

# Benford’s Law

**Ian Gent**

The School of Computer Science  
University of St Andrews  
St Andrews  
Scotland  
ipg@dcs.st-and.ac.uk

**Toby Walsh**

Department of Computer Science  
University of York  
York  
England  
tw@cs.york.ac.uk

## Abstract

Benford’s Law predicts the frequency of the leading digit in numbers met in a wide range of naturally occurring phenomena. In data following Benford’s Law, numbers start with a small leading digit more often than those with a large leading digit. Here we demonstrate that Benford’s Law also describes a wide range of computational phenomena. In particular, we show that a number of different statistics associated with computation like space and runtime often follow Benford’s Law. We also show that search cost on input data that follows Benford’s Law is often very different to that on more uniform data. These results could be used to improve algorithm performance (for example, for load balancing or disk de-fragmentation), as well as to help model algorithm performance. Benford’s Law can also be used to generate data for benchmarking algorithms that may be more realistic than purely random data.

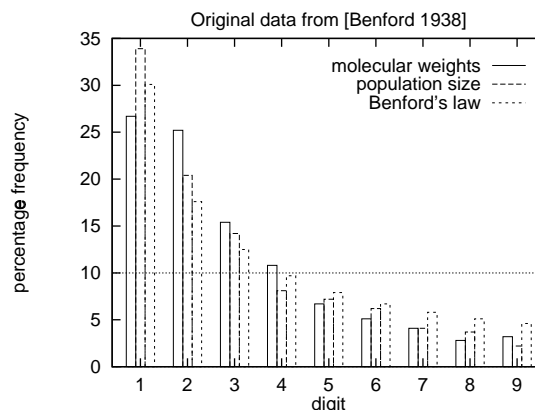
## 1 Introduction

Frank Benford [Benford, 1938] studied a large amount of naturally occurring data (e.g. tables of molecular weights, population sizes, river basin drainage areas, and numbers appearing on newspaper front pages) and showed that the leading digits tend not to be uniformly distributed (see Figure 1).

For instance, numbers tend to start with the digit 1 in over 30% of his data, but with the digit 9 in less than 5% of the data. To model such data, he proposed an (empirical) law which predicts that the leading digit,  $d$  occurs with a frequency given by:

$$\text{prob}(d = i) = \log_{10}\left(1 + \frac{1}{i}\right) \quad (1)$$

We show here that Benford’s Law models not just natural phenomena like population sizes but also many computational phenomena. In particular, many statistics associated with algorithms like runtime often obey this law. These results could be used to improve how we model such statistics. In addition, by using Benford’s Law, we can generate benchmark sets that may be more realistic than purely uniform random data. Finally, we show that data obeying Benford’s Law is often easier to reason about than random data that is more uniformly generated.

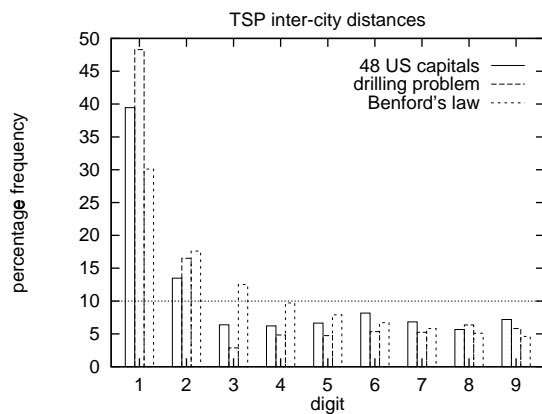


**Figure 1.** Distribution of leading digits in data summarized in Benford’s original 1938 paper [Benford, 1938]. “Molecular weights” is a sample of 1,800 different molecular weights. “Population size” is a sample of 3,259 different population sizes.

## 2 Input data

We begin with some input data taken from TSPLib. In Figure 2, we plot the distribution of leading digits in the inter-city distance matrix. We looked at another dozen data sets to those used in Figure 2 and observed similar results. As distances are computed using the 2-dimensional Euclidean distance function, we expected to see significant correlation between the distances. We were therefore surprised to see data following Benford’s Law. We conjecture this is a result of the clustering of cities which provide the data with several different length scales (i.e. distances within the clusters, and distances between the clusters). Indeed, the data displays considerable scale invariance. The choice of units was therefore not very important. If the inter-city distances are converted from imperial to metric units, we observe a similar distribution in the leading digits.

We next looked at some time-series data (see Figure 3). Again the data appears to follow Benford’s Law. This is perhaps less surprising than the TSP data. For a time-series to go from 1 to 2, the statistic must double. But to go from 8 to 9, we need a much more modest increase. A time-varying statistic will therefore often spend more time with a small leading



**Figure 2.** Distribution of leading digits in inter-city TSP data. “48 US capitals” is problem at t48 and contains 1,128 different inter-city distances. “drilling problem” is problem a280 and contains 39,060 different inter-city distances. In both cases, the problems are symmetric and distances are computed using the 2-dimensional Euclidean distance function.

digit than a large one. As the data again displays considerable scale invariance, the choice of units does again not appear to be very important.

