Estimating the Number of *s-t* Paths in a Graph

Ben Roberts[1], Dirk P. Kroese[2]

[1] Department of Pure Mathematics and Mathematical Statistics, University of Cambridge, Cambridge CB3 0WB, United Kingdom
[2] Department of Mathematics, The University of Queensland, Brisbane 4072, Australia

Abstract. The problem of counting the number of *s-t* paths in a graph is #P-complete. We provide an algorithm to estimate the solution stochastically, using sequential importance sampling. We show that the method works effectively for both graphs and digraphs. We also use the method to investigate the expected number of *s-t* paths in a random graph of size n and density d, and develop a model that shows how this quantity behaves when n and d are varied.

Key words. Counting *s-t* paths, #P-completeness, random graphs, Monte Carlo simulation, sequential importance sampling.

1. Introduction

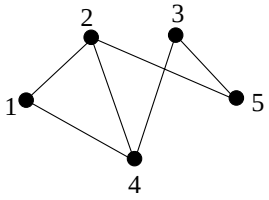
Many difficult counting problems belong to the class of $\#P$-complete problems [11,18]. One such problem is counting the number of *s-t* paths in a graph, which was proved to be $\#P$-complete in [17]. Although it is trivial to establish whether there *exists* an *s-t* path in an arbitrary graph, to date no efficient algorithm exists for *counting* the number of such paths for general graphs. The purpose of this paper is to introduce a simple Monte Carlo method for fast and accurate estimation of the number of *s-t* paths for general graphs, and to apply this to find an expression for the expected number of such paths in a random graph as a function of the order and density of the graph. The method uses the principle of *importance sampling*, a fundamental variance reduction technique in statistics, which has been successfully applied to a great variety of estimation problems, see for example [5,6,10,3,16,9]. Although the idea of using importance sampling for counting goes as far back as [4] and [13], who developed algorithms for generating and counting self-avoiding random walks, there has been renewed interest in this area following recent developments in sequential and adaptive Monte Carlo techniques, such as sequential importance sampling [1], the cross-entropy method

[15] and importance resampling, see e.g., [12]. Various importance sampling approaches to combinatorial counting problems, such as counting the number of zero-one tables, counting the permanent of a binary matrix, and determining the number of Hamiltonian cycles, are discussed in [7] and [14]. Other stochastic algorithms for counting problems can be found in [19], who provide an algorithm to estimate the maximum number of node-independent paths between two vertices of a graph, and in [2], who estimate the number of Hamiltonian cycles in dense graphs.

The rest of the paper is organized as follows: In Section 2 we discuss the main ideas behind counting via importance sampling, and in Section 3 we formulate our main algorithm. Several test cases are provided in Section 4 (and further specified in the appendix) to support the accuracy of the method, especially for high-density graphs. We also discuss a number of simple modifications to the algorithm when dealing with graphs of lower density. In Section 5 we use the method to investigate the expected number of s-t paths in random graphs as a function of the size (order) n and density d of the graph. Finally, in Section 6 we summarize our findings and discuss possible directions for future research.

2. Counting by Estimation

Consider a graph G of order n. We wish to determine/estimate the number of s-t paths in the graph. Let $\mathscr{X}^*$ be the set of all such paths. Our objective is thus to find $|\mathscr{X}^*|$. We use an arbitrary vertex labeling $\{1, \ldots, n\}$, and assume that the source vertex s is labeled as 1, and the termination vertex t is labeled as n. We denote by A the adjacency matrix of G. As an example, in the graph G of order 5 shown in Figure 1, we have $\mathscr{X}^* = \{(1,2,5),\ (1,4,2,5),\ (1,4,3,5),\ (1,2,4,3,5)\}$, and $|\mathscr{X}^*| = 4$.



$$A = \begin{pmatrix} 0\,1\,0\,1\,0 \\ 1\,0\,0\,1\,0 \\ 0\,0\,0\,1\,1 \\ 1\,1\,1\,0\,0 \\ 0\,1\,1\,0\,0 \end{pmatrix}$$

Fig. 1. There are four 1-5 paths in the graph with adjacency matrix A

Instead of determining $|\mathscr{X}^*|$ exactly, e.g., by total enumeration — which is only feasible for very small graphs —, one can try to estimate $|\mathscr{X}^*|$ via Monte Carlo simulation. This can be done by sampling objects $\mathbf{x}$ from a larger set $\mathscr{X} \supseteq \mathscr{X}^*$ via some probability mass function (pmf) g, where $g(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathscr{X}^*$. With N independent copies, $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$, the quantity $|\mathscr{X}^*|$ can be

estimated using the estimator

$$\widehat{|\mathscr{X}^*|} = \frac{1}{N}\sum_{i=1}^{N}\frac{I\{\mathbf{X}^{(i)} \in \mathscr{X}^*\}}{g(\mathbf{X}^{(i)})}\,, \tag{1}$$

where $I\{\mathbf{X}^{(i)} \in \mathscr{X}^*\}$ is an indicator variable — that is, it is 1 if the expression within the brackets is true, and 0 otherwise. To see that (1) is an unbiased estimator, observe that the expectation of $I\{\mathbf{X}^{(i)} \in \mathscr{X}^*\}/g(\mathbf{X}^{(i)})$ is given by

$$\sum_{\mathbf{x}^{(i)}\in\mathscr{X}}\frac{I\{\mathbf{x}^{(i)} \in \mathscr{X}^*\}}{g(\mathbf{x}^{(i)})}\,g(\mathbf{x}^{(i)}) = \sum_{\mathbf{x}^{(i)}\in\mathscr{X}} I\{\mathbf{x}^{(i)} \in \mathscr{X}^*\} = |\mathscr{X}^*|\,.$$

The challenge is to find a pmf g that is easy to sample from and that introduces low variance to the estimator (1). The theoretically best pmf is the uniform pmf g^* on $\mathscr{X}^*$ — which generates only $\mathbf{x} \in \mathscr{X}^*$, and each of these with equal probability. However, obtaining g^* requires knowledge of the unknown $|\mathscr{X}^*|$, and is therefore as difficult as the original problem. An alternative approach is to choose a pmf g that is "close" to the uniform pmf g^* in some sense. This is where *sequential* Monte Carlo methods are often useful. Many of these procedures are carried out in combination with importance sampling techniques, and such methods are frequently referred to as *sequential importance sampling* (SIS) methods; see e.g., [8].

With respect to (1) the sequential approach means that paths $\mathbf{x}$ are generated in a sequential manner. That is, assuming that $x_1 = 1$, the second vertex x_2 is drawn from some pmf $g_2(x_2 \mid x_1)$, then, given x_1 and x_2, the third vertex x_3 is drawn from a conditional pmf $g_3(x_3 \mid x_1, x_2)$, and so on, so that eventually

$$g(\mathbf{x}) = g_2(x_2 \mid x_1)g_3(x_3 \mid x_1, x_2)\cdots g_m(x_m \mid x_1, \cdots, x_m)\,. \tag{2}$$

Note that $g_1(x_1) = 1$, as we always choose x_1 to be the vertex 1.

