terms of average, maximum and minimum simulation numbers (except for the minimum simulation number in the $(\varepsilon, \delta) = (0.1, 0.1)$ case). For example, in the $(\varepsilon, \delta) = (0.01, 0.01)$ case, the average, maximum and minimum simulation numbers of our adaptive estimation algorithm are only 70.2%, 63.8% and 53.3% of the Massart bound-based sequential algorithm.

2) Soundness: We have proved the soundness of our simple estimation algorithm and our adaptive estimation algorithm in Section IV. Here we evaluate their error rates in practice. Formally, if the estimation result $\hat{p}$ satisfies $|\hat{p} - p| > \varepsilon$, then we regard it as an *error*. The error rate of a sound algorithm should not exceed δ (see Problem 1). In this experiment, we consider $p = 0, 0.01, 0.02, \cdots, 0.99, 1$. For each combination of (ε, δ) (except the $(\varepsilon, \delta) = (0.005, 0.005)$ case) and p, we run our two algorithms 100000 times and calculate the average error rate. For the $(\varepsilon, \delta) = (0.005, 0.005)$ case, errors are rarer compared to other cases, so we run each algorithm 1000000 times to reduce the impact of randomness. As an example, when $p = 0.5$ in the $(\varepsilon, \delta) = (0.01, 0.05)$ case, we repeat our adaptive estimation algorithm 100000 times and find 4920 errors (i.e. $|\hat{p} - 0.5| > 0.01$), so the error rate is 0.0492, which is lower than $\delta = 0.05$. Detailed results are shown in Fig. 3. We use the green line to represent the user-defined error rate δ. If an algorithm's error rate is higher than the green line, then it is not sound. On the other hand, if an algorithm's error rate is much lower than the green line, then it is overly conservative and its simulation number is too large. An ideal algorithm has exactly the error rate δ for all $p \in [0, 1]$, which is sound and without any unnecessary simulation. Usually it is hard to obtain such an ideal algorithm, so in practice, we pursue an algorithm that is always below the green line (to guarantee the soundness), but not too far away from it (to avoid being too conservative).

In all cases, the error rates of our two algorithms are lower than the green line (δ), which further supports the soundness of our two algorithms. When p is near 0.5, the error rates of our two algorithms are similar and near δ, which indicates that our two algorithms are close to the ideal algorithm. However, when p is near 0 or 1, both algorithms are far away from the green line, which means they are relatively conservative. Compared with our simple estimation algorithm (blue), our adaptive estimation algorithm (orange) is closer to the ideal algorithm (green) when p is near 0 or 1. It is because our adaptive estimation algorithm uses much fewer simulations when p is near 0 or 1, which reduces the conservativeness.

B. Case Studies

In this subsection, we conduct case studies to evaluate our algorithms. Compared with numerical experiments in Section V-A which only consider the simulation number, here we also consider the running time of different algorithms, thus can reflect the performance of algorithms in practice.

The first case study is the IPv4 zeroconf protocol which can be modeled as a discrete-time Markov chain. The model comes from Example 10.5 of [1] and is shown in Fig. 4.

There are three parameters in this model: r, q and n. At first we are in s_0, then with probability $1 - q$ we will end in s_{n+1}, and with probability q we will jump to s_1. For each s_i $(1 \le i \le n - 1)$, we have probability r to jump to s_{i+1} and probability $1 - r$ to return to s_0. If we reach s_n, then with probability r we will end in s_{n+2}, and with probability $1 - r$ we will go back to s_0. In this model, s_{n+1} is a good state that we want to reach (successfully allocate an IP address), and s_{n+2} is a bad state that we do not want to end in (IP address

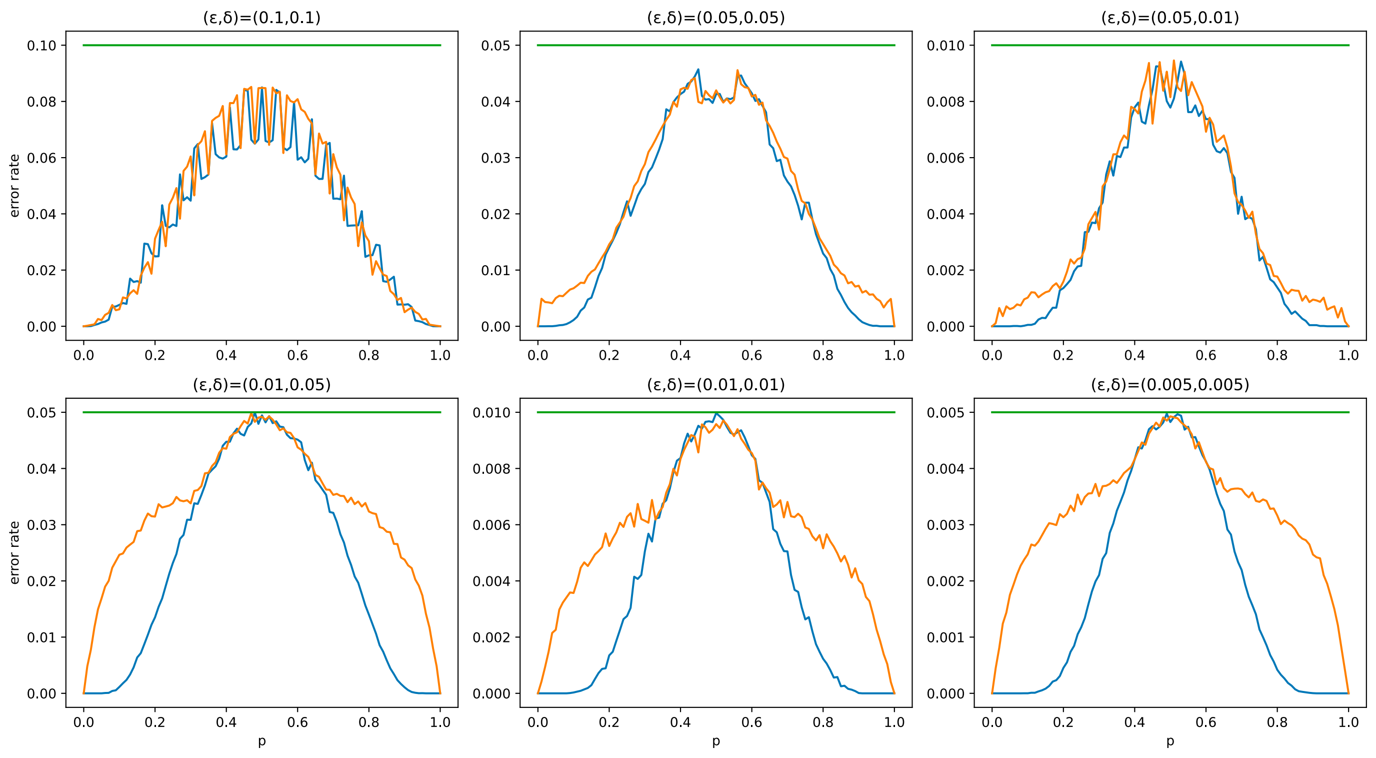

Fig. 3. Average error rates of our simple estimation algorithm (blue) and our adaptive estimation algorithm (orange) compared with δ (green).