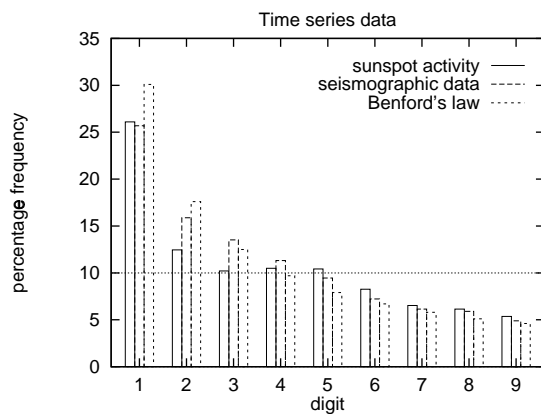
What consequences do results like these have? First, a wide variety of input data can follow Benford’s Law. Features of the real world (like the clustering of populations) may be responsible for these appearances of Benford’s Law. Second, we should be careful of testing algorithms on uniform and random data. It may therefore be more realistic to generate data that follows Benford’s Law. In later sections, we explore the impact this can have on algorithm performance. Before that we identify some more computational examples of Benford’s Law that help illustrate its ubiquity.

### 3 Space and runtime

We next looked at file size. In Figure 4, we plot the leading digit frequency for file sizes on an Unix computer. We measured filesizes using the “ls” command which returns the number of bytes contained in a file. We obtained similar results if we measure the number of blocks occupied.

The distribution of leading digits, for one user or all users, appears to follow Benford’s Law closely. File sizes range over several orders of magnitude. As with previous data sets, file sizes display considerable scale invariance, so we obtain a similar distribution of leading digits when file size are measured in 512 or 1024 byte blocks. We have also observed similar leading digit frequencies for file sizes on other computers and operating systems.

We then looked at runtimes. In Figure 5, we plot the leading digit frequency for the cost to solve a hard optimization and a hard decision problem. We looked at the traveling salesperson (TSP) problem, and the satisfiability of propositional formulae (SAT). Given a list of cities, and a matrix giving the distance between each pair of cities, the TSP problem is to find the shortest tour that visits each city. To find optimal



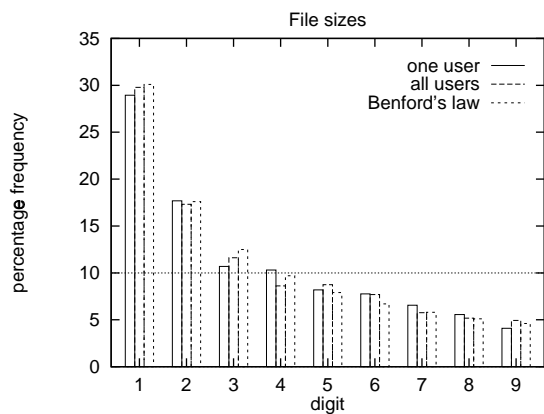
**Figure 3.** Distribution of leading digits in some time-series data. “sunspot activity” is problem andrews14.dat, the 2,820 monthly sunspot numbers observed in Zurich from 1749 to 1983. “seismographic data” is problem kobe.dat, the vertical acceleration in nm/sq.sec due to the Kobe earthquake, recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing for 51 minutes at 1 second intervals.

TSP tours, we use a branch and bound algorithm with the Hungarian heuristic for branching [Lawler *et al.*, 1985]. The search tree explored by this algorithm on some randomly generated Euclidean 2-D problems has a mean of 30,859 nodes, a median of 306 nodes, and a maximum of 17,838,398 nodes. Given a propositional formula, the SAT problem is to determine if there is an assignment of true and false to the variables that makes the whole formula true. To answer the SAT decision problem, we use the Davis-Putnam algorithm [Davis *et al.*, 1962]. The mean time by this algorithm to determine if a formula is satisfiable on some randomly generated “phase transition” problems is 19,117 milliseconds, the median is 3,460 ms, and the maximum is 31,559 ms.

Runtimes in both experiments range over several orders of magnitude, and the data appears to follow Benford’s Law closely. Similar results are obtained with a variety of different measures of search cost (e.g. CPU time, nodes or branches visited in the search tree) again suggesting a certain amount of scale invariance. Results like these might be used to improve system performance, especially for online algorithms. For example, software for allocating disk space could be optimized to minimize disk fragmentation with file sizes obeying Benford’s Law. As a second example, software for scheduling jobs could be optimized to runtimes for individual jobs that obey Benford’s Law.

