Perfect Simulation With Exponential Tails on the Running Time

Mark Huber*
Department of Mathematics and ISDS, Duke University, Durham, NC 27713;
e-mail: mhuber@math.duke.edu

Received 15 September 2006; accepted 14 May 2007; received in final form 15 June 2007
Published online 2 April 2008 in Wiley InterScience (www.interscience.wiley.com).
DOI 10.1002/rsa.20212

ABSTRACT: Monte Carlo algorithms typically need to generate random variates from a probability distribution described by an unnormalized density or probability mass function. Perfect simulation algorithms generate random variates exactly from these distributions, but have a running time T that is itself an unbounded random variable. This article shows that commonly used protocols for creating perfect simulation algorithms, such as Coupling From the Past can be used in such a fashion that the running time is unlikely to be very much larger than the expected running time. © 2008 Wiley Periodicals, Inc. Random Struct. Alg., 33, 29–43, 2008

Keywords: Monte Carlo; perfect simulation; coupling from the past; noninterruptible algorithms

1. MONTE CARLO METHODS

Perfect simulation algorithms require an unbounded random amount of time T to generate random variates from distributions. This article looks into a natural question about these algorithms, namely, how likely is T to be very much larger than its expected value?

The basic problem of Monte Carlo algorithms is the following. Given a target distribution π over a measurable state space $(\Omega, \mathcal{F})$, generate random variates drawn from this distribution. Monte Carlo methods are used in statistics to estimate exact p-values [1], in physics to estimate partition functions [12], in computer science for approximation algorithms for $\#P$ complete problems [11], and in a variety of other applications (see [6] for an overview).

Correspondence to: M. Huber
*Supported by NSF CAREER (DMS-05-48153).

Typically the distribution is given in terms of an unnormalized density or probability mass function:

$$\pi(\mathrm{d}x) = [f(x)\mathrm{d}x]/Z, \quad \text{or } \pi(x) = w(x)/Z. \tag{1.1}$$

Often it is an $\#P$ complete problem to compute Z exactly.

The classical approach to this problem is to use Monte Carlo Markov chain methods. Using a technique such as Gibbs sampling or Metropolis-Hastings, build a Markov chain whose stationary distribution matches π. Then, starting from an arbitrary state, run the Markov chain forward until the distribution of the state is close to stationary.

Unfortunately, it is usually very difficult to ascertain how long a Markov chain must be run before the state is close to stationary. Heuristic tests such as autocorrelation can alert the user when the state is far away from stationarity, but cannot guarantee that the sample generated has a distribution close to the target.

Coupling from the past (CFTP) [13] was the first widely used perfect simulation algorithm. The advantages of a perfect simulation algorithm are twofold.

1. Random variates come exactly from the desired distribution π (they are exact simulation methods.)
2. They are true algorithms where number of steps that need to be taken varies dynamically with the problem.

There are situations where an analysis of the mixing time of the Markov chain is possible ([2, 10]). These results are usually of the following form. Define the total variation distance between distributions ν_1 and ν_2 on $(\Omega, \mathcal{F})$:

$$d_{TV}(\nu_1, \nu_2) := \sup_{A \in \mathcal{F}} |\nu_1(A) - \nu_2(A)|. \tag{1.2}$$

Consider a Markov chain with stationary distribution π, and let $x_0 \in \Omega$, and $p_t(x_0)$ be the distribution of the state of the Markov chain after t steps given that the state started in x_0. Note that $d_{TV}(p_t(x_0), \pi)$ is a decreasing function of time. Set

$$\tau(x_0, \epsilon) = \inf\{t : d_{TV}(p_t(x_0), \pi) \leq \epsilon\}. \tag{1.3}$$

Then $\tau(x_0, \epsilon)$ is the mixing time of the chain started at x_0. Taking the supremum over all $x_0 \in \Omega$ gives the mixing time of the chain $\tau(\epsilon)$.

The chain is rapidly mixing if $\tau(\epsilon) = p_1(n) + p_2(n)\ln(\epsilon^{-1})$, where n is a measure of problem input size and p_1 and p_2 are $O(n^d)$ for some d.

When a perfect simulation algorithm with bounded expected running time and a rapidly mixing Markov chain both exist for a problem, it is usually better to run the perfect simulation method. For this, consider a Markov chain where the mixing time for a fixed ϵ is shown to be less than or equal to $p(n)$. Then $p(n)$ steps must be taken for each random variate generated in order to guarantee that the chain has mixed. That is, the algorithm to approximately generate variates will be $\Theta(p(n))$. Unless the user is willing to reanalyze the mixing time argument for the specific problem at hand, the full $p(n)$ steps must be taken every single time.

On the other hand, perfect sampling methods are algorithms whose running time T has a distribution that varies depending on the problem at hand. Suppose that $b(n)$ is an upper bound on the expected running time of the algorithm for all problem instances. Then

for specific problems the algorithm could take much less time then $b(n)$. This gives the algorithm an $O(b(n))$ expected running time.

This is well-known in studying other algorithms: bubble sort is well-known to be an $O(n^2)$ sorting algorithm but can take $\Theta(n)$ time on inputs that are close to being sorted. But with approximate variate generation for Markov chains with known mixing time, the algorithms created are always $\Theta(p(n))$ rather than $O(p(n))$. This can give a large speed advantage for perfect simulation methods.