3. Main Algorithm

In the following methods, the adjacency matrix A is heavily relied upon. We start with a "naive" method for generating a path $\mathbf{x}$, which can be used to gain information about $\mathscr{X}^*$. It works simply by starting at vertex 1, and at each step choosing a random available vertex that is adjacent to the previous one. Note that every path from vertex 1 to n can be generated in this manner, so $g(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathscr{X}^*$. Let $\mathcal{V} = \{1, \ldots, n\}$ be the vertex set of G.

Algorithm 1 (Naive path generation)

1. Start with $x_1 = 1$, let $c = 1$ (current vertex), $g = 1$ (likelihood) and $t = 1$ (counter).
2. Set $A(\cdot, 1) = 0$ to ensure that the path will not return to 1.
3. Let $\mathcal{V}'$ be the set of possible vertices for the next step of the path. ($\mathcal{V}' = \{i \in \mathcal{V} \mid A(c, i) = 1\}$.) If $\mathcal{V}' = \emptyset$, we do not generate a valid path, so stop.
4. Choose the next vertex $i \in \mathcal{V}'$ of the path randomly using the uniform distribution on $\mathcal{V}'$. So $x_{t+1} = i$.
5. Set $c = i$, $g = g/|\mathcal{V}'|$, $A(\cdot, i) = 0$, and $t = t + 1$.
6. If $c = n$, then stop. Otherwise return to step 3.

After having generated independently N (partial) paths in this way one can take the outcome of (1) as an estimate of the total number of 1-n paths. Note, however, that the naive generation method is very biased towards generating paths of shorter length. To reduce this bias we introduce next the following *length-distribution* method.

Let the length of a path $\mathbf{x}$ be denoted by $|\mathbf{x}|$. Note that this is not the length of the vector, but rather one less than it. We first simulate a pilot run of N' samples using the "naive" method to find an estimate of the *length-distribution vector* $\boldsymbol{l} = (l_1, \ldots, l_{n-1})$, where

$$l_k = \frac{\text{number of paths of length } k}{\text{number of paths of length} \geqslant k \text{ in which } A(x_k, n) = 1}\,. \tag{3}$$

We can think of l_k in the following way: Suppose a 1-n path $\mathbf{x}$ is chosen at random from $\mathscr{X}^*$, and suppose that for some k, $A(x_k, n) = 1$, then we would expect x_{k+1} to be the vertex n with probability l_k. We estimate $\boldsymbol{l}$ using the Crude Monte Carlo (CMC) estimator

$$\widehat{l}_k = \frac{\sum_{i=1}^{N'} \frac{I\{|\mathbf{X}^{(i)}|=k\}\, I\{\mathbf{X}^{(i)} \in \mathscr{X}^*\}}{g(\mathbf{X}^{(i)})}}{\sum_{i=1}^{N'} \frac{I\{|\mathbf{X}^{(i)}|\geqslant k\}\, I\{\mathbf{X}^{(i)} \in \mathscr{X}^*\}\, I\{A(X_k^{(i)}, n)=1\}}{g(\mathbf{X}^{(i)})}}\,. \tag{4}$$

We then use $\widehat{\boldsymbol{l}} = (\widehat{l}_1, \ldots, \widehat{l}_{n-1})$ to generate paths similarly to the "naive" method, except that at each step t where $A(x_t, n) = 1$, we choose the next vertex to be n with probability $\widehat{l}_t$, and choose a different random available vertex with probability $1 - \widehat{l}_t$. If there are no other available vertices we just choose the next vertex to be n. To ensure that $g(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathscr{X}^*$, it is sufficient that $0 < \widehat{l}_k < 1$, for all k. If we do calculate $\widehat{l}_k = 0$ or 1 for some k, then we just replace it with $1/\widehat{|\mathscr{X}_1^*|}$ or $1 - 1/\widehat{|\mathscr{X}_1^*|}$ respectively, where $\widehat{|\mathscr{X}_1^*|}$ is the estimate of $|\mathscr{X}^*|$ from the pilot run.

Algorithm 2 (Length-distribution Method)

1. Simulate a pilot run of N' samples using the "naive" method to find an estimate $\widehat{\boldsymbol{l}}$ of the length distribution $\boldsymbol{l}$ using (4).
2. Start with $x_1 = 1$, so let $c = 1$ (current vertex), $g = 1$ (likelihood) and $t = 1$ (counter).
3. Set $A(\cdot, 1) = 0$ to ensure that the path will not return to 1.
4. If $A(c, n) = 0$, go to step 5. If $A(c, n) = 1$ and n is the only available vertex adjacent to c, set $x_{t+1} = n$ and stop. If there are other vertices adjacent to c, choose the next vertex to be n with probability $\widehat{l}_t$. If this happens, set $x_{t+1} = n$, $g = g \times \widehat{l}_t$ and stop. Otherwise, set $g = g \times (1 - \widehat{l}_t)$, $A(c, n) = 0$ and continue with step 5.
5. Let $\mathcal{V}'$ be the set of possible vertices for the next step of the path. ($\mathcal{V}' = \{i \in \mathcal{V} \mid A(c, i) = 1\}$.) If $\mathcal{V}' = \emptyset$, we do not generate a valid path, so stop.
6. Choose the next vertex $i \in \mathcal{V}'$ of the path randomly using the uniform distribution on $\mathcal{V}'$. So $x_{t+1} = i$.
7. Set $c = i$, $g = g/|\mathcal{V}'|$, $A(\cdot, i) = 0$, and $t = t + 1$. Return to step 4.

Final Step: Lastly, we estimate $|\mathscr{X}^*|$, using (1) and the values of g, for each valid path generated.

4. Test Cases

We tested the length-distribution method on 6 random graphs and digraphs of differing sizes and densities. Descriptions of these are shown in Table 1 and the adjacency matrices are given in Appendix A. All programs were written in Matlab, and were run on an AMD Athlon 2600 with 512MB of RAM. When feasible we also calculated the exact answer using a systematic counting algorithm. For each test graph, the program to find the exact answer counted the paths at a rate between 3000 and 20000 paths per second. There are too many paths in Case 4 and Case 6 to count in this fashion.