### 4 Generating Benford’s Law data

Given the ubiquity of data obeying Benford’s Law, we may wish to benchmark algorithms with random data that obeys Benford’s Law. To generate such data, we merely need to scatter points uniformly and at random on a log scale. The magnitude of such numbers then fits Benford’s Law as the following theorem shows.



**Figure 4.** Distribution of leading digits in the size of files on a Sun Ultra 10 running the Solaris operating system. The machine address is `cally.dai.ed.ac.uk`. “One user” are the 10,215 files owned by a single user. The mean file size is 14,728 bytes, the median is 1,839 bytes, and the maximum is 10,950,842 bytes. “All users” are the 83,698 files owned by all users. The mean file size is now 30,787 bytes, the median is 2,704 bytes, and the maximum is 102,756,352 bytes.

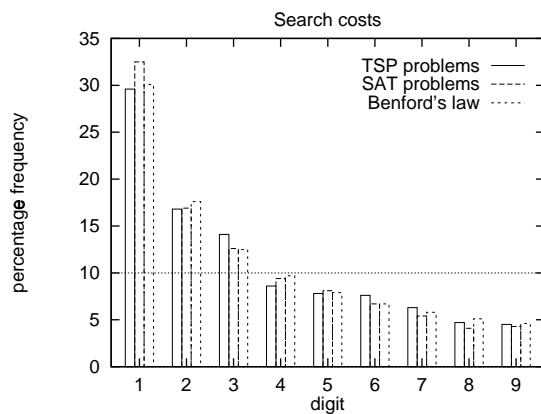
**Theorem 1** *The random variable  $10^{\text{rand}(k)}$  obeys Benford’s Law under the assumptions that  $k$  is positive and integral, and that the function  $\text{rand}(k)$  returns a random number in the range  $[0, k)$ .*

**Proof:** By induction on  $k$ . Consider the case  $k = 1$ . The probability that the leading digit is  $i$  equals the probability that  $10^{\text{rand}(1)}$  lies in the interval  $[i, i + 1)$ . This equals the probability that  $\text{rand}(1)$  lies in the interval  $[\log_{10}(i), \log_{10}(i + 1))$ . This is simply  $\log_{10}(i + 1) - \log_{10}(i)$ , which equals  $\log_{10}(1 + 1/i)$  as predicted by Benford’s Law.

In the step case, consider  $k = n + 1$ . The proof divides into two cases depending on whether  $\text{rand}(k)$  lies in  $[0, n)$  or  $[n, n + 1)$ . In each case, by the induction hypothesis, we get a distribution of leading digits which obeys Benford’s Law. As a weighted sum of two statistics obeying Benford’s Law also obeys Benford’s Law, the combined distribution obeys Benford’s Law.  $\square$

This result demonstrates how to generate numbers within the interval  $[1, 10^k)$  that obey Benford’s Law. We can easily generalize the result to generate numbers obeying Benford’s Law in the interval  $[10^j, 10^k)$  for  $j < k$  and both integral. If  $k \gg j$ , we can drop the restriction that  $j$  and  $k$  are integral since the error caused by the first and last log interval being incomplete will be small, and the distribution will approach Benford’s Law.

Data obeying Benford’s Law could be generated by other methods. For example, we could fix the size of numbers and generate the first digit of each at random using the probability given by Equation (1), and the remaining digits with uniform probability. However, such methods will tend to sample numbers less uniformly than scattering points at random on a log scale. Another desirable property of scattering points at random on a log scale is that it gives scale invariant data, and our experiments have shown that such data is common.



**Figure 5.** Distribution of leading digits in the cost to solve some hard computational problems. “TSP problems” are 1,000 randomly generated problems with between 4 and 24 cities, placed at random on a square of side 1000. “SAT problems” are 1,000 randomly generated formulae with between 10 and 100 variables. Problems are at the phase transition in satisfiability [Mitchell *et al.*, 1992] with a ratio of clauses to variables of 4.3. A similar distribution of the leading digits of runtimes is observed if we fix the problem size at 100 variables and varied the ratio of clauses to variables.

## 5 Benchmarking

Algorithms are often benchmarked with uniformly generated random problems, especially in those domains in which “phase transition” behavior has been identified (e.g. [Mitchell *et al.*, 1992; Korf, 1995; Pemberton and Zhang, 1996]). We therefore compared algorithm performance on input data obeying Benford’s Law with that on uniform random data. We considered two problem defined solely by a sequence of numbers: TSP and number partitioning problems.