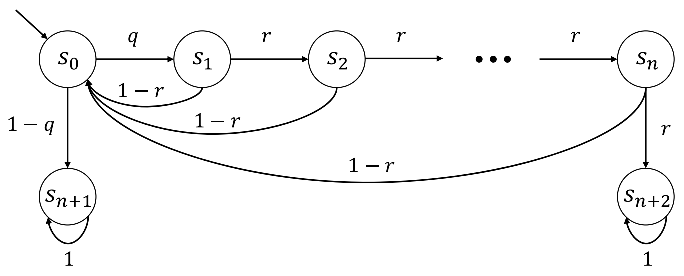

Fig. 4. IPv4 zeroconf protocol model.

TABLE III
PROPERTIES FOR THE IPV4 ZEROCONF PROTOCOL MODEL

	r	q	n	p	Property
P1	0.988564	0.99	1000	0.001	Ends in s_{n+2}
P2	0.990160			0.005	
P3	0.990852			0.01	
P4	0.992489			0.05	
P5	0.993231			0.1	
P6	0.994036			0.2	
P7	0.994572			0.3	
P8	0.995012			0.4	
P9	0.995415			0.5	

collision). We want to estimate the probability p that the system eventually ends in s_{n+2}. In fact, we can calculate the exact expression of p: $p = \frac{q \cdot r^n}{1-q+q \cdot r^n}$. Therefore, we manually select nine combinations of r, q, n such that the corresponding values of p are 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4 and 0.5 respectively. These properties are shown in Table III and are denoted as P1-P9.

The second case study comes from Section 3.3 of [39] and is called "grid world" (shown in Fig. 5). It is modeled as a continuous-time Markov chain.

This is a 5×5 grid world with a robot and a janitor in it. There are three parameters in this model: λ_1, λ_2 and r. At first, the robot is in the bottom left grid, and the janitor is in the centre grid. The robot and the janitor move at rate λ_1 and λ_2 (i.e. the time interval between two steps follows the exponential distribution with parameter $\frac{1}{\lambda_1}$ and $\frac{1}{\lambda_2}$). For the robot, in each step it has probability r to return to the bottom left grid and probability $1-r$ to move forward one grid along the dashed

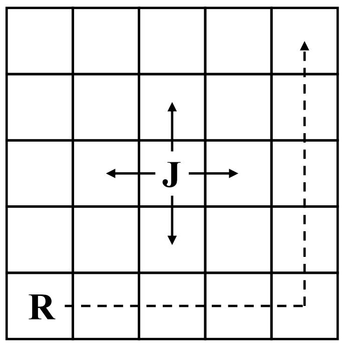

Fig. 5. Grid world model.

line. For the janitor, in each step it has equal probability to move to one of its adjacent grid. For example, if the janitor is in the centre gird, then it has 25% probability to move to its right (or left, upper, below) grid in the next step. If the janitor

TABLE IV
PROPERTIES FOR THE GRID WORLD MODEL

	λ_1	λ_2	r	p (approx)	Property
P10	0.5	0.5	0.4	0.96	Arrive within 1000 time units
P11	0.5	0.5	0.5	0.61	
P12	0.5	0.5	0.6	0.18	
P13	3.0	0.5	0.4	0.56	MEET
P14	0.8	0.5	0.5	0.89	
P15	0.5	0.5	0.6	0.98	

is in the bottom left grid, then it has 50% probability to move to its right (or upper) grid in the next step. The special case is that if the janitor is in the destination of the robot, then the robot will not move in this step, and this is called the "MEET" event. For example, if the robot decides to return to the bottom left grid (which is of r probability) and the janitor is already in the bottom left grid, then the robot will not move in this step. The system ends when the robot reaches the top right grid. In the best case, the robot reaches the top right grid in eight steps (never return to the left bottom grid and no "MEET" event).

We consider two types of properties for estimation (shown in Table IV). The first (P10-P12) is estimating the probability that the robot reaches the top right grid within 1000 time units. The second (P13-P15) is estimating the probability that the robot meets the janitor (i.e. the "MEET" event) before it reaches the top right grid. As this model is relatively complex, we only provide the approximated p for each property in the "p (approx)" column of Table IV.

For different combinations of (ε, δ), we use four algorithms to estimate the probability of P1-P15. We repeat each experiments 100 times and report the average simulation number (rounded to the nearest integer) and running time (rounded to the nearest hundredth) in Table V. The smallest running time is in bold.

For each property, the running time is roughly proportional to the simulation number. Take P5 as an example, in the $(\varepsilon, \delta) = (0.05, 0.01)$ case, the Okamoto bound runs 1060 simulations in 2.00s, with about $1.886 \cdot 10^{-3}$s for one simulation. In the $(\varepsilon, \delta) = (0.01, 0.01)$ case, the running time for one simulation is $1.882 \cdot 10^{-3}$s (Okamoto), $1.885 \cdot 10^{-3}$s (Simple), $1.893 \cdot 10^{-3}$s (Adaptive) and $1.891 \cdot 10^{-3}$s (Massart), all are very close to each other. On the other hand, for different properties, the running time is not necessarily proportional to the simulation number. For example, in the $(\varepsilon, \delta) = (0.05, 0.01)$ case of P10 and P12, the Okamoto bound both runs 1060 simulations, but the running time is quite different (1.01s for P10 and 15.55s for P12). That is because different properties have different complexities for estimation. For P10, the return probability r is relatively small ($r = 0.4$, see Table IV), so in most cases the robot can quickly reach the top right grid. However, for P12, the return probability r is relatively large ($r = 0.6$), so the robot frequently returns to the bottom left grid, which leads to very long running paths and costs a lot of time for simulation.