Table 1. Test graphs

Case	n	Graph/Digraph	Density	$\lvert\mathscr{X}^*\rvert$
1	8	graph	0.8	397
2	12	digraph	0.95	4,959,864
3	16	digraph	0.4	138,481
4	20	graph	0.85	NA
5	24	graph	0.2	1,892,724
6	30	digraph	0.8	NA

The results for the length-distribution method on the 6 test graphs are shown in Table 2, using a pilot run of $N' = 5000$ samples, and an estimation run of

$N = 10000$ samples. We show results of three runs of the program for each graph to illustrate the consistency of the estimates. We estimate the relative error (RE) of the estimate $|\mathscr{X}^*|$ using the following:

$$\widehat{\mathrm{RE}} = \frac{s_Y}{\sqrt{N}\,\widehat{|\mathscr{X}^*|}}\,, \tag{5}$$

where $Y = I\{\mathbf{X} \in \mathscr{X}^*\}/g(\mathbf{X})$ and s_Y is the sample standard deviation of Y. The solution frequency is the proportion of valid 1-n paths generated out of the N samples.

Table 2. Results for "length-distribution" method on test graphs, $N' = 5000$, $N = 10000$

Case	Description	$\widehat{\lvert\mathscr{X}^*\rvert}$	$\widehat{\mathrm{RE}}$	time(sec)	sol'n freq
1	$n = 8$, graph, high density	396.6	0.0067	1.4	84.1%
	$\lvert\mathscr{X}^*\rvert = 397$	396.0	0.0066	1.4	84.9%
		398.1	0.0065	1.4	84.9%
2	$n = 12$, digraph, high density	4.993×10^6	0.0045	2.6	93.7%
	$\lvert\mathscr{X}^*\rvert = 4,959,864$	4.960×10^6	0.0045	2.5	94.0%
		4.966×10^6	0.0044	2.6	93.4%
3	$n = 16$, digraph, low density	1.36×10^5	0.024	2.8	77.1%
	$\lvert\mathscr{X}^*\rvert = 138,481$	1.38×10^5	0.029	2.8	77.7%
		1.38×10^5	0.032	2.8	76.6%
4	$n = 20$, graph, high density	9.532×10^{14}	0.0059	4.9	88.8%
		9.504×10^{14}	0.0060	4.9	88.9%
		9.495×10^{14}	0.0059	4.9	89.0%
5	$n = 24$, graph, low density	1.85×10^6	0.040	4.6	29.4%
	$\lvert\mathscr{X}^*\rvert = 1,892,724$	1.92×10^6	0.042	4.6	29.6%
		1.98×10^6	0.043	4.6	29.5%
6	$n = 30$, digraph, high density	1.463×10^{27}	0.0093	8.1	92.0%
		1.438×10^{27}	0.0093	8.4	91.9%
		1.471×10^{27}	0.0096	8.5	92.3%

We can see that this method does indeed provide accurate estimates for $|\mathscr{X}^*|$ in all of the test graphs for which we know the true value. We can also see that it produced significantly less error in the estimates for $|\mathscr{X}^*|$ in the graphs with higher density, and that the error is not very dependent on the size of the graph. This is good because we know that as the graphs get larger, the error does not get out of control.

There are various ways in which one can further improve the efficiency of the algorithm when dealing with graphs of lower density, such as in Cases 3 and 5. We next discuss a few common-sense modifications.

One of the main sources of error in Case 5 is the fact that valid paths are generated only about 30% of the time. By inspecting the adjacency matrix (see Appendix A), we see that this is because there are only 2 vertices that are adjacent to the termination vertex n. If during the generation of a path both of these vertices are included without vertex n immediately following the last one included, one cannot generate a valid path. To avoid this happening one can introduce a condition in the algorithm that *forces the path to finish* if there are no other available vertices adjacent to vertex n. This makes a significant difference to the solution frequency, increasing it to about 70% for Case 5, which reduces the error significantly. However, this is arguably less efficient for the graphs of higher density, as the error is often not reduced enough to offset the extra computation time involved.

Another simple modification is to consider both *backwards and forward* paths. Namely, for any ordinary graph (not a digraph), the number of paths from 1 to n is the same as the number of paths from n to 1. Therefore, if it makes it easier to generate paths, one should generate them backwards starting at n. Generally, this happens if $\deg(1) > \deg(n)$. The reason for this is that it is easier to end up at a vertex of higher degree that one of lower degree.

Finally, in a case such as Case 5 where there are only 2 vertices adjacent to vertex n (vertices 16 and 23), it is often more efficient to *split the problem* into two, counting the paths from 1 to 16 and from 1 to 23, in the graph $G - \{n\}$. We then sum the two estimates to find an estimate for the number of paths from 1 to n.

5. Random Graphs

We can use the previous algorithm to investigate the behaviour of $|\mathscr{X}^*|$ for random graphs of various size and density. Let Z_G be the number of 1-n paths in a random graph G of given size n and density d. Note that Z_G is a random variable. We are interested in finding both the mean and variance of Z_G. Let $\mathscr{Z}_{n;d} = \mathbb{E}[Z_G]$.

5.1. The Complete Graph K_n

It is easy to investigate the behaviour of $\mathscr{Z}_{n;1}$ for varying n, as any random graph of size n and density 1 is the complete graph K_n. It is not difficult to see that the number of paths between any two vertices of K_n is

$$K(n) = \sum_{k=0}^{n-2} \frac{(n-2)!}{k!} = \mathscr{Z}_{n;1}\,. \tag{6}$$

Note that

$$K(n) = (n-2)!e(1+o(1))\,. \tag{7}$$

5.2. Estimating $\mathcal{Z}_{n;d}$

Algorithm 2 gives us a method to efficiently estimate $\mathcal{Z}_{n;d}$ for any given n and d. We take our samples $\{Y^{(i)}\}$ to be estimates of $\{Z_{G^{(i)}}\}$ for a number of random graphs of appropriate size and density. We then take the mean of the $\{Y^{(i)}\}$ as our estimate of $\mathcal{Z}_{n;d}$.

Random Graphs: It is important to specify exactly what we mean by a random graph of size n and density d. We do not simply include each possible edge in the graph independently with probability d, as this adds unneeded variance. Instead, we include the precise number of edges required for a density of d. As there are $\frac{1}{2}\,n\,(n-1)$ possible edges in a graph of size n, we include $E = \frac{1}{2}d\,n\,(n-1)$ edges in the graph, chosen randomly. However, this will often not be an integer. We do not simply include $[E]$ edges, because the density of the resulting graph might not be d. Instead, we include $\lfloor E+U \rfloor$ edges, where U is a uniform random number on the interval $[0,1)$. The reason for this is that the *expected* density of the resulting graph is exactly d.