The expected value of the running time for a perfect sampling algorithm can be very small. However, this hides an important question: even if the expected run time is small, what about the tail of the distribution? This is the question addressed in this article, where it is shown that for several common perfect simulation methods (including basic CFTP), it is possible to run these algorithms in such a manner that the probability that the running time is much greater than the expected time declines faster than any polynomial.

Theorem 1.1. *Consider an interruptible perfect simulation algorithm or Coupling From the Past (CFTP) with running time T. If $\mathbf{E}[T] < \infty$ and is known, it is possible to run these algorithms so that*

$$\mathbf{P}(T > k\mathbf{E}[T]) \leq 2\exp(-k(\ln 2)/4) \tag{1.4}$$

independent of the problem.

If the mean of T is unknown, it is possible to run these algorithms in such a fashion so that for $k \geq 2$,

$$\mathbf{P}(T > k\mathbf{E}[T]) \leq 4\exp(-0.186k^{\ln 2/\ln 3}) \tag{1.5}$$

independent of the problem.

Equation (1.4) follows from Lemma 2.1 in Section 2, and Lemma 5.1 in Section 5. Note (1.5) provides a slightly weaker bound on the tail of T, although both bounds decline faster than any polynomial in k. It is to be expected that (1.5) is weaker as the mean $E[T]$ is unknown in this case. The method for proving (1.5) is shown in Section 4 for interruptible methods and Section 5 for CFTP, and follows from Lemmas 4.1 and 5.3.

The next section defines perfect sampling and interruptibility, and develops the basic methods for interruptible algorithms. Section 3 analyzes the running time of interruptible methods using a known procedure, the doubling method. Section 4 develops a new procedure by expanding upon the doubling method that has much smaller tails on the running time. Section 5 shows how to apply these analyses and methods to Coupling From the Past.

2. INTERRUPTIBLE PERFECT SAMPLING

Perfect sampling protocols such as Coupling From the Past [13] (and variants), cycle popping [14], FMMR [5], and the Randomness Recycler [4] all can be used to design algorithms to generate samples from a particular target distribution. The downside of these algorithms is that the running time of the procedure is itself a random variable that can be arbitrarily large.

Definition. For a target distribution π, a perfect sampling algorithm

1. outputs random variates whose distribution is exactly π,
2. has a running time T such that $\mathbf{P}(T > t) > 0$ for any positive t.

The second part of the definition is needed to distinguish perfect sampling algorithms from algorithms that compute the normalizing constant explicitly in a deterministic fashion. Dynamic programming is an example of a protocol for creating algorithms that falls in this category, and such algorithms are not typically referred to as perfect sampling algorithms.

Example 1. A perfect simulation algorithm for a geometric random variable with parameter p.

GEO
Input: p, *Output:* X
1) **Let** $X \leftarrow 0$
1) **Repeat**
2) **Choose** $U \leftarrow \text{Unif}[0,1]$, **Let** $X \leftarrow X + 1$
3) **Until** $U \leq p$

As with **GEO**, many Monte Carlo algorithms are iterative in nature, with the time needed for one step being dominated by the generation of a pseudorandom number drawn uniformly from $[0, 1]$. Therefore it is customary to measure the running time of these algorithms by the number of uniforms that they utilize. In the example above, if T is the number of uniforms used, then $X = T$.

Definition. A perfect sampling algorithm with output X and running time T is interruptible if X has distribution π when conditioned on T.

The above example is a noninterruptible algorithm, since X conditioned on T equals T. A user running the algorithm **GEO** must commit to running the algorithm to completion with probability 1, otherwise bias will be introduced into the sample. For instance, if a user begins a run of a noninterruptible algorithm knowing that they only have time to generate one million uniforms, then the output will not be from π, but instead comes from X conditioned on $T \leq 10^6$. In the above example it is easy to determine how this changes the distribution but in more complex situations this is rarely possible.

By contrast, an interruptible perfect sampling algorithm can be aborted and restarted in the middle of a run without altering the distribution of the output. The algorithm **TIMED_ALG** explicitly aborts after a time limit.

TIMED_ALG: Interruptible perfect sampling with time limit
Input: t
1) **Run** interruptible perfect sampler for t time steps or until it finishes
2) **If** the algorithm finished, **output** the result
3) **Else output** $\perp$

Let $m(T)$ denote the smallest median of the random variable T (note that while the median might not be unique as T is integral, the set of medians will have a smallest element.) When the input to **TIMED_ALG** is $t = m(T)$, the probability that $\perp$ is output is at most $1/2$. When $m(T)$ is polynomial in the problem input size, Dyer and Greenhill [3] called such an algorithm a "strong perfect sampler."

The following algorithm breaks the interruptible algorithm into blocks of fixed length, as illustrated in Fig. 1.

Fig. 1. Blocks of fixed length.