The results in Table V are consistent with Fig. 2. In all 90 experiments, our simple estimation algorithm, our adaptive estimation algorithm and the Massart bound-based sequential algorithm use the least running time in 39, 46 and 5 experiments respectively. Specifically, when the probability p of the property is near 0.5 (i.e. P7-P9, P11, P13), our simple estimation algorithm and our adaptive estimation algorithm are much better than the two baseline algorithms. In those cases that our simple estimation algorithm performs the best, the running time of our adaptive estimation algorithm is usually very close. For example, in all cases of P8, P9, P11 and P13, our simple estimation algorithm uses the least running time, and our adaptive estimation algorithm follows closely. In some cases, the Massart bound-based sequential algorithm performs even worse than the Okamoto bound. For example, in all cases of P13, the Massart bound-based sequential algorithm uses slightly more time than the Okamoto bound due to its complex implementation. When p is near 0 or 1 (i.e. P1-P6, P10, P12, P14, P15), in most cases, our adaptive estimation algorithm uses the least time, followed by the Massart bound-based sequential algorithm and our simple estimation algorithm, and finally the Okamoto bound (much worse than the previous three algorithms). As a special case, when $(\varepsilon, \delta) = (0.1, 0.1)$, our simple estimation algorithm has the best performance in most cases, but our adaptive estimation algorithm still has decent performance.

C. Extra Time Cost Analysis

As we have mentioned in Section IV-C, both our simple estimation algorithm and adaptive estimation algorithm involve some extra time cost, mainly due to calculating Clopper-Pearson confidence intervals. Moreover, the Massart bound-based sequential algorithm also involves extra time cost as it computes on-the-fly Clopper-Pearson confidence intervals after each new simulation. Formally, we define *extra time cost* as the time used apart from the simulations. For the Okamoto bound, it simply calculates the value of $n = \lceil \frac{1}{2\varepsilon^2} \ln(\frac{2}{\delta}) \rceil$ and performs n simulations, so its extra time cost can be regarded as 0 (calculating n only takes about $3 \cdot 10^{-7}$s on our laptop). We use the following approach to quantitatively measure the extra time cost of the other three algorithms. Suppose in an experiment, the Okamoto bound performs N_1 simulations in T_1 time, and another algorithm $\mathcal{A}$ performs N_2 simulations in T_2 time, then the algorithm $\mathcal{A}$'s extra time cost is $T_2 - N_2 \cdot \frac{T_1}{N_1}$. For example, for the $(\varepsilon, \delta) = (0.005, 0.005)$ case in Table II, we have $T_1 = 0.3955$s and $N_1 = 119830$. For our adaptive estimation algorithm, we have $T_2 = 0.4237$s and $N_2 = 57893$. Therefore, the extra time cost of our adaptive estimation algorithm in this experiment is $0.4237\text{s} - 57893 \cdot \frac{0.3955\text{s}}{119830} = 0.2326$s. Similarly, in the $(\varepsilon, \delta) = (0.01, 0.01)$ case of P4 in Table V, we have $T_1 = 47.35$s and $N_1 = 26492$. For our simple estimation algorithm, we have $T_2 = 29.92$s and $N_2 = 16684$. Therefore, the extra time cost of our simple estimation algorithm in this experiment is $29.92\text{s} - 16684 \cdot \frac{47.35\text{s}}{26492} = 0.100$s. Full results are shown in Table VI. The "Numerical" row is calculated from Table II, and the other rows are calculated from Table V. The extra time cost is in seconds and rounded to the nearest thousandth.

In numerical experiments (shown in the "Numerical" row), for all three algorithms, the lower ε and δ are, the larger the