Length Distribution Vector: It is not efficient to generate a different length distribution vector for each random graph, as most of them will be very similar. Instead, we want to generate an "overall" length distribution vector. We do this simply by defining $\boldsymbol{l}$ as follows:

$$l_k = \mathbb{E}\left[\frac{\text{number of paths of length } k}{\text{number of paths of length} \geqslant k}\right]. \tag{8}$$

From a pilot run of the "naive" method over N_1' random graphs, with N_2' samples for each graph, we estimate $\boldsymbol{l}$ by

$$\widehat{l}_k = \frac{1}{N_1'}\sum_{i=1}^{N_1'}\left(\frac{\sum_{j=1}^{N_2'}\frac{I\{\mathbf{X}^{(ij)}\in\mathscr{X}^*_{G^{(i)}}\}I\{|\mathbf{X}^{(ij)}|=k\}}{g(\mathbf{X}^{(ij)})}}{\sum_{j=1}^{N_2'}\frac{I\{\mathbf{X}^{(ij)}\in\mathscr{X}^*_{G^{(i)}}\}I\{|\mathbf{X}^{(ij)}|\geqslant k\}}{g(\mathbf{X}^{(ij)})}}\right), \tag{9}$$

where $\mathscr{X}^*_{G^{(i)}}$ is the set of valid 1-n paths in the graph $G^{(i)}$.

Algorithm 3 (Estimating $\mathcal{Z}_{n;d}$)

1. Simulate a pilot run of the "naive" method over N_1' random graphs of size n and density d, with N_2' samples for each graph, and find an estimate $\widehat{\boldsymbol{l}}$ of the length distribution vector $\boldsymbol{l}$ using (9).
2. Simulate the "length-distribution" method over N_1 random graphs of size n and density d, with N_2 samples for each graph, using the length distribution vector $\widehat{\boldsymbol{l}}$.

3. Estimate $\mathscr{Z}_{n;d}$ by

$$\widehat{\mathscr{Z}}_{n;d} = \frac{1}{N_1}\sum_{i=1}^{N_1} Y^{(i)}\,, \tag{10}$$

$$Y^{(i)} = \frac{1}{N_2}\sum_{j=1}^{N_2} \frac{I\{\mathbf{X}^{(ij)} \in \mathscr{X}^*_{G^{(i)}}\}}{g(\mathbf{X}^{(ij)})}\,. \tag{11}$$

5.3. Estimating Var(Z_G)

We have an estimate (10) for $\mathbb{E}[Z_G] = \mathscr{Z}_{n;d}$, but we are also interested in the variance of Z_G. Let Y be an estimate of Z_G in a random graph G. Then,

$$\begin{aligned}\mathrm{Var}(Y) &= \mathbb{E}[\mathrm{Var}(Y \mid G)] + \mathrm{Var}(\mathbb{E}[Y \mid G])\\ &= \mathbb{E}[\mathrm{Var}(Y \mid G)] + \mathrm{Var}(Z_G)\,.\end{aligned}$$

We can estimate $\ell_1 = \mathrm{Var}(Y)$ and $\ell_2 = \mathbb{E}[\mathrm{Var}(Y \mid G)]$ from our previous simulation as follows:

$$\widehat{\ell_1} = \frac{1}{N_1 - 1}\sum_{i=1}^{N_1}(Y^{(i)} - \widehat{\mathscr{Z}}_{n;d})^2, \tag{12}$$

$$\widehat{\ell_2} = \frac{1}{N_1(N_2-1)}\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\left(\frac{I\{\mathbf{X}^{(ij)} \in \mathscr{X}^*_{G^{(i)}}\}}{g(\mathbf{X}^{(ij)})} - Y^{(i)}\right)^2. \tag{13}$$

Therefore, we can estimate $\mathrm{Var}(Z_G)$ by

$$\widehat{\mathrm{Var}(Z_G)} = \widehat{\ell_1} - \widehat{\ell_2}\,. \tag{14}$$

5.4. Results

We ran the previous algorithm over varying values of n and d, n ranged from 11 to 40, and d ranged from 0.1 to 1, with a step size of 0.1. For each combination of n and d, we ran a simulation of Algorithm 3 with $(N_1', N_2', N_1, N_2) = (50, 100, 500, 100)$. Figure 2 shows the estimates of $\log(\mathscr{Z}_{n;d})$ for these simulations.

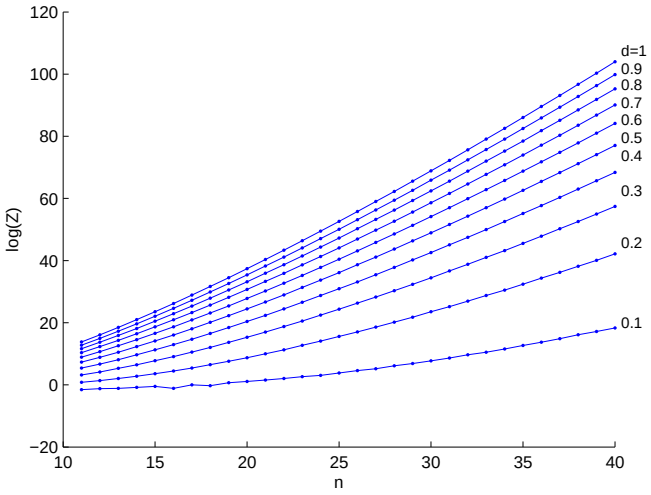


Fig. 2. $\log(\widehat{\mathcal{Z}}_{n;d})$ for varying values of n and d. The contours represent the different values of d. $d = 0.1$ is the lowest contour, ranging up to $d = 1$ for the highest contour. The contour representing $d = 1$ contains the exact values of $\log(\mathcal{Z}_{n;1})$, using (6).

For $d \geqslant 0.2$, we can see very consistent behaviour showing that our estimates are accurate. However for $d = 0.1$, slightly inconsistent behaviour for low n shows that our estimates might not be as accurate. This is supported by the estimates for the relative errors of the $\{\widehat{Z}_{n;d}\}$ shown in Appendix B. Using (14), we obtain estimates for the relative standard deviations of Z_G for each (n, d). These are also given in Appendix B.