Interruptible perfect sampling with fixed length blocks
Input: t, Output: Y
1) **Repeat**
2) **Let** Y be the result of **TIMED_ALG**(t)
3) **Until** $Y \neq \perp$

Lemma 2.1. *Let T be the running time of the original interruptible perfect sampling algorithm. Suppose $m(T)$ is known, and let T_K be the running time of the interruptible perfect sampling with $t = m(T)$. Then*

$$\mathbf{E}[T_K] \leq 2m(T) \text{ and } (\forall k \geq 0)(\mathbf{P}(T_K > k \cdot m(T)) \leq 2(1/2)^k). \tag{2.1}$$

If T has a known finite expectation then the running time of fixed length interruptible perfect sampling with $t = 2\mathbf{E}[T]$ satisfies

$$\mathbf{E}[T_K] \leq 4\mathbf{E}[T] \text{ and } (\forall k \geq 0)(\mathbf{P}(T_K > k\mathbf{E}[T]) \leq 2(1/2)^{k/2}). \tag{2.2}$$

Proof. The number of times through the repeat loop in the algorithm is a geometric random variable with parameter at least $1/2$, and each run through the loop takes $m(T)$ time. Hence the expected number of runs through the loop is at most 2, and $\mathbf{P}(T_K > km(T)) \leq (1/2)^{\lfloor k \rfloor} \leq 2(1/2)^k$. If T has finite expectation, then $m(T) \leq 2\mathbf{E}[T]$ by Markov's inequality, and the bound is $(1/2)^{\lfloor k/2 \rfloor} \leq 2(1/2)^{k/2}$ and (2.2) follows. ∎

3. INTERRUPTIBLE PERFECT SAMPLING WITH UNKNOWN RUNNING TIME

The previous section dealt with the situation where the median or expectation of the running time T was known before starting the algorithm. Unfortunately, this is rarely the case. In their original paper on Coupling From the Past, Propp and Wilson [13] suggested a way around this problem: doubling the amount of time given to each block, as shown in Fig. 2.

Their technique can be applied to interruptible perfect simulation as follows.

Fig. 2. Doubling block length.

Doubling algorithm for interruptible perfect sampling
Output: Y
1) **Let** $t \leftarrow 1$
2) **Repeat**
3) **Let** $t \leftarrow 2t$
4) **Let** Y be the result of **TIMED_ALG**(t)
5) **Until** $Y \neq \perp$

The expected running time for the doubling algorithm will be the same order as the (3/4)-quantile of T. When T has finite expectation, the doubling algorithm has expected running time of the same order as $\mathbf{E}[T]$ and the tail declines faster than any polynomial, but not exponentially fast.

Lemma 3.1. *Let T be the running time of the original interruptible perfect sampling algorithm, and T_D be the running time of the doubling algorithm for interruptible perfect sampling. Let $s(T)$ be the smallest* (3/4)*-quantile of T, so that $\mathbf{P}(T \geq s(T)) \geq 1/4$. Then*

$$\mathbf{E}[T_D] \leq 3s(T) \text{ and } (\forall k > 0)(\mathbf{P}(T_D > k \cdot s(T)) \leq 4/k^2). \tag{3.1}$$

If T has finite expectation then

$$\mathbf{E}[T_D] \leq 3\mathbf{E}[T] \text{ and } (\forall k \geq 3)(\mathbf{P}(T_D > k\mathbf{E}[T]) \leq 0.242/k^{0.721 \ln k - 2.09} \tag{3.2}$$

Proof. Let $k \geq 2$, and suppose $T > k \cdot s(T)$. Then there must be at least one block that is of size between $s(T)$ and $2s(T)$ that returned $\perp$. In fact, because the block size is doubling at each step, there will be $\lfloor \log_2 k \rfloor$ blocks of size at least $s(T)$. The probability that any one of these blocks returns $\perp$ is at most $1/4$, therefore the probability that all these blocks return $\perp$ is at most

$$(1/4)^{\lfloor \log_2 k \rfloor} \leq (1/4)^{(\log_2 k)-1} = 4/k^2, \tag{3.3}$$

so $\mathbf{P}(T_D > k \cdot s(T)) \leq 4/k^2$. For $k < 2$, $4/k^2 > 1$ and the bound also holds.

It remains to bound $\mathbf{E}[T_D]$. Since T_D is a nonnegative random variable,

$$\begin{aligned}
\mathbf{E}[T_D] &= \int_0^\infty t \, \mathrm{d}\mathbf{P}_{T_D}(t) \\
&\leq \sum_{k=1}^{\infty} k \cdot s(T)\mathbf{P}((k-1)s(T) < T_D \leq k \cdot s(T)) \\
&= s(T)\left[\sum_{k=0}^{\infty} \mathbf{P}(T_D > k \cdot s(T))\right] \\
&\leq s(T)\left[\mathbf{P}(T_D > 0) + \sum_{k=1}^{\infty} (1/4)^{\lfloor \log_2 k \rfloor}\right] \\
&\leq s(T)\left[1 + (1/4)^0 + 2(1/4)^1 + 4(1/4)^2 + \ldots\right] = 3s(T).
\end{aligned}$$

Now suppose that $\mathbf{E}[T] < \infty$. Markov's inequality provides a means to obtain a stronger upper bound. Fix $k \geq 2$ and let $t = k\mathbf{E}[T]$. Then for $T > t$, the last run between $t - 2^{\lfloor \log_2 t \rfloor}$ and t has to fail, and the previous block as well. Let $\ell_1 := t - 2^{\lfloor \log_2 t \rfloor}$ and $\ell_2 := (1/2)2^{\lfloor \log_2 t \rfloor}$

denote the lengths of these two blocks. Block size doubles at each step, and so the lengths of all the other blocks combined is exactly $\ell_2 - 1$. Hence $\ell_1 + \ell_2 + \ell_2 - 1 = t$. So either $\ell_1 \geq t/3$ or $\ell_2 \geq t/3$.