TABLE V
Average Simulation Numbers and Running Time of Four Algorithms

(ε, δ)		(0.1,0.1)		(0.05,0.05)		(0.05,0.01)		(0.01,0.05)		(0.01,0.01)		(0.005,0.005)	
		Number	Time(s)	Number	Time(s)	Number	Time(s)	Number	Time(s)	Number	Time(s)	Number	Time(s)
P1	Okamoto	150	0.19	738	0.92	1060	1.30	18445	22.64	26492	32.47	119830	146.86
	Simple	76	0.10	402	0.50	680	0.85	9701	11.98	16684	20.54	78990	97.28
	Adaptive	80	0.12	82	**0.13**	101	**0.15**	680	**0.87**	959	**1.19**	2196	**2.70**
	Massart	66	**0.08**	167	0.22	224	0.28	914	1.13	1252	1.56	3153	3.91
P2	Okamoto	150	0.22	738	1.06	1060	1.51	18445	26.21	26492	37.64	119830	170.39
	Simple	76	0.11	402	0.58	680	0.98	9701	13.83	16684	23.79	78990	112.65
	Adaptive	81	0.14	94	**0.16**	167	**0.26**	1144	**1.65**	1608	**2.32**	5123	**7.31**
	Massart	67	**0.10**	180	0.26	242	0.35	1270	1.82	1759	2.52	5326	7.63
P3	Okamoto	150	0.23	738	1.14	1060	1.62	18445	28.09	26492	40.33	119830	182.81
	Simple	76	0.12	402	0.63	680	1.05	9701	14.78	16684	25.48	78990	120.75
	Adaptive	81	0.15	134	**0.23**	221	**0.36**	1447	**2.24**	2136	**3.30**	6896	**10.55**
	Massart	70	**0.11**	191	0.29	264	0.40	1664	2.56	2319	3.55	7744	11.92
P4	Okamoto	150	0.27	738	1.34	1060	1.90	18445	32.97	26492	47.35	119830	214.03
	Simple	76	**0.14**	402	0.73	680	1.22	9701	17.35	16684	29.92	78990	141.59
	Adaptive	88	0.18	257	**0.50**	444	0.82	3448	**6.22**	5497	**9.87**	21548	**38.71**
	Massart	91	0.16	304	0.55	426	**0.76**	4430	7.96	6328	11.39	25913	46.69
P5	Okamoto	150	0.29	738	1.40	1060	2.00	18445	34.75	26492	49.87	119830	225.62
	Simple	76	**0.15**	402	0.77	680	1.29	9701	18.30	16684	31.46	78990	149.20
	Adaptive	89	0.19	354	**0.72**	567	**1.11**	5254	**9.97**	8604	**16.28**	36262	**68.50**
	Massart	111	0.21	415	0.80	595	1.14	7570	14.35	10780	20.38	46142	87.56
P6	Okamoto	150	0.29	738	1.41	1060	2.02	18445	35.14	26492	50.41	119830	228.42
	Simple	76	**0.15**	402	**0.78**	680	**1.29**	9701	18.52	16684	31.82	78990	150.86
	Adaptive	87	0.19	423	0.85	692	1.35	7993	**15.34**	13385	**25.60**	58429	**111.41**
	Massart	138	0.26	590	1.14	836	1.60	12574	24.10	18002	34.49	79318	152.07
P7	Okamoto	150	0.27	738	1.36	1060	1.94	18445	33.85	26492	48.52	119830	219.76
	Simple	76	**0.14**	402	**0.75**	680	**1.25**	9701	**17.83**	16684	30.63	78990	145.37
	Adaptive	87	0.18	421	0.82	700	1.32	9722	17.96	16353	**30.11**	73545	**135.21**
	Massart	149	0.27	699	1.30	998	1.84	16123	29.77	23036	42.53	102789	189.91
P8	Okamoto	150	0.26	738	1.28	1060	1.81	18445	31.54	26492	45.29	119830	204.75
	Simple	76	**0.13**	402	**0.70**	680	**1.17**	9701	**16.64**	16684	**28.57**	78990	**135.52**
	Adaptive	86	0.17	417	0.75	696	1.23	9805	16.87	16851	28.99	79460	136.50
	Massart	150	0.26	738	1.28	1060	1.82	18151	31.31	25950	44.72	116664	201.09
P9	Okamoto	150	0.23	738	1.16	1060	1.64	18445	28.57	26492	41.08	119830	185.82
	Simple	76	**0.12**	402	**0.63**	680	**1.06**	9701	**15.09**	16684	**25.93**	78990	**122.65**
	Adaptive	86	0.16	413	0.69	690	1.11	9799	15.30	16784	26.17	79090	123.46
	Massart	150	0.23	738	1.15	1060	1.66	18445	28.75	26492	41.26	119830	187.11
P10	Okamoto	150	0.14	738	0.78	1060	1.01	18445	17.48	26492	27.57	119830	122.58
	Simple	76	**0.07**	402	0.43	680	0.64	9701	9.23	16684	17.18	78990	80.88
	Adaptive	86	0.11	244	0.29	359	**0.37**	2943	**2.86**	4849	**5.07**	18189	**18.66**
	Massart	85	0.09	271	**0.29**	390	0.37	3718	3.58	5276	5.57	20917	21.70
P11	Okamoto	150	0.49	738	2.63	1060	3.33	18445	58.57	26492	91.72	119830	409.15
	Simple	76	**0.25**	402	**1.44**	680	**2.16**	9701	**30.94**	16684	**57.59**	78990	**269.32**
	Adaptive	86	0.30	414	1.51	700	2.25	9815	31.38	16882	58.17	79569	272.41
	Massart	150	0.49	738	2.64	1059	3.38	18033	57.74	25780	90.41	115746	398.05
P12	Okamoto	150	2.22	738	12.18	1060	15.55	18445	273.56	26492	429.16	119830	1807.86
	Simple	76	**1.11**	402	**6.71**	680	**9.99**	9701	143.73	16684	268.49	78990	1188.88
	Adaptive	88	1.32	415	6.97	683	10.06	7587	**112.74**	12576	**201.82**	54612	**822.33**
	Massart	132	2.01	566	9.33	797	11.71	11745	174.55	16735	272.23	73610	1104.28
P13	Okamoto	150	0.08	738	0.42	1060	0.54	18445	9.46	26492	14.69	119830	67.00
	Simple	76	**0.04**	402	**0.24**	680	**0.35**	9701	**5.01**	16684	**9.33**	78990	**44.22**
	Adaptive	86	0.07	414	0.28	692	0.40	9799	5.15	16791	9.48	79293	44.62
	Massart	150	0.08	738	0.43	1060	0.56	18445	9.73	26492	15.14	119642	68.42
P14	Okamoto	150	0.37	738	2.00	1060	2.56	18445	44.44	26492	70.31	119830	319.52
	Simple	76	**0.20**	402	1.11	680	1.64	9701	23.59	16684	43.90	78990	210.24
	Adaptive	88	0.24	368	**1.04**	573	**1.42**	5420	**13.22**	8829	**23.32**	37482	**99.62**
	Massart	111	0.28	428	1.18	600	1.47	7810	19.11	11146	29.90	47684	128.35
P15	Okamoto	150	2.28	738	12.19	1060	15.52	18445	271.73	26492	429.96	119830	1950.85
	Simple	76	**1.13**	402	6.69	680	9.94	9701	143.75	16684	267.53	78990	1284.04
	Adaptive	84	1.30	191	**3.18**	264	**3.88**	2104	**31.27**	3278	**52.94**	11866	**192.82**
	Massart	77	1.17	223	3.75	309	4.56	2408	35.69	3480	56.61	12619	205.95

extra time cost is. Our adaptive estimation algorithm has a base time cost of about 0.02s, which comes from the strategy analysis part (lines 7-15, Algorithm 6). As a result, when the simulation number is small, the extra time cost of our adaptive estimation algorithm is higher than the other two algorithms. When the simulation number is large, the Massart bound-based sequential algorithm has more extra time cost than the other two algorithms. Note that in our numerical experiments in Table II, we use the Bernoulli variables generator to perform simulations, so the running time for each simulation can be regarded as