5.5. Modeling the Data

To help construct a model for the data, consider the following analysis for large n. Let $\mathscr{X}$ be the set of all 1-n paths in K_n, so that $|\mathscr{X}| = K(n)$. Suppose that $\mathbf{X} \sim \mathsf{U}(\mathscr{X})$, and recall that $|\mathbf{X}|$ is the number of edges in the path $\mathbf{X}$. Let the number of edges in K_n be $k(n) = n(n-1)/2$. Let $\mathscr{X}^*$ be the set of 1-n paths in a random graph G of size n and density d (so we treat $\mathscr{X}^*$ as a random variable).

$$
\begin{aligned}
\mathcal{Z}_{n;d} &= K(n)\,\mathbb{P}(\mathbf{X} \in \mathscr{X}^*) \\
&= K(n)\,\mathbb{E}\left[\left(\frac{dk(n)}{k(n)}\right)\left(\frac{dk(n)-1}{k(n)-1}\right)\cdots\left(\frac{dk(n)-|\mathbf{X}|+1}{k(n)-|\mathbf{X}|+1}\right)\right] \\
&= K(n)\,\mathbb{E}\left[\left(\frac{dk(n)-|\mathbf{X}|/2+1/2}{k(n)-|\mathbf{X}|/2+1/2}\right)^{|\mathbf{X}|}\right](1+o(1)) \\
&= K(n)\,\mathbb{E}\left[\left(\frac{dn-1+\frac{n-|\mathbf{X}|}{n-1}}{n-1+\frac{n-|\mathbf{X}|}{n-1}}\right)^{|\mathbf{X}|}\right](1+o(1))\,.
\end{aligned}
$$

The third equality comes from the fact that $|\mathbf{X}| < n$, and the following result for $a < b \ll n$:

$$\begin{aligned}
&(n^2+bn)(n^2+bn-1)\cdots(n^2+an+1)(n^2+an)\\
=&(n^2+\tfrac{b+a}{2}n)^{(b-a)n+1}+\tfrac{1}{24}(b-a)^3n^3(n^2+\tfrac{b+a}{2}n)^{(b-a)n-1}+\cdots\\
=&(n^2+\tfrac{b+a}{2}n)^{(b-a)n+1}(1+o(1))\,.
\end{aligned}$$

We know that $\mathbb{P}(|\mathbf{X}|=y) = \big[(n-2)!/(n-y-1)!\big]/K(n)$, so

$$\begin{aligned}
\mathcal{Z}_{n;d} = K(n)\Bigg[\frac{(n-2)!}{K(n)}\Bigg(\frac{1}{0!}\Bigg(\frac{dn-1+\frac{1}{n-1}}{n-1+\frac{1}{n-1}}\Bigg)^{n-1}+\frac{1}{1!}\Bigg(\frac{dn-1+\frac{2}{n-1}}{n-1+\frac{2}{n-1}}\Bigg)^{n-2}\\
+\frac{1}{2!}\Bigg(\frac{dn-1+\frac{3}{n-1}}{n-1+\frac{3}{n-1}}\Bigg)^{n-3}+\cdots\Bigg)\Bigg](1+o(1))\,.
\end{aligned}$$

For $b \ll n$, it is easy to see that $(an-1+b/(n-1))^{n-b} = (an-1)^{n-b}(1+o(1))$, so

$$\begin{aligned}
\mathcal{Z}_{n;d} &= (n-2)!\Bigg(\frac{1}{0!}\Big(\frac{dn-1}{n-1}\Big)^{n-1}+\frac{1}{1!}\Big(\frac{dn-1}{n-1}\Big)^{n-2}+\cdots\Bigg)(1+o(1))\\
&= (n-2)!\Big(\frac{dn-1}{n-1}\Big)^{n-1}\mathrm{e}^{(\frac{n-1}{dn-1})}(1+o(1))\\
&= (n-2)!\,d^{n-1}\Big(1+\frac{1-1/d}{n-1}\Big)^{n-1}\mathrm{e}^{1/d}(1+o(1))\\
&= (n-2)!\,d^{n-1}\mathrm{e}(1+o(1))\,.
\end{aligned}$$

Using the fact that $K(n) = (n-2)!\,\mathrm{e}(1+o(1))$ from (7), and to force equality to $K(n)$ when $d=1$, we use a model of the form:

$$\mathcal{Z}_{n;d} = K(n)d^{n-1+\delta(n,d)}\,, \tag{15}$$

where $\delta(n,d) \to 0$ as $n \to \infty$. Note that $K(n)$ can be calculated explicitly for any n very quickly.

In Figure 3 we show the data transformed to show the corresponding estimates of $\delta(n,d)$ for each pair (n,d). That is, the Z-axis displays the values of $\widehat{\delta}(n,d) = \log(\widehat{Z}_{n;d}/K(n))/\log(d) - n + 1$. Note that we disregard the data obtained when $d = 0.1$, as the estimates are inaccurate. For $d = 1$, the transformation is not defined, so we cannot use these values either.

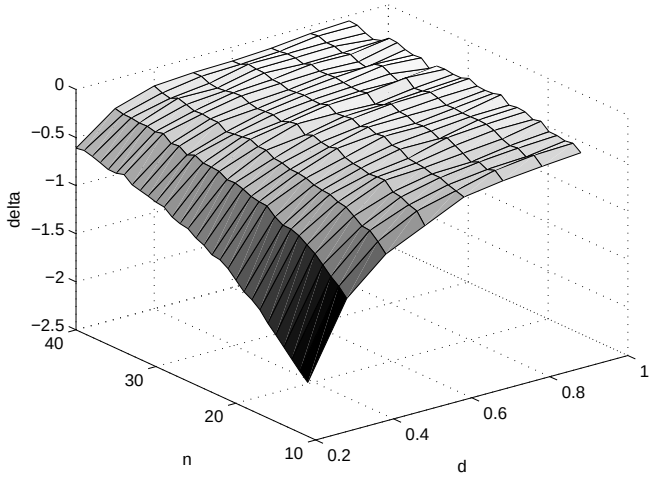


Fig. 3. $\widehat{\delta}(n,d)$ for varying values of d and n.

This supports the model (15) as the estimates of $\delta(n,d)$ are small, and seem to be approaching 0 as n increases. Suppose $\delta(n,d) = \alpha/n+\beta/(dn)+o(n^{-1})$, for some constants α and β. When fitting the data, we find the the following least squares estimates $(\alpha,\beta) = (3.32,-5.16)$. The corresponding proportion of variation of the $\{\widehat{\delta}(n,d)\}$ explained by the model is a very encouraging $R^2 \approx 0.985$, while the explained proportion of variation of the $\{\log(\widehat{\mathcal{Z}}_{n;d})\}$ is a striking $R^2 \approx 0.999996$. In Figure 4 we show how our model for $\delta(n,d)$ behaves over the same range as the data.

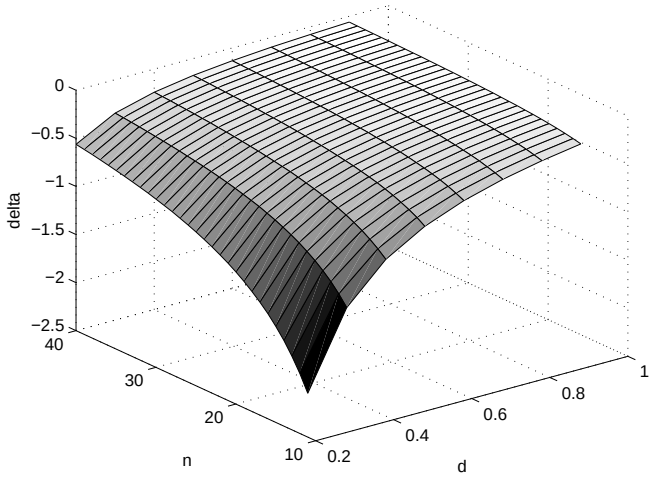


Fig. 4. Our model for $\delta(n,d)$ over the same range as the data.