Hence there is a sequence of blocks of length at least $t/3, t/6, t/12, \ldots$. By Markov's inequality, a block of length t/a has probability at most $\mathbf{E}[T]a/t = a/k$ of returning $\perp$. So, the probability that all of the blocks return $\perp$ is at most

$$f(k) = \frac{3}{k} \cdot \frac{6}{k} \cdots \frac{3.2^{\lfloor \log_2(k/3) \rfloor}}{k}. \tag{3.4}$$

Note for integer i, when $k = 3 \cdot 2^i$, $f(k) = 2^{-(i^2+i)/2}$, and $\ln f(k) = -0.5(\ln 2)(i^2 + i)$. Since $f(k)$ decreases in k, for all k: $\ln f(k) \leq -0.5(\ln 2)((\log_2 k/6)^2 + \log_2 k/6)$, so $\ln f(k) \leq 0.5 \ln 6 - 0.5(\ln 6)^2/\ln 2 + \ln k[-\ln k/(2 \ln 2) + \ln 6/\ln 2 - 0.5]$. Exponentiating and bounding the constants yields $f(k) \leq 0.242/k^{0.721 \ln k - 2.09}$, giving the desired upper bound on $\mathbf{P}(T_D > k\mathbf{E}[T])$.

Bound $\mathbf{E}[T_D]$ by considering $\mathbf{P}(T_D > t)$. As above, $T_D > t$ implies that there is a block of size at least $t/3$. The probability that this block fails is $\mathbf{P}(T > t/3)$, so:

$$\sum_{t=0}^{\infty} \mathbf{P}(T_D > t) \leq \sum_{t=0}^{\infty} \mathbf{P}(T > t/3) = 3\mathbf{E}[T]. \tag{3.5}$$

∎

Remark. The most important property of the doubling method is that even if T does not have finite expectation, the doubling procedure will.

Remark. In Lemma 3.1, the (3/4)-quantile is utilized. In fact, any α-quantile could be used where $\alpha \in (1/2, 1)$ to achieve similar results. The closer α is to 1, the greater the polynomial decrease for the tail.

4. TRIPLING METHOD FOR INTERRUPTIBLE PERFECT SAMPLING

In the previous section, the doubling method was described for interruptible perfect sampling. The analysis shows that when T has finite expectation, the tails decline exponentially at a rate of order $(\log k)^2$. For most applications, the simplicity of the doubling method combined with this dropoff in running time will be sufficient.

It is possible to do better. In this section a new method is described that has expected running time that is of the order of α-quantiles for any α. Moreover, the tail of T declines exponentially at rate of order $k^{\ln 2/\ln 3}$. The cost is that the algorithm is somewhat more complex.

As in the previous section **TIMED_ALG**(t) will denote the interruptible algorithm which aborts and returns $\perp$ if the running time is greater than t. The goal is to divide the allotted time into three pieces. Two of these pieces are recursive copies of the original, and the last piece is just a run of **TIMED_ALG**. Because of the division by 3, this algorithm is named **3_REC**.

3_REC
Input: t (where $t = 3^d$ for some integer d) *Output:* Y
1) **If** $t = 1$
2) **Let** $Y \leftarrow$ **TIMED_ALG**(1)
3) **Else**
4) **Let** $Y \leftarrow$ **3_REC**$(t/3)$
5) **If** $Y = \perp$
6) **Let** $Y \leftarrow$ **3_REC**$(t/3)$
7) **If** $Y = \perp$
8) **Let** $Y \leftarrow$ **TIMED_ALG**$(t/3)$

This creates a block structure shown in Fig. 3. Although the block size is still increasing exponentially, there are far more smaller blocks in this structure. In a run of length t, a block of length $t/3^d$ will be repeated at most $3 \cdot 2^{d-1}$ times.

In order to be useful when the median of T is unknown, it is necessary to create this block structure in a forward fashion. After running forward 3^d steps, the next 3^d steps need an identical block structure, and then the next 3^d steps are just a straightforward run of **TIMED_ALG**. This is encoded as follows:

3_REC_FORWARD
Output: Y
1) **Let** $t \leftarrow 1$, **let** $Y \leftarrow$ **TIMED_ALG**(1)
2) **While** $Y = \perp$ do $\{$
3) **Let** $Y \leftarrow$ **3_REC**(t)
4) **If** $Y = \perp$
5) **Let** $Y \leftarrow$ **TIMED_ALG**(t)
6) **Let** $t \leftarrow 3t$ $\}$

Lemma 4.1. *Let T be the running time of the original interruptible perfect sampling algorithm, and T_3 be the running time of **3_REC_FORWARD**. Let $m(T)$ be the smallest median of T, and $\beta = \ln 2 / \ln 3 \approx 0.6309$. Then*

$$\mathbf{E}[T_3] \le 15.9 m(T) \text{ and } (\forall k \ge 0)(\mathbf{P}(T_3 > km(T)) \le 4 \exp(-0.447 k^\beta). \tag{4.1}$$

If T has finite expectation then

$$\mathbf{E}[T_3] \le 7\mathbf{E}[T] \text{ and } (\forall k \ge 0)(\mathbf{P}(T_3 > k\mathbf{E}[T]) \le 4 \exp(-0.289 k^\beta). \tag{4.2}$$

Proof. Let $k > 0$ and $t = km(T)$. Suppose that $km(T)$ is a power of 3. The recursive structure of **3_REC** is two recursive blocks followed by a block of length $(k/3)m(T)$. Let $d = \lfloor \log_3 k \rfloor$. Then $\mathbf{P}(T > km(T)) \le f(k)$, where

$$f(k) = f(k/3)^2 (1/2) \quad \text{for } k \ge 3, f(k) = 1 \text{ otherwise.} \tag{4.3}$$

Fig. 3. 3_REC block length.