TABLE VI
EXTRA TIME COST (IN SECONDS) OF THREE ALGORITHMS

(ε,δ)		(0.1,0.1)	(0.05,0.05)	(0.05,0.01)	(0.01,0.05)	(0.01,0.01)	(0.005,0.005)
Numerical	Simple	0.001	0.002	0.003	0.025	0.036	0.179
	Adaptive	0.026	0.036	0.037	0.094	0.114	0.233
	Massart	0.002	0.008	0.012	0.191	0.304	1.223
P1	Simple	0.002	0.001	0.015	0.078	0.088	0.476
	Adaptive	0.027	0.026	0.023	0.031	0.013	0.013
	Massart	0.001	0.006	0.005	0.014	0.021	0.050
P2	Simple	-0.001	0.000	0.009	0.045	0.085	0.331
	Adaptive	0.028	0.025	0.026	0.028	0.031	0.025
	Massart	0.001	0.001	0.003	0.016	0.016	0.055
P3	Simple	0.001	0.008	0.009	0.003	0.085	0.247
	Adaptive	0.024	0.028	0.023	0.032	0.046	0.030
	Massart	-0.001	-0.002	0.001	0.026	0.022	0.111
P4	Simple	0.001	-0.004	0.002	0.005	0.100	0.504
	Adaptive	0.023	0.027	0.024	0.059	0.044	0.227
	Massart	0.001	-0.002	-0.002	0.044	0.081	0.409
P5	Simple	0.001	0.005	0.006	0.022	0.052	0.480
	Adaptive	0.023	0.049	0.036	0.073	0.089	0.224
	Massart	0.002	0.014	0.019	0.090	0.092	0.685
P6	Simple	0.000	0.008	-0.006	0.038	0.073	0.289
	Adaptive	0.022	0.037	0.031	0.110	0.131	0.036
	Massart	-0.001	0.007	0.002	0.144	0.234	0.874
P7	Simple	0.001	0.005	0.009	0.025	0.070	0.508
	Adaptive	0.025	0.042	0.034	0.115	0.154	0.332
	Massart	0.002	0.011	0.011	0.178	0.339	1.403
P8	Simple	0.002	0.001	0.009	0.056	0.042	0.551
	Adaptive	0.026	0.027	0.042	0.102	0.182	0.728
	Massart	0.000	-0.001	0.011	0.270	0.352	1.748
P9	Simple	0.003	-0.004	0.003	0.062	0.060	0.167
	Adaptive	0.024	0.035	0.041	0.124	0.147	0.815
	Massart	0.003	-0.010	0.019	0.177	0.182	1.289
P10	Simple	0.002	0.008	-0.007	0.038	-0.182	0.078
	Adaptive	0.029	0.036	0.022	0.069	0.029	0.055
	Massart	0.004	0.006	0.001	0.054	0.083	0.304
P11	Simple	-0.002	0.014	0.019	0.136	-0.176	-0.382
	Adaptive	0.023	0.040	0.051	0.212	-0.278	0.726
	Massart	0.002	0.015	0.047	0.484	1.156	2.844
P12	Simple	-0.011	0.079	0.015	-0.143	-1.778	-2.832
	Adaptive	0.024	0.117	0.049	0.219	-1.910	-1.592
	Massart	0.067	-0.010	0.023	0.362	1.130	-6.263
P13	Simple	0.002	0.009	0.003	0.030	0.085	0.051
	Adaptive	0.025	0.042	0.041	0.118	0.171	0.280
	Massart	0.000	0.013	0.016	0.270	0.457	1.517
P14	Simple	0.007	0.016	-0.007	0.217	-0.384	-0.381
	Adaptive	0.021	0.044	0.034	0.159	-0.111	-0.325
	Massart	-0.001	0.015	0.022	0.288	0.313	1.203
P15	Simple	-0.026	0.052	-0.015	0.838	-3.245	-1.933
	Adaptive	0.024	0.030	0.015	0.278	-0.260	-0.361
	Massart	0.001	0.066	0.030	0.213	0.138	0.505

a constant, which significantly reduces the randomness when analyzing the extra time cost.

The results for other rows are similar. However, as we use concrete models and properties in Table V, the simulation time fluctuates much stronger than the Bernoulli variables generator, which brings some randomness to results. Therefore, in some cases, the extra time cost is negative.

In general, for these three algorithms, the extra time cost is very small compared to their running time in Section V-B. Note that the models and properties in Section V-B are relatively easy to simulate. If we consider more complex models and properties in practice, the simulation part will take more time, and the extra time cost of these three algorithms will be almost negligible.

VI. DISCUSSION

A. Selection of Hyperparameters

There are several hyperparameters in our adaptive estimation algorithm (Algorithm 6). In this subsection, we discuss the selection of these hyperparameters.

1) Pre-Simulation Number n_2: In our adaptive estimation algorithm, we perform $n_2 = max(min(\lceil 0.01 \cdot n_1 \rceil, 100), 10)$ pre-simulations first to obtain a rough estimation for p (lines 3-6, Algorithm 6), where n_1 is the simulation number needed by our simple estimation algorithm. Here we discuss the influence of the pre-simulation number n_2. Specifically, for each combination of (ε, δ) and $p = 0, 0.01, 0.02, \cdots, 0.99, 1$, we consider six different $n_2 = 2/5/10/50/100/500$. We repeat each

TABLE VII
AVERAGE SIMULATION NUMBERS OF OUR ADAPTIVE ESTIMATION ALGORITHM USING DIFFERENT n_2

ε	δ	n_1	n_2					
			2	5	10	50	100	500
0.1	0.1	76	86	84	87	126	176	576
0.05	0.05	402	387	379	380	413	462	861
0.05	0.01	680	635	627	625	648	696	1094
0.01	0.05	9701	7973	7830	7704	7659	7678	8051
0.01	0.01	16684	13550	13305	13047	12885	12881	13259
0.005	0.005	78990	61575	60254	58971	57959	57893	58232

experiment 100 times and report the average simulation number (rounded to the nearest integer) in Table VII. We also list n_1 here for ease of reference.

The average simulation number is relatively large when n_2 is too small or too large. Intuitively, if n_2 is too small (e.g. $n_2 = 2$), the estimation p_1 will be very inaccurate, which may mislead the strategy analysis part. On the other hand, if n_2 is too large, the loss outweighs the gain. For example, in the $(\varepsilon, \delta) = (0.01, 0.01)$ case, if we perform 2 pre-simulations, the total simulation number is 13550. If we perform 5 pre-simulations, the estimated p_1 will be more accurate, which leads to a lower simulation number (13305) in total. The total simulation number further decreases if we perform 10, 50 or 100 pre-simulations. However, if we perform 500 pre-simulations, the total simulation number increases again. Actually, if we perform 100 pre-simulations, we need 12881-100=12781 simulations later, and if we perform 500 pre-simulations, we need 13259–500=12759 simulations later. That means although we perform 400 more pre-simulations, we only gain a reduction of 12781-12759=22 simulations later, which is not a good trade.

2) Confidence δ' for Interval Estimation: As we have mentioned in the last paragraph of Section IV-B, we need to set a hyperparameter δ' $(0 < \delta' < \delta)$ in our adaptive estimation algorithm to control the allocation of confidence. A higher δ' leads to a tighter confidence interval for the interval estimation part (lines 19-22, Algorithm 6), but also leaves less confidence for the final estimation part (lines 23-25, Algorithm 6). In our implementation, we select $\delta' = 0.05 \cdot \delta$. Here we discuss the sensitivity of our adaptive estimation algorithm towards δ'. For each combination of (ε, δ) and $p = 0, 0.01, 0.02, \cdots, 0.99, 1$, we consider nine different $\delta' = 0.0001/0.001/0.01/0.02/0.05/0.1/0.2/0.5/0.9 \cdot \delta$. We repeat each experiment 100 times and report the average simulation number (rounded to the nearest integer) in Table VIII.