Due to this extremely good fit, we propose the following approximation $\widetilde{\mathcal{Z}}_{n;d}$ to $\mathcal{Z}_{n;d}$:

$$\widetilde{\mathcal{Z}}_{n;d} = K(n) \cdot d^{n-1+\delta(n,d)}\,, \quad \text{where} \tag{16}$$
$$K(n) = \sum_{k=0}^{n-2} \frac{(n-2)!}{k!}\,, \quad \text{and}$$
$$\delta(n,d) = \frac{3.32}{n} - \frac{5.16}{d\,n}\,.$$

5.6. Testing the Model

We tested the final approximation (16) for random graphs of varying size and density. These results are shown in Table 3, comparing the approximations $\widetilde{\mathcal{Z}}_{n;d}$ with 95% confidence intervals for $\mathcal{Z}_{n;d}$ obtained from simulations of Algorithm 3, using $(N_1', N_2', N_1, N_2) = (50, 100, 1000, 100)$.

Table 3. Comparing the approximation with 95% CIs for $\mathcal{Z}_{n;d}$

n	d	95% CI for $\mathcal{Z}_{n;d}$	$\widetilde{\mathcal{Z}}_{n;d}$
50	0.7	$[8.850, 8.980] \times 10^{53}$	8.924×10^{53}
60	0.3	$[1.146, 1.226] \times 10^{48}$	1.193×10^{48}
70	0.8	$[1.388, 1.404] \times 10^{90}$	1.401×10^{90}
80	0.4	$[1.212, 1.271] \times 10^{84}$	1.255×10^{84}
90	0.9	$[4.261, 4.294] \times 10^{130}$	4.280×10^{130}
100	0.5	$[4.148, 4.301] \times 10^{124}$	4.244×10^{124}

In each case, the estimate from our model (16) lies within the 95% CI of the true value. So we can see that our model estimates $\mathcal{Z}_{n;d}$ extremely well for practically no computational cost, even for graphs of size 100. Recall that the model is based on data from graphs of size 11 to 40. This extrapolation is a good indication that for large n, $\mathcal{Z}_{n;d}$ does indeed increase as our model says it should.

6. Summary

We briefly described the main ideas behind counting via importance sampling and how it applies to the enumeration of simple s-t paths in a graph. We provided an importance sampling algorithm to efficiently estimate the solution to this problem, and showed various numerical results to support the accuracy and speed of this algorithm. Furthermore, the algorithm was applied to the investigation of s-t paths in random graphs, and resulted in a model (16) for the expected number

of s-t paths in a random graph of fixed size and density. Numerical results that justify the accuracy of the model were also shown.

In this paper, we have shown how to estimate the mean and variance of Z_G, the number of paths in a random graph G of fixed size and density. However, for future research, the *distribution* of Z_G could be further investigated. It is intuitive that this distribution would be significantly skewed to the right, implying that it is unlikely to have an approximate normal distribution. If this distribution was more accurately understood, it would be possible to make prediction intervals for Z_G which could be useful.

Acknowledgements. The authors would like to thank professor Reuven Rubinstein for many fruitful discussions on how to solve $\#P$ complete problems via Monte Carlo simulation. This research was supported by the Australian Research Council, via grant No. DP0558957.

References

1. A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, 2001.

2. M. Dyer, A. Frieze, and M. Jerrum. Approximately counting Hamilton paths and cycles in dense graphs. `http://citeseer.ifi.unizh.ch/dyer95approximately.html`, 1995.

3. P. W. Glynn and D. L. Iglehart. Importance sampling for stochastic simulations. *Management Science*, 35:1367–1392, 1989.

4. J. M. Hammersley and K. W. Morton. Poor man's Monte Carlo. *Journal of the Royal Statistical Society*, 16(1):23–38, 1954.

5. H. Khan. Modification of the Monte Carlo method. In *Proceedings, Seminar on Scientific Computation*, pages 20–27. IBM, 1950.

6. H. Khan. Use of different Monte Carlo sampling techniques. In *Symposium on Monte Carlo Methods*, pages 146–190. John Wiley & Sons, 1956.

7. J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, 2001.

8. J. S. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93:1032–1044, 1998.

9. N. Madras and M. Piccioni. Importance sampling for families of distributions. *Ann. Appl. Probab.*, 9:1202–1225, 1999.

10. M. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. of Chemical Physics*, 21:1087–1092, 1953.

11. Motwani R. and R. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

12. C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer, New York, 2nd edition, 2004.

13. M. N. Rosenbluth and A. W. Rosenbluth. Monte Carlo calculations of the average extension of molecular chains. *J. Chem. Phys.*, 23:356–360, 1955.

14. R. Y. Rubinstein. How many needles are in a hay stack, or how to solve fast #p-complete counting problems. *Methodology and Computing in Applied Probability*, 2006. To appear.

15. R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A unified approach to Combinatorial Optimization, Monte Carlo Simulation and Machine Learning*. Springer Verlag, New York, 2004.

16. D. Siegmund. Importance sampling in the Monte Carlo study of sequential tests. *Annals of Statistics*, 4:673–684, 1976.

17. L. G. Valiant. The complexity of enumeration and reliability problems. *Siam Journal of Computing*, 8(3):410–421, 1979.

18. D. J. A. Welsh. *Complexity: Knots, Colouring and Counting*. Cambridge University Press, 1993.

19. D. R. White and M. E. J. Newman. Network vulnerability and cohesion: Fast approximation algorithms for finding node-independent paths in networks and k-components. Informal working paper: Santa Fe Institute, 2001.