Suppose $k = 3^d$, and let $g(d) = f(3^d)$, so that $g(d) = g(d-1)^2(1/2), g(0) = 1$. Taking the logarithm base 2 yields the more tractable $\log_2 g(d) = 2\log_2 g(d-1) - 1$ which has solution $\log_2 g(d) = -[2^d - 1]$. Exponentiating gives $f(3^d) = \exp(-(\ln 2)(2^d - 1)) = 2\exp(-(\ln 2)k^\beta)$.

Now suppose that for some d, $3^d \le k < 2\cdot 3^d$. So there exists some $k' = 3^d$ such that $k' > k/2$. Since f is decreasing, $f(k) \le f(3^d) = 2\exp(-(\ln 2)(k')^\beta) \le 2\exp(-(\ln 2)(k/2)^\beta) < 2\exp(-0.447k^\beta)$.

Finally, if $2\cdot 3^d \le k < 3\cdot 3^d$, then again let $k' = 3^d$ and note that $k' > k/3$. There are at least two blocks of size k', and so $f(k) \le f(k')^2 \le 2^2\exp(-(\ln 2)(k/3)^\beta)^2 < 4\exp(-0.447k^\beta)$, yielding the bound in (4.1). Note

$$\mathbf{E}[T_3] \le \sum_{k=0}^{\infty} m(T)\mathbf{P}(T_3 > km(T)) \le 15.9m(T). \tag{4.4}$$

The last inequality comes from first summing $\min\{1, 4\exp(-0.447k^\beta)\}$ for k from 0 to 99, then grouping the remaining terms in blocks of size 100, 200, 400, etc., to bound the rest of the sum by $(100)e^{-0.447(100)^\beta}/(1 - 2/3) < 0.09$.

Now suppose T has finite mean. Then $m(T) \le 2\mathbf{E}[T]$, and the bound (4.2) follows from (4.1) replacing k with $k/2$. When $t = 3^d$ the longest block is of length $3^{d-1} = t/3$. As t grows, this 3^{d-1} block remains the longest until the end block of length $t - 2\cdot 3^d$ takes over as the largest block. This happens when $t = 7\cdot 3^{d-1}$, and so the largest block at this point is $t/7$. As t grows toward 3^{d+1}, this end block becomes the largest, and grows from $1/7$ the size of t back down to $1/3$ the size of t. Hence there exists a block of length $t/7$, and this bound is tight. So

$$\mathbf{E}[T_3] = \sum_{t=0}^{\infty} \mathbf{P}(T_3 > t) \le \sum_{t=0}^{\infty} \mathbf{P}(T > t/7) = 7\mathbf{E}[T]. \tag{4.5}$$

∎

Remark. It was not necessary to use the median in the statement of the lemma. Changing the quantile used changes the constant in front of the k^β in the exponent of the bound.

4.1. Generalization

The log of the tail grows as $k^{\ln 2/\ln 3}$ in the tripling procedure, but by varying the recursive structure, the exponent of k can be made to match any number in $(0, 1)$.

```
γ_REC
Input: t,γ    Output: Y
1)  If t ≤ 1
2)     Let Y ← TIMED_ALG(t)
3)  Else
4)     Let Y ← γ_REC (⌊t(1 − γ)/2⌋)
5)     If Y = ⊥
6)        Let Y ← γ_REC (⌊t(1 − γ)/2⌋)
7)        If Y = ⊥
8)           Let Y ← TIMED_ALG(t − 2⌊(1 − γ)/2⌋)
```

When $t = 3^d$ and $\gamma = 1/3$, this block structure is exactly the same as **3_REC**. The number of blocks always doubles at each level for all γ, but the size of lower levels is about $(1-\gamma)/2$, so the exponent for t changes from $\ln 2/\ln 3$ to $\ln 2/\ln[2/(1-\gamma)]$. As γ goes to 1, the exponent goes to 0, and as γ goes to 0, the exponent goes to 1.

Having $\gamma = 0$ corresponds to the block structure where every block has the same length, and $\gamma = 1$ just uses the time allotted for 1 long block. As long as $\gamma \in (0,1)$ the essential nature of the tails remains unchanged: the tail continues to decline exponentially in a power of t, and even if T itself does not have finite expectation, the recursive algorithm will.

5. COUPLING FROM THE PAST

The most widely used perfect simulation protocol, the Coupling From the Past (CFTP) technique of Propp and Wilson [13], is not interruptible. Still, the methods of the previous section can be applied to CFTP with similar results.

CFTP works as follows. A Markov chain $\{X_t\}_{t=-\infty}^{\infty}$ on measurable state space $(\Omega, \mathcal{F})$ is usually simulated through the use of an update function $\phi : \Omega \times [0,1] \to \Omega$ where for all measurable A, and a uniform $[0,1]$ random variable U,

$$\mathbf{P}(\phi(x,U) \in A) = \mathbf{P}(X_{t+1} \in A | X_t = x). \tag{5.1}$$

Let $\ldots, U_{-2}, U_{-1}, U_0, U_1, U_2, \ldots$ be iid uniform on $[0,1]$, and define:

$$F_a^b(x) = \phi(\phi(\cdots\phi(x, U_a), \ldots, U_{b-2}), U_{b-1}), \tag{5.2}$$

so that if $X_a = x$, then $X_b = F_a^b(x)$. The simple fact underpinning CFTP is that if $F_{-t}^0(\Omega) = \{X\}$ for some $t > 0$, then $X \sim \pi$. That is, if the choices of U_i lead to the entire state space coalescing to a single state at time 0, that state is stationary.