For different combinations of (ε, δ), the optimal δ' is different. When $0.01 \cdot \delta \leq \delta' \leq 0.2 \cdot \delta$, the difference between the average simulation numbers using different δ' is very small, usually no more than 2%. Therefore, our adaptive estimation algorithm is insensitive to δ' in a relative wide range. However, if δ' is too big (e.g. $0.9 \cdot \delta$) or too small (e.g. $0.0001 \cdot \delta$), the performance of our algorithm becomes poorer, especially when δ' approaches δ.

3) Candidate Simulation Number List n_list: In the strategy analysis part of our adaptive estimation algorithm, we consider a candidate simulation number list n_list (line 8, Algorithm 6). For each candidate in n_list, we estimate its total cost and select the best candidate for later use. In our implementation, we consider 20 different candidate simulation numbers from $\lceil 0.01 \cdot n_1 \rceil$ to $\lceil 0.20 \cdot n_1 \rceil$. Here we discuss the influence of choosing different n_list. Specifically, we consider five variants for n_list. The first variant V1 is to consider a wider range of simulation numbers, from $\lceil 0.01 \cdot n_1 \rceil$ to $\lceil 0.99 \cdot n_1 \rceil$, with 99 candidates in total. The second variant V2 is to consider a finer division of simulation numbers, from $\lceil 0.002 \cdot n_1 \rceil$ to $\lceil 0.200 \cdot n_1 \rceil$, with 100 candidates in total. The third variant V3 is to integrate the above two variants, from $\lceil 0.002 \cdot n_1 \rceil$ to $\lceil 0.998 \cdot n_1 \rceil$, with 499 candidates in total. The fourth variant V4 is to consider a narrower range of simulation numbers, from $\lceil 0.01 \cdot n_1 \rceil$ to $\lceil 0.10 \cdot n_1 \rceil$, with 10 candidates in total. The fifth variant V5 is to consider a coarser division of simulation numbers, from $\lceil 0.05 \cdot n_1 \rceil$ to $\lceil 0.20 \cdot n_1 \rceil$, with 4 candidates in total. For each variant, combination of (ε, δ) and $p = 0, 0.01, 0.02, \cdots, 0.99, 1$, we repeat the experiment 100 times and report the average simulation number (rounded to the nearest integer) in Table IX. The results using the original implementation in Algorithm 6 (i.e. consider 20 candidates from $\lceil 0.01 \cdot n_1 \rceil$ to $\lceil 0.20 \cdot n_1 \rceil$) are presented in the "Original" column.

For variants V1-V3, the average simulation numbers are very close to the original implementation. To explore the reason behind it, we inspect all best candidates selected in V1 and find most of them lie between $\lceil 0.01 \cdot n_1 \rceil$ and $\lceil 0.20 \cdot n_1 \rceil$. Therefore, the benefit of a wider range of simulation numbers is limited. Moreover, the strategy analysis part relies heavily on p_1, which itself is a rough estimation of p, so the benefit of a finer division is also limited. For variant V4, the average simulation numbers when $(\varepsilon, \delta) = (0.05, 0.05)$ and $(0.05, 0.01)$ are slightly higher than the original implementation. For variant V5, the average simulation number when $(\varepsilon, \delta) = (0.005, 0.005)$ is slightly higher than the original implementation. In general, the difference between the average simulation numbers of these five variants and the original implementation is small, usually no more than 3%.

B. Suggestion for Practical Use

Among the four algorithms we consider in this paper, the Okamoto bound performs much worse than the other three algorithms, so we only recommend using it when implementation simplicity or extra conservativeness is the priority. The Massart bound-based sequential algorithm is worse than our adaptive estimation algorithm in most cases, and when p is near 0.5, its performance is quite poor (similar to the Okamoto bound). Therefore, we also do not recommend using it in practice.

Our adaptive estimation algorithm has consistently good performance in all cases. Although it uses slightly more time than our simple estimation algorithm when p is near 0.5, the difference is usually very small. When p is near 0 or 1, our adaptive estimation algorithm is much better than our simple estimation algorithm. The only exception is when the simulation number is very small (e.g. the $(\varepsilon, \delta) = (0.1, 0.1)$ case), our simple estimation algorithm always has slightly better performance than our

TABLE VIII
AVERAGE SIMULATION NUMBERS OF OUR ADAPTIVE ESTIMATION ALGORITHM USING DIFFERENT δ'

ε	δ	δ'/δ								
		0.0001	0.001	0.01	0.02	0.05	0.1	0.2	0.5	0.9
0.1	0.1	86	86	86	87	87	86	86	86	87
0.05	0.05	404	397	385	382	380	378	378	387	408
0.05	0.01	663	641	631	628	625	622	621	630	682
0.01	0.05	7896	7801	7702	7669	7662	7699	7791	8149	8787
0.01	0.01	13441	13077	12928	12909	12881	12936	13039	13539	14544
0.005	0.005	59806	58270	57925	57870	57893	58244	59029	61586	66704

TABLE IX
AVERAGE SIMULATION NUMBERS OF OUR ADAPTIVE ESTIMATION ALGORITHM USING DIFFERENT n_list

ε	δ	Original	V1	V2	V3	V4	V5
0.1	0.1	87	84	87	84	86	87
0.05	0.05	380	380	379	379	389	380
0.05	0.01	625	625	624	624	634	622
0.01	0.05	7662	7659	7658	7664	7662	7672
0.01	0.01	12881	12909	12888	12909	12876	12904
0.005	0.005	57893	57919	57975	57974	57907	58272

adaptive estimation algorithm. Therefore, we recommend using our simple estimation algorithm when the simulation number is very small, and using our adaptive estimation algorithm in all other cases. Specifically, both our two algorithms calculate $n = interval\_sensitive\_bound(\varepsilon, \delta, 0, 1)$ first (see line 1 of Algorithm 5 and line 1 of Algorithm 6), so in practice, we calculate n first, if $n < 200$, we use our simple estimation algorithm, otherwise we use our adaptive estimation algorithm.

C. Parallelization

Like most SMC algorithms, all simulations in our algorithms (i.e. line 5 and 16 of Algorithm 4, line 5 and 21 of Algorithm 6) can be parallelized because each simulation is independent of each other. The parallelization implementation is the same with existing methods: we simply use many threads to perform simulations in parallel, then combine their simulation results. Ideally, using n threads can speed up the simulations by n times at most (if all threads start and terminate simultaneously, and extra time cost is negligible). Moreover, the validation procedure (lines 7-9, Algorithm 3) and the strategy analysis part (lines 10-14, Algorithm 6) can also be parallelized. However, as their time cost is already very small (see Table VI), the benefit of parallelization is marginal.