A. Adjacency Matrices of Test Graphs

Case 1: ($n = 8$, normal graph, high density.)

$$A = \begin{pmatrix} 0&1&1&1&0&1&1&1 \\ 1&0&0&0&1&1&1&0 \\ 1&0&0&1&1&1&1&1 \\ 1&0&1&0&1&1&1&1 \\ 0&1&1&1&0&1&0&0 \\ 1&1&1&1&1&0&1&1 \\ 1&1&1&1&0&1&0&1 \\ 1&0&1&1&0&1&1&0 \end{pmatrix}.$$

Case 2: ($n = 12$, digraph, high density.)

$$A = \begin{pmatrix} 0&1&1&1&1&1&1&0&1&1&1&1 \\ 1&0&1&1&1&1&1&1&1&1&1&1 \\ 1&1&0&1&1&1&1&1&1&0&1&1 \\ 1&1&1&0&1&1&1&1&1&1&0&1 \\ 1&1&1&0&0&1&1&1&0&1&1&1 \\ 1&1&1&1&1&0&1&1&1&1&1&0 \\ 1&1&1&1&1&1&0&1&1&1&1&1 \\ 1&1&1&1&1&1&1&0&1&1&1&1 \\ 1&1&1&1&1&1&1&1&0&1&1&1 \\ 1&1&1&1&1&1&1&1&1&0&1&0 \\ 1&1&1&1&1&1&1&1&1&1&0&1 \\ 1&1&1&1&1&1&1&1&1&1&1&0 \end{pmatrix}.$$

Case 3: ($n = 16$, digraph, low density.)

$$A = \begin{pmatrix} 0&1&1&0&0&1&0&0&1&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&1&1&0&0&1&0&1&1&1 \\ 1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&1 \\ 1&1&0&0&1&1&1&1&0&1&1&0&1&0&0&1 \\ 0&1&0&0&0&0&1&0&0&1&1&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0 \\ 0&0&0&1&0&1&0&0&1&0&0&0&0&1&0&1 \\ 1&0&0&0&0&0&0&0&1&1&1&0&0&1&0&0 \\ 1&1&0&0&1&0&0&0&0&0&1&0&0&1&0&0 \\ 1&1&0&0&1&1&0&0&1&0&0&0&1&0&1&0 \\ 0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&1 \\ 0&0&1&1&1&1&0&1&0&0&1&0&1&1&0&1 \\ 1&1&0&0&1&1&0&0&0&0&0&0&0&1&0&0 \\ 0&0&1&0&1&1&1&0&1&0&0&1&0&0&1&0 \\ 1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1 \\ 1&1&0&1&0&0&0&1&0&0&1&0&0&0&0&0 \end{pmatrix}.$$

Case 4: ($n = 20$, normal graph, high density.)

$$A = \begin{pmatrix} 0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&0 \\ 1&0&1&1&0&1&1&1&0&1&1&1&1&1&1&1&1&1&1&1 \\ 1&1&0&0&1&0&1&1&1&0&1&1&1&1&1&1&1&0&1&1 \\ 1&1&0&0&1&1&1&1&0&1&1&1&0&1&1&0&1&1&1&1 \\ 1&0&1&1&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&0 \\ 1&1&0&1&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&0&1&0&1&1&1&1&1&1&0&0&1&1&1&1 \\ 1&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1&0&1&1&1 \\ 1&0&1&0&1&1&1&1&0&1&1&1&1&1&1&1&1&0&1&1 \\ 1&1&0&1&1&1&1&1&1&0&1&1&1&0&0&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1&1&1&0&0&1&0&1&1&1&1&0&1 \\ 1&1&1&1&1&1&1&1&1&1&1&0&0&0&1&1&0&1&1&1 \\ 1&1&1&0&1&1&1&1&1&1&1&1&0&0&1&1&1&1&1&0 \\ 1&1&1&1&1&1&0&1&1&0&0&1&1&0&1&1&1&1&1&0 \\ 1&1&1&1&1&1&0&1&1&0&1&1&1&1&1&0&1&1&1&0 \\ 1&1&1&0&1&1&1&0&1&1&1&1&1&1&1&1&0&1&1&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&0&1&1 \\ 0&1&0&1&1&1&1&1&0&1&1&1&1&1&1&1&1&0&1&1 \\ 1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&1&0&1 \\ 0&1&1&1&0&1&1&1&1&1&1&1&0&0&0&0&1&1&1&0 \end{pmatrix}.$$

Case 5: ($n = 24$, normal graph, low density.)

$$A = \begin{pmatrix}
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&1&1&1&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&1&0&0&1&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0\\
1&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
0&1&1&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&1&0&1&0&1&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
1&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&1&1\\
1&0&0&1&1&0&1&1&1&0&1&0&0&0&0&0&0&1&1&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
1&0&0&1&1&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&1&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0
\end{pmatrix}.$$

Case 6: ($n = 30$, digraph, high density.)

$$A = \begin{pmatrix}
0&1&1&1&1&1&1&0&1&1&1&1&0&1&0&1&1&0&0&1&1&0&1&0&1&1&0&1&1&1\\
1&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&0&0&0&1&1&1&1&0\\
1&1&0&0&0&1&1&1&1&1&1&1&1&1&1&0&0&1&1&0&0&1&1&0&1&1&1&1&1&1\\
1&0&1&0&1&1&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&1&1&0&1&1&1&1&1&1&1&0&1&1&0&1&0&1&0&1&0&1&1&1&1\\
1&1&1&1&0&0&1&1&1&0&1&1&1&0&1&0&1&1&1&1&1&0&1&1&1&0&1&0&1&1\\
1&0&1&1&0&1&0&0&1&1&1&0&1&1&1&1&1&1&1&1&0&1&1&0&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&0&1&1&0&1&0&1&1&1&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&1&1&1&1&0&0&0&1&1&1&0&1&1&1&0&1&0&0&1&1&1&1&1&1&0&1&1&0\\
1&1&1&0&1&1&0&1&1&0&1&0&1&1&1&0&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
1&0&1&1&1&1&1&1&1&1&1&0&1&0&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&0&1&0&1&1&1&1&0&1&0&1&1&1&0&1&1&1&1&1&0&1&0\\
0&1&0&1&1&1&1&1&1&1&0&1&0&1&0&1&1&1&0&1&1&0&1&1&1&1&1&1&0&1\\
1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&0&1&0&0&1&1&1&1&0&1&1&0&1&1&1\\
0&1&1&1&1&1&0&1&0&0&1&1&0&1&1&1&0&1&1&0&0&1&1&0&0&1&1&1&1&1\\
0&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&0&1&0&1&1&1&1&1&1&1&1&0&0&1\\
1&1&1&1&1&1&1&1&1&1&0&1&1&1&0&1&1&1&0&1&1&1&1&1&1&1&1&0&1&1\\
0&1&1&1&0&1&0&1&1&1&1&1&1&1&0&1&1&0&1&0&1&1&1&1&1&1&0&0&1&1\\
1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1&0&1&0&0&1&1&1&1&0&1\\
1&1&0&1&1&1&1&0&0&1&1&1&1&1&1&1&1&1&1&0&1&0&0&1&1&1&1&1&1&0\\
0&1&1&1&0&1&1&1&1&1&0&1&1&1&1&1&0&1&0&1&1&1&0&1&1&1&0&0&1&1\\
1&1&0&1&1&1&1&1&0&1&1&1&1&1&1&0&1&0&1&1&1&1&1&1&0&1&1&0&0&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&0&1&0\\
1&1&0&1&1&1&1&1&1&0&1&1&1&0&1&1&0&1&1&1&0&1&0&0&1&0&1&1&0&1\\
1&1&1&1&1&1&1&0&1&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&0&1&1&1&1&1&1&1&0&1&0&1&0&1&1&1&1&0&1&1&1&1&1&1&0&0&1&1&\\
1&0&0&1&1&1&1&1&0&1&0&1&1&1&1&0&0&1&1&0&1&1&1&1&1&1&0&1&0&1\\
1&1&1&1&1&1&0&1&1&1&1&1&1&0&1&1&0&1&1&1&1&1&1&0&1&0&1&1&1&0
\end{pmatrix}.$$