That idea is half the CFTP algorithm. The other half is some means for determining when $|F_{-t}^0(\Omega)| = 1$. Let **CC** be such an algorithm. The input to **CC** is the number of steps in the Markov chain to be taken, $t = b - a$, and the uniforms needed for the update function $(U_a, U_{a+1}, \ldots, U_{b-1})$. Examples of mechanisms for such a determination include bounding chains [7–9] and monotonicity [13].

Given a set of uniforms and time t, if $F_{-t}^0(\Omega) = \{Y\}$, then **CC** returns either the state Y or $\perp$. If $F_{-t}^0(\Omega)$ is not a single state, then **CC** must return $\perp$. Let $\tau := \inf\{t : \mathbf{CC}(t, U_{-t}, \ldots, U_{-1}) \neq \perp\}$. Then it is desirable that runs of **CC** longer than τ also do not return $\perp$.

Definition. Call **CC** consistent if for all $t \geq \tau$, $\mathbf{CC}(t, U_{-t}, \ldots, U_{-1}) \neq \perp$.

Note that even when $F_{-t}^0(\Omega)$ is a singleton, the algorithm **CC** is not guaranteed to return that singleton. The fact that **CC** returns a state is a sufficient but not necessary condition for $|F_{-t}^0(\Omega)| = 1$.

An ideal CFTP algorithm would call **CC** at time τ and be done. The simplest implementation of CFTP checks backwards in time using blocks of fixed length.

CFTP with fixed block size
Input: t_{block}, *Output:* Y
1) **Let** $t \leftarrow 0$
2) **Repeat**
3) **Let** $t \leftarrow t + t_{block}$, **let** $Y \leftarrow \mathbf{CC}(t, U_{-t}, U_{-t+1}, \ldots, U_{-t+t_{block}-1})$
4) **Until** $Y \neq \perp$
5) **Let** $Y \leftarrow F^0_{-t+t_{block}}(Y)$.

When $E[\tau]$ or the median of τ is known, then the fixed length CFTP has a running time similar to that of interruptible algorithms in Lemma 2.1. The proof of the following lemma is nearly identical to that of Lemma 2.1, the difference being that line 5) causes the actual work of CFTP to be almost double that of a forward running interruptible algorithm.

Lemma 5.1. *Let τ be the random variable that is the optimal running time for a run of CFTP (with median $m(\tau)$), and T_K be the running time of CFTP with fixed block size $m(\tau)$. Then*

$$\mathbf{E}[T_K] \leq 4m(\tau) \text{ and } (\forall k \geq 0)(\mathbf{P}(T_K > k \cdot m(\tau)) \leq 2(1/2)^{k/2}). \tag{5.3}$$

If the median of τ is unknown but $\mathbf{E}[\tau] < \infty$ is known, then running CFTP with fixed block length $2\mathbf{E}[\tau]$ yields

$$\mathbf{E}[T_K] \leq 8\mathbf{E}[\tau] \text{ and } (\forall k \geq 0)(\mathbf{P}(T_K > k\mathbf{E}[\tau]) \leq 2(1/2)^{k/4}). \tag{5.4}$$

However, as with the interruptible algorithms, τ is generally unknown at the beginning of the procedure, and so some means of increasing the length of a block is needed. Propp and Wilson suggested the doubling method used for interruptible algorithms in Section 3. This can be implemented in several ways, the version most suitable to analysis is as follows:

Doubling CFTP
Output: Y
1) **Let** $t \leftarrow 1$
2) **Repeat**
3) **Let** $t \leftarrow 2t$, **let** $Y \leftarrow \mathbf{CC}(t/2, U_{-t}, \ldots, U_{-(t/2)-1})$
4) **Until** $Y \neq \perp$
5) **Let** $Y \leftarrow F^0_{-t/2}(Y)$

Now line 3) only runs from time $-t$ up to time $-t/2$ in CFTP. If successful, line 5) finishes the run by moving the Markov chain up to time 0. Because the number of steps is doubling at each level, if $-t$ to $-t/2$ is the last block considered, the total number of steps taken by the algorithm will be (for $t > 1$) equal to $(3/2)t$. The proof of the following lemma is exactly the same as for Lemma 3.1 with the $3/2$ factor taken into account. These bounds are equivalent to replacing k by $(2/3)k$.

Lemma 5.2. *Let τ be the optimal running time for CFTP with a consistent **CC**, and T_D be the running time of the doubling algorithm for CFTP Let $s(\tau)$ be the smallest* (3/4)*-quantile of τ, so that $\mathbf{P}(T \geq s(\tau)) \geq 1/4$. Then*

$$\mathbf{E}[T_D] \leq 4.5s(\tau) \text{ and } (\forall k > 0)(\mathbf{P}(T_D > k \cdot s(\tau)) \leq 9/k^2). \tag{5.5}$$

Time (in base 3)	10100	10200
Next Block Size (in base 3)	00001	00100

Fig. 4. The next block size given the current time.

If τ has finite expectation then

$$\mathbf{E}[T_D] \le 4.5\mathbf{E}[\tau] \text{ and } (\forall k \ge 4.5)(\mathbf{P}(T_D > k\mathbf{E}[\tau]) \le 0.092/k^{0.721 \ln k - 2.68}. \tag{5.6}$$

As before, the tripling algorithm with recursive structure can be used to give a tail that is exponential in a polynomial in k. For CFTP, it is helpful given the current time t, to know the next block size s. From the recursive description, it is possible to build a method for finding the next block size.