D. Bias

The *bias* of an estimator $\hat{p}$ is the expected difference between it and the true value p. An algorithm is *unbiased* if its bias is 0 for any p. In our two estimation algorithms, we use the midpoint of the confidence interval $CP\_int(n, x, \delta)$ (or $CP\_int(n, x, \delta) \cap [a, b]$) as the estimation result, which is different from the unbiased estimator $\frac{x}{n}$ used by the Okamoto bound. This difference is particularly notable if x is near 0 or n. The bias does not affect the soundness of our algorithms, however, sometimes people may prefer an unbiased estimation algorithm. Therefore, we briefly discuss how to eliminate the bias of our algorithms.

In our simple estimation algorithm, we first calculate a simulation number n such that the length of $CP\_int(n, x, \delta)$ is lower than $2 \cdot \varepsilon$ for all $x = 0, 1, \cdots, n$, then perform n simulations and use the midpoint of the confidence interval $CP\_int(n, x, \delta)$ as the estimation result. The unbiased version of our simple estimation algorithm is as follows. We first calculate a simulation number n (by binary search) such that $CP\_int(n, x, \delta) \subseteq [\frac{x}{n} - \varepsilon, \frac{x}{n} + \varepsilon]$ for all $x = 0, 1, \cdots, n$, then perform n simulations and use $\frac{x}{n}$ as the estimation result. The soundness proof is very similar to Theorem 3 ($Pr(p \in [\frac{x}{n} - \varepsilon, \frac{x}{n} + \varepsilon]) \geq Pr(p \in CP\_int(n, x, \delta)) \geq 1 - \delta$). In practice, we find the obtained simulation number n is slightly larger than the original version. For example, when $(\varepsilon, \delta) = (0.05, 0.05)/(0.01, 0.01)$, the simulation number for the unbiased version is 404/16687, while for the original simple estimation algorithm is 402/16684.

It is possible to use the similar method to modify our adaptive estimation algorithm. The modified $interval\_sensitive\_bound$ function is to calculate a simulation number n (by binary search) such that $CP\_int(n, x, \delta) \cap [a, b] \subseteq [\frac{x}{n} - \varepsilon, \frac{x}{n} + \varepsilon]$ for all $x = 0, 1, \cdots, n$. The modified $estimate$ function simply performs n simulations and uses $\frac{x}{n}$ as the estimation result. However, the performance of the modified algorithm is much worse than the original adaptive estimation algorithm. The reason is that the modified algorithm always uses $\frac{x}{n}$ as the estimator even if $\frac{x}{n} \notin CP\_int(n, x, \delta) \cap [a, b]$, while the original algorithm uses the midpoint of $CP\_int(n, x, \delta) \cap [a, b]$ as the estimator.

Finally, it should be noted that many sequential algorithms (e.g. [19], [21], [30]) also use $\frac{x}{n}$ as the estimator, however, their sequential sampling manner might introduce some bias to the final estimation result.

E. Threats to Validity

Due to the stochastic nature of SMC, all our experiments inevitably involve randomness. Therefore, we repeat all experiments 100 times (100000/1000000 times for the soundness check experiment in Section V-A2) and report the average results. Moreover, the results of different experiments are consistent with each other (e.g. Fig. 2 and Table V), which further supports the reliability of our results.

Generality is another important issue. In our experiments, we consider six combinations of (ε, δ), from lightweight estimation using less than 100 simulations to heavyweight estimation using more than 100000 simulations, which can cover most cases in practice. In the numerical experiments part, we consider 101 different $p = 0, 0.01, 0.02, \cdots, 0.99, 1$. In the case studies part, we consider two models and fifteen properties. There are many other models and properties that we did not evaluate. However, the simulation number only depends on ε, δ and p, and the running time is roughly proportional to the simulation number, so our numerical experiments can partly fill the gap.

There may be implementation errors in our implementations of algorithms. Therefore, we double-check our code and thoroughly test it. For the Massart bound-based sequential algorithm, we also compare our results with the results in [19] and find they are similar. We release all our code in [38] for further validation.

VII. Conclusion and Future Work

Determining the simulation number is the foundation of SMC estimation algorithms. In this paper, we propose an adaptive estimation algorithm that can achieve the same theoretical guarantee using 40%-60% fewer simulations than the widely-used Okamoto bound. Our adaptive estimation algorithm is efficient, sound and robust. It can be directly integrated into existing statistical model checkers to significantly reduce the verification time of SMC estimation problems.

Although our algorithm is much more efficient than existing algorithms, there still exists some optimization room when p is near 0 or 1 (see Section V-A2). In the future, we plan to investigate if we can further reduce the simulation number for p near 0 or 1.

References

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[2] M. Y. Vardi, “Automatic verification of probabilistic concurrent finite state programs,” in *Proc. 26th Annu. Symp. Found. Comput. Sci.*, 1985, pp. 327–338.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, “Model-checking algorithms for continuous-time Markov chains,” *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 524–541, Jun. 2003.

[4] D. Bustan, S. Rubin, and M. Y. Vardi, “Verifying ω-regular properties of Markov chains,” in *Proc. 16th Int. Conf. Comput. Aided Verification*, 2004, pp. 189–201.

[5] H. L. Younes and R. G. Simmons, “Probabilistic verification of discrete event systems using acceptance sampling,” in *Proc. 14th Int. Conf. Comput. Aided Verification*, 2002, pp. 223–235.

[6] K. Sen, M. Viswanathan, and G. Agha, “Statistical model checking of black-box probabilistic systems,” in *Proc. 16th Int. Conf. Comput. Aided Verification*, 2004, pp. 202–215.

[7] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, “Approximate probabilistic model checking,” in *Proc. 5th Int. Workshop Verification Model Checking Abstract Interpretation*, 2004, pp. 73–84.

[8] K. Sen, M. Viswanathan, and G. Agha, “On statistical model checking of stochastic systems,” in *Proc. 17th Int. Conf. Comput. Aided Verification*, 2005, pp. 266–280.

[9] R. Grosu and S. A. Smolka, “Monte Carlo model checking,” in *Proc. 11th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2005, pp. 271–286.

[10] H. L. Younes, “Error control for probabilistic model checking,” in *Proc. 7th Int. Workshop Verification, Model Checking, Abstract Interpretation*, 2006, pp. 142–156.