B. Estimates for various quantities from the simulation of Algorithm 3.

Table 4. Relative errors for the estimates of $\mathcal{Z}_{n;d}$ from Algorithm 3 using (5).

	d								
n	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
11	.013	.0095	.0077	.0060	.0052	.0041	.0034	.0026	.0018
12	.013	.011	.0082	.0066	.0050	.0043	.0034	.0026	.0018
13	.013	.011	.0083	.0066	.0053	.0044	.0036	.0027	.0018
14	.013	.011	.0088	.0070	.0055	.0046	.0036	.0028	.0018
15	.016	.012	.0094	.0071	.0056	.0046	.0037	.0028	.0019
16	.016	.013	.0091	.0072	.0058	.0047	.0038	.0029	.0019
17	.018	.014	.0096	.0075	.0058	.0048	.0038	.0028	.0020
18	.019	.015	.010	.0077	.0061	.0049	.0039	.0030	.0019
19	.019	.014	.010	.0080	.0062	.0050	.0040	.0030	.0020
20	.023	.015	.010	.0078	.0064	.0051	.0040	.0030	.0020
21	.023	.015	.011	.0082	.0063	.0052	.0041	.0030	.0020
22	.025	.016	.011	.0083	.0067	.0052	.0041	.0031	.0020
23	.028	.017	.011	.0086	.0067	.0053	.0042	.0031	.0021
24	.025	.017	.011	.0085	.0066	.0054	.0042	.0032	.0021
25	.030	.017	.012	.0088	.0072	.0054	.0043	.0033	.0021
26	.032	.016	.012	.0092	.0071	.0055	.0043	.0032	.0021
27	.037	.017	.013	.0091	.0074	.0056	.0044	.0033	.0022
28	.036	.018	.012	.0094	.0075	.0057	.0044	.0033	.0022
29	.042	.018	.012	.0096	.0074	.0059	.0045	.0033	.0022
30	.055	.020	.012	.0098	.0076	.0058	.0045	.0033	.0021
31	.041	.019	.013	.0097	.0075	.0060	.0045	.0033	.0022
32	.030	.020	.014	.010	.0077	.0061	.0046	.0035	.0022
33	.047	.021	.015	.011	.0077	.0061	.0046	.0034	.0022
34	.057	.020	.014	.011	.0080	.0060	.0046	.0034	.0022
35	.044	.023	.015	.010	.0079	.0061	.0048	.0035	.0022
36	.051	.022	.015	.010	.0081	.0061	.0048	.0035	.0023
37	.047	.022	.016	.011	.0085	.0062	.0048	.0035	.0023
38	.076	.021	.014	.011	.0084	.0066	.0048	.0036	.0023
39	.061	.024	.016	.011	.0084	.0065	.0049	.0035	.0023
40	.068	.024	.016	.012	.0083	.0065	.0049	.0036	.0023

Note that these are the relative errors of the $\{\widehat{\mathcal{Z}}_{n;d}\}$, not the $\{\log(\widehat{\mathcal{Z}}_{n;d})\}$. However, they are small enough so that they are still very good estimates of the relative errors of the $\{\log(\widehat{\mathcal{Z}}_{n;d})\}$.

Table 5. Estimates for the relative standard deviations of the $\{Z_G\}$ from Algorithm 3 using (14).

					d				
n	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
11	2.0	0.83	0.57	0.36	0.39	0.30	0.27	0.20	0.16
12	1.9	0.79	0.50	0.42	0.35	0.32	0.28	0.21	0.15
13	1.8	0.72	0.49	0.40	0.37	0.30	0.26	0.20	0.14
14	1.6	0.75	0.52	0.42	0.36	0.34	0.26	0.19	0.13
15	1.5	0.64	0.47	0.40	0.37	0.31	0.24	0.18	0.14
16	1.8	0.58	0.47	0.40	0.38	0.30	0.25	0.18	0.12
17	1.4	0.62	0.49	0.41	0.36	0.31	0.25	0.18	0.13
18	1.5	0.66	0.50	0.43	0.38	0.31	0.24	0.18	0.12
19	1.3	0.56	0.50	0.44	0.41	0.30	0.25	0.18	0.12
20	1.2	0.54	0.46	0.41	0.36	0.28	0.22	0.16	0.10
21	1.1	0.53	0.50	0.43	0.36	0.28	0.20	0.16	0.11
22	1.2	0.53	0.59	0.47	0.36	0.29	0.23	0.16	0.11
23	1.1	0.59	0.52	0.45	0.39	0.29	0.21	0.17	0.11
24	1.1	0.62	0.57	0.45	0.38	0.29	0.21	0.16	0.10
25	1.1	0.58	0.53	0.46	0.36	0.27	0.21	0.14	0.097
26	1.0	0.64	0.55	0.45	0.36	0.28	0.22	0.15	0.10
27	0.99	0.61	0.49	0.39	0.33	0.27	0.21	0.16	0.099
28	0.97	0.63	0.51	0.44	0.33	0.28	0.21	0.15	0.093
29	0.80	0.63	0.53	0.44	0.33	0.24	0.20	0.15	0.10
30	0.89	0.69	0.56	0.41	0.31	0.27	0.20	0.14	0.10
31	0.85	0.60	0.66	0.41	0.34	0.24	0.19	0.15	0.089
32	1.1	0.78	0.51	0.45	0.32	0.24	0.20	0.14	0.093
33	0.94	0.62	0.55	0.45	0.32	0.24	0.19	0.14	0.085
34	0.87	0.74	0.57	0.43	0.33	0.25	0.18	0.14	0.086
35	0.86	0.64	0.57	0.42	0.32	0.24	0.18	0.13	0.087
36	0.68	0.62	0.55	0.40	0.32	0.22	0.18	0.14	0.084
37	0.87	0.73	0.56	0.42	0.30	0.25	0.17	0.13	0.080
38	0.77	0.77	0.51	0.39	0.31	0.21	0.17	0.12	0.086
39	0.75	0.68	0.49	0.38	0.30	0.24	0.18	0.12	0.080
40	0.73	0.71	0.53	0.39	0.29	0.24	0.18	0.12	0.080