When t is written in base 3, the next block size will be 1 if the least significant nonzero digit is 1, and if this digit is 2, the next block size has this digit 1 and all digits of less significance 0. (See Fig. 4).

The following code calculates this value.

Block_Size
Input: t Output: s
1) **Let** $b \leftarrow 1/3$
2) **Repeat**
3) **Let** $b \leftarrow 3b$
4) **Let** a be t/b modulo 3
5) **Until** $a \neq 0$
6) **Let** $s \leftarrow \mathbf{1}(a = 1) + (3b)\mathbf{1}(a = 2)$

Now CFTP is run as before, only the next block size comes from **Block_Size** rather than simple doubling.

Tripling CFTP
Output: Y
1) **Let** $t \leftarrow 0$
2) **Repeat**
3) **Let** $s \leftarrow$ **Block_Size**(t)
4) **Let** $t \leftarrow t + s$
4) **Let** $Y \leftarrow \mathbf{CC}(s, U_{-t}, \ldots, U_{-t+s-1})$
5) **Until** $Y \neq \perp$
6) **Let** $Y \leftarrow F^0_{-t+s-1}(Y)$

The length of the final block that does not return $\perp$ might be arbitrarily small compared to the the $t + s$ steps taken in line 6). Hence the total running time could be close to 2 times the final value of t. This modifies the analysis in Lemma 4.1 to give the following.

Lemma 5.3. *Let τ be the optimal running time for CFTP with a consistent **CC**, and T_3 be the running time of the tripling algorithm for CFTP. Let $m(\tau)$ be the smallest median of τ, and $\beta = \ln 2/\ln 3 \approx 0.6309$. Then*

$$\mathbf{E}[T_3] \le 31.8m(\tau) \text{ and } (\forall k \ge 0)(\mathbf{P}(T_3 > k \cdot m(\tau)) \le 4\exp(-0.289k^{\beta}). \tag{5.7}$$

If τ has finite expectation then

$$\mathbf{E}[T_3] \le 14\mathbf{E}[\tau] \text{ and } (\forall k \ge 0)(\mathbf{P}(T_3 > k\mathbf{E}[\tau]) \le 4\exp(-0.186k^{\beta}). \tag{5.8}$$

5.1. Reducibility

Often **CC** has a property that makes the doubling algorithm alone run much faster. For $t_1 < t_2 < t_3$, $\mathbf{P}(F_{-t_3}^{-t_1}(\Omega) = \{X\}) \le \mathbf{P}(F_{-t_3}^{-t_2}(\Omega) = \{X\})\mathbf{P}(F_{-t_2}^{-t_1}(\Omega) = \{X\})$. The desire to have a similar property for **CC** motivates the following definition.

Definition. **CC** is reducible if for all $t_1 < t_2 < t_3$, $\mathbf{P}(A) \le \mathbf{P}(B)\mathbf{P}(C)$, where A, B, and C are the events that $\{\mathbf{CC}(t_3 - t_1, U_{-t_3}, \ldots, U_{-t_1-1}) \ne \perp\}$, $\{\mathbf{CC}(t_3 - t_2, U_{-t_3}, \ldots, U_{-t_2-1}) \ne \perp\}$, and $\{\mathbf{CC}(t_2 - t_1, U_{-t_2}, \ldots, U_{-t_1-1}) \ne \perp\}$, respectively.

In other words, **CC** is reducible if when a single block is broken into two subblocks, then the chance of failure on the large block is bounded above by the chance that both subblocks fail.

An example of a reducible **CC** method is monotonic CFTP, which was one of the first methods proposed by Propp and Wilson [13]. Suppose that the state space Ω has a partial order $\preceq$, a minimum state $x_{\min}$ and maximum state $x_{\max}$. Then the update function ϕ is monotonic if for all $u \in [0, 1]$, $x \preceq y \to \phi(x, u) \preceq \phi(y, u)$.

To determine if $F_a^b(\Omega)$ has collapsed to a single state with monotonic updates is easy: simply see if $F_a^b(x_{\max}) = F_a^b(x_{\min})$. If they are equal, then every other state is squeezed between them down to the single state. And if they are not equal then $F_a^b(\Omega)$ is not a single state. Therefore monotonicity is one of the rare situations where it is possible to exactly determine if $|F_a^b(\Omega)| = 1$ for a given update function. Because a monotonic chain keeps track of F_a^b exactly, **CC** is immediately reducible.

Running CFTP with a reducible **CC** yields exponential tails with the simple doubling scheme.