[11] G. Agha and K. Palmskog, “A survey of statistical model checking,” *ACM Trans. Model. Comput. Simul.*, vol. 28, no. 1, pp. 1–39, Jan. 2018.

[12] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in *Proc. 23rd Int. Conf. Comput. Aided Verification*, 2011, pp. 585–591.

[13] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen, “UPPAAL SMC tutorial,” *Int. J. Softw. Tools Technol. Transfer*, vol. 17, no. 4, pp. 397–415, Aug. 2015.

[14] 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,” *Perform. Eval.*, vol. 90, pp. 53–77, Aug. 2015.

[15] A. Wald, “Sequential tests of statistical hypotheses,” *Ann. Math. Statist.*, vol. 16, no. 2, pp. 117–186, Jun. 1945.

[16] A. Legay, S. Sedwards, and L.-M. Traonouez, “Plasma lab: A modular statistical model checking platform,” in *Proc. 7th Int. Symp. Leveraging Appl. Formal Methods*, 2016, pp. 77–93.

[17] B. L. Mediouni et al., “SBIP 2.0: Statistical model checking stochastic real-time systems,” Verimag Research Report, Tech. Rep. TR-2018-5, 2018.

[18] M. Okamoto, “Some inequalities relating to the partial sum of binomial probabilities,” *Ann. Inst. Statist. Math.*, vol. 10, no. 1, pp. 29–35, Mar. 1959.

[19] C. Jegourel, J. Sun, and J. S. Dong, “Sequential schemes for frequentist estimation of properties in statistical model checking,” *ACM Trans. Model. Comput. Simul.*, vol. 29, no. 4, pp. 1–22, Oct. 2019.

[20] C. E. Budde, P. R. D’Argenio, A. Hartmanns, and S. Sedwards, “An efficient statistical model checker for nondeterminism and rare events,” *Int. J. Softw. Tools Technol. Transfer*, vol. 22, no. 6, pp. 759–780, Dec. 2020.

[21] J. Frey, “Fixed-width sequential confidence intervals for a proportion,” *Amer. Statist.*, vol. 64, no. 3, pp. 242–249, 2010.

[22] C. J. Clopper and E. S. Pearson, “The use of confidence or fiducial limits illustrated in the case of the binomial,” *Biometrika*, vol. 26, no. 4, pp. 404–413, 1934.

[23] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 13–30, Mar. 1963.

[24] P. Massart, “The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality,” *Ann. Probability*, vol. 18, no. 3, pp. 1269–1283, Jul. 1990.

[25] F. Scholz, “Confidence bounds & intervals for parameters relating to the binomial, negative binomial, Poisson and hypergeometric distributions with applications to rare events,” UW Homepage. Accessed: May 16, 2024. [Online]. Available: https://faculty.washington.edu/fscholz/DATAFILES498B2008/ConfidenceBounds.pdf

[26] “PRISM manual.” PRISM. Accessed: May 16, 2024. [Online]. Available: https://www.prismmodelchecker.org/manual/RunningPRISM/StatisticalModelChecking

[27] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and Z. Wang, “Time for statistical model checking of real-time systems,” in *Proc. 23rd Int. Conf. Comput. Aided Verification*, 2011, pp. 349–355.

[28] P. Ballarini, H. Djafri, M. Duflot, S. Haddad, and N. Pekergin, “COSMOS: A statistical model checker for the hybrid automata stochastic logic,” in *Proc. 8th Int. Conf. Quantitative Eval. Syst.*, 2011, pp. 143–144.

[29] Y. S. Chow and H. Robbins, “On the asymptotic theory of fixed-width sequential confidence intervals for the mean,” *Ann. Math. Statist.*, vol. 36, no. 2, pp. 457–462, Apr. 1965.

[30] J. Chen and J. Xu, “Sampling adaptively using the Massart inequality for scalable learning,” in *Proc. 12th Int. Conf. Mach. Learn. Appl.*, 2013, pp. 362–367.

[31] S. Peyronnet, R. Lassaigne, and T. Herault, “APMC 3.0: Approximate verification of discrete and continuous time Markov chains,” in *Proc. 3rd Int. Conf. Quantitative Eval. Syst.*, 2006, pp. 129–130.

[32] “UPPAAL online documentation,” UPPAAL. Accessed: May 16, 2024. [Online]. Available: https://docs.uppaal.org/language-reference/query-semantics/smc_queries/ci_estimation/

[33] O. Watanabe, “Sequential sampling techniques for algorithmic learning theory,” *Theor. Comput. Sci.*, vol. 348, no. 1, pp. 3–14, Dec. 2005.

[34] P. Dagum, R. Karp, M. Luby, and S. Ross, “An optimal algorithm for Monte Carlo estimation,” *SIAM J. Comput.*, vol. 29, no. 5, pp. 1484–1496, 2000.

[35] V. Mnih, C. Szepesvári, and J.-Y. Audibert, “Empirical Bernstein stopping,” in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 672–679.

[36] D. Reijsbergen, P.-T. de Boer, W. Scheinhardt, and B. Haverkort, “Rare event simulation for highly dependable systems with fast repairs,” *Perform. Eval.*, vol. 69, nos. 7–8, pp. 336–355, 2012.

[37] C. Jegourel, A. Legay, and S. Sedwards, "Importance splitting for statistical model checking rare properties," in *Proc. 25th Int. Conf. Comput. Aided Verification*, 2013, pp. 576–591.
[38] Zenodo, "Implementation of the SMC estimation algorithms", 2024. Accessed: May 16, 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10654358
[39] H. L. Younes, M. Kwiatkowska, G. Norman, and D. Parker, "Numerical vs. statistical probabilistic model checking," *Int. J. Softw. Tools Technol. Transfer*, vol. 8, pp. 216–228, Jun. 2006.

Hao Bu received the bachelor's degree in mathematics from Peking University, in 2020, where he is currently working toward the Ph.D. degree. His research interests include formal methods and trustworthy AI.

Meng Sun received the bachelor's and Ph.D. degrees in applied mathematics from Peking University, in 1999 and 2005, respectively. He was a Postdoctoral Researcher with the National University of Singapore, from 2005 to 2006, and a Scientific Staff Member with the CWI, the Netherlands, from 2006 to 2010. He joined as an Assistant Professor with the School of Mathematical Sciences, Peking University, in 2010, and became a Full Professor, in 2017. His research interests mainly lie in theory of programming and software formal methods in general, and in particular includes coordination models and languages, coalgebra theory and its applications, software verification and testing, cyber-physical systems, blockchain and smart contracts, theoretical foundation, and certification techniques of deep learning systems.