Lemma 5.4. *Let τ be the optimal running time for CFTP, and T be the running time of **CFTP** using a reducible **CC** where at stage d, $\mathbf{CC}(2^{d-1}, U_{-2^d}, \ldots, U_{-2^{d-1}-1})$ is evaluated. Let $m(\tau)$ be the smallest median of τ. Then*

$$\mathbf{E}[T] = 8m(\tau) \text{ and } (\forall k > 0)(\mathbf{P}(T > k \cdot m(\tau)) \le 2\exp(-(\ln 2)k/4) \tag{5.9}$$

If τ has finite expectation then

$$\mathbf{E}[T] \le 4\mathbf{E}[\tau] \text{ and } (\forall k \ge 0)(\mathbf{P}(T > k\mathbf{E}[\tau]) \le \exp(1 - e^{-1}k/4). \tag{5.10}$$

Proof. Let $k \ge 0$. Then for T to be greater than $km(\tau)$, a run of length at least $(k/4)m(\tau)$ must have failed since τ will be at most twice the length of the largest block, and $T \le 2\tau$. Because the **CC** algorithm being used is reducible, this large block can be broken into $\lfloor (k/4) \rfloor$ blocks of length $m(\tau)$. Each of these blocks has a $1/2$ chance of failure, hence the total probability of failure is at most

$$\mathbf{P}(T > k \cdot m(\tau)) \le (1/2)^{\lfloor (k/4) \rfloor} \le 2\exp(-(\ln 2)k/4). \tag{5.11}$$

So $\mathbf{E}[T] \le m(\tau) \sum_{k=0}^{\infty} \mathbf{P}(T > k \cdot m(\tau)) \le m(\tau) \sum_{k=0}^{\infty} (1/2)^{\lfloor k/4 \rfloor} = 8m(\tau)$.

When τ has finite expectation, then the longest block is at most 2τ, and T is twice the size of the longest block, so $\mathbf{E}[T] \leq 4\mathbf{E}[\tau]$. Markov's inequality means that the probability that a block of length $\alpha\mathbf{E}[\tau]$ fails is at most $1/\alpha$. Putting that into the argument for $m(\tau)$ yields

$$\mathbf{P}(T > k \cdot \mathbf{E}[\tau]) \leq \alpha \exp(-(\ln \alpha)\alpha^{-1}k/4). \tag{5.12}$$

Setting $\alpha = e$ maximizes $(\ln \alpha)/\alpha$, and yields the rest of (5.10). ∎

There are ways other than monotonicity for creating a reducible **CC**. For example, the method of bounding chains (see [8] for details) will generate **CC** algorithms with this property as well.

6. INTERRUPTIBILITY

The reason why the exponential tails are so important for CFTP is because CFTP is not an interruptible algorithm. This introduces an unknown amount of bias to the samples based on the (typically unknown or unacknowledged) user impatience. The exponential tail is helpful in that the user impatience is unlikely to have a large impact as long as the user commits to running at least a constant times the expected running time.

Dyer and Greenhill [3] noted that any rapidly mixing Markov chain can be turned into a noninterruptible exact sampling algorithm. Without going into details, the running time of their method is polynomial $p(n)$ with extremely high probability, and exponential $x(n)$ with extremely small probability. (Here n measures the size of the problem input.) As long as the $x(n)$ branch is taken with small enough probability, the overall expected running time will be polynomial.

Of course, in practice if the $x(n)$ branch is taken, the user would actually abort the procedure. This is an example of an algorithm design where user interruption is expected to happen. This is in sharp contrast to CFTP, where user interruption is typically never expected to happen by the user. This is usually an unjustified assumption on the part of the user since in reality a CFTP algorithm could be interrupted at any time.

However, the exponential declining tail on the running time for CFTP explains why with practitioners this has never been an issue: with the doubling algorithm and a reducible **CC** algorithm a user could run millions of iterations and still be extremely unlikely to have to double more than a few times more than average.

ACKNOWLEDGEMENTS

The author thanks Wilfrid Kendall for several helpful discussions, and the anonymous referees for some helpful comments.

REFERENCES

[1] P. Diaconis and B. Efron, Testing for independence in a two-way table: New interpretations of the chi-square statistic, Ann Statist 13 (1985), 845–913.

[2] P. Diaconis and D. Stroock, Geometric bounds for eigenvalues of Markov chains, Ann Appl Probab 1 (1991), 36–61.

[3] M. Dyer and C. Greenhill, "Random walks on combinatorial objects," Surveys in combinatorics, J. D. Lamb and D. A. Preece (Editors), London Math Soc Lecture Note Ser 267, Cambridge University Press, Cambridge, 1999.

[4] J. A. Fill and M. L. Huber, The randomness recyler: A new approach to perfect sampling, Proc of 41st Symp Foundations of Computer Science, 2000, pp. 503–511.

[5] J. A. Fill, M. Machida, D. J. Murdoch, and J. S. Rosenthal, Extension of Fill's perfect rejection sampling algorithm to general chain, Random Structures Algorithms 17 (2000), 290–316.

[6] G. S. Fishman, Choosing sample path length and number of sample paths when starting in the steady state, Oper Res Lett 16 (1994), 209–219.

[7] O. Häggström and K. Nelander, On exact simulation from Markov random fields using coupling from the past, Scand J Statist 26 (1999), 395–411.

[8] M. Huber, Perfect sampling using bounding chains, Ann Appl Probab 14 (2004), 734–753.

[9] M. L. Huber, Exact sampling and approximate counting techniques, Proc 30th Symp Theory of Computing, 2000, pp. 31–40.

[10] M. Jerrum and A. Sinclair, Approximating the permanent, J Comput 18 (1989), 1149–1178.

[11] M. Jerrum and A. Sinclair, The Markov chain Monte Carlo method: An approach to approximate counting and integration, PWS, Boston, 1996.

[12] M. Newman and G. Barkema, Monte Carlo methods in statistical physics, Oxford University Press, New York, 1999.

[13] J. G. Propp and D. B. Wilson, Exact sampling with coupled Markov chains and applications to statistical mechanics, Random Structures Algorithms 9 (1996), 223–252.

[14] J. G. Propp and D. B. Wilson, How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph, J Algorithms 217 (1998), 170–217.