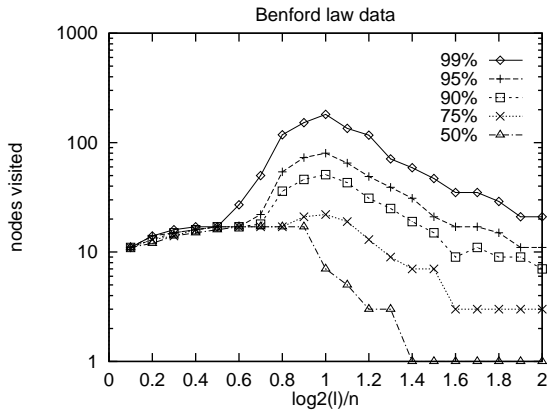
### 5.1 Number partitioning

Given a bag of integers, we wish to find the partition into two bags which minimizes the difference between the sum of the integers in each bag. Table 1 demonstrates a large difference in the cost to partition a bag of numbers obeying Benford’s Law compared to a bag containing purely random numbers. To find the optimal partition, we use Korf’s efficient CKK algorithm [Korf, 1995]. Problems contain  $n$  numbers drawn at random from the interval  $(0, l]$  either uniformly or so as to obey Benford’s Law. We set  $l = 2^n$  as this is close to the phase boundary where hard problems are found.

As with uniform data [Gent and Walsh, 1998], random data generated to obey Benford’s Law displays a phase transition in the probability of a perfect partition that sharpens around  $\log_2(l)/n \approx 1$  as  $n$  increases. For over-constrained problems (i.e. for  $\log_2(l)/n > 1$ ), the size of the optimal partition increases roughly exponentially with  $n$  (for fixed  $\log_2(l)/n$ ) and with  $\log_2(l)/n$  (for fixed  $n$ ). This is similar to random data generated uniformly. Search cost is, however, much less than that for uniform random data. This highlights why we may wish to benchmark algorithms using random data generated to obey Benford’s Law.

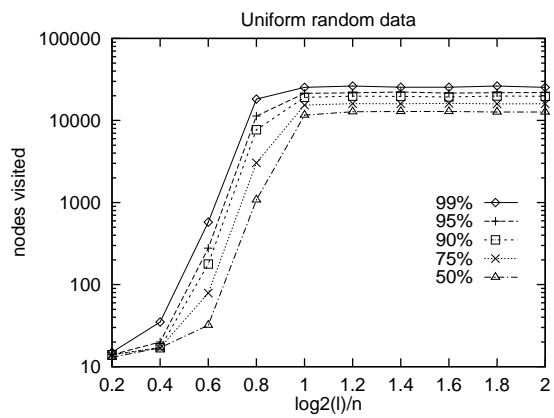
$n$	nodes visited	mean	median	max	min
10	Benford Law	2.83	1	21	1
	uniform data	20.00	21	46	1
20	Benford Law	8.63	2	221	1
	uniform data	5711.38	5725	16312	1
30	Benford Law	19.96	3	1498	1
	uniform data	2519894.5	2581221	5714518	72122
40	Benford Law	36.30	3	1357	1
	uniform data	-	-	-	-
50	Benford Law	94.11	4	4494	1
	uniform data	-	-	-	-
60	Benford Law	280.24	5	51880	1
	uniform data	-	-	-	-

**Table 1.** Search cost to partition 1000 randomly generated problems. Each problem contains  $n$  integers drawn at random from the interval  $(0, 2^n]$  either uniformly or so as to obey Benford’s Law.

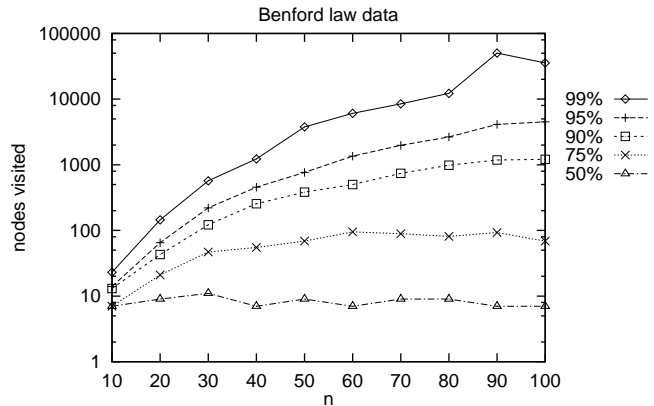


**Figure 6.** Percentiles in the search cost for the CKK algorithm to partition  $n = 20$  numbers drawn from the interval  $(0, l]$  using a distribution that follows Benford’s Law. 1000 problems were generated at each integer value of  $\log_2(l)$ .

In Figures 6 and 7, we plot phase transition curves for the search cost of partitioning numbers drawn either from an uniform distribution or from a distribution that obeys Benford’s Law. Although both uniform and Benford’s Law data have a complexity peak that is associated with the transition from perfect to imperfect partitions, there are significant differences between the two graphs. In particular, numbers drawn from a distribution that follows Benford’s Law are much easier to partition than numbers drawn from an uniform distribution, especially for  $\log_2(l)/n > 1$ . In Figure 8, we plot how the search costs increases as  $n$  increases and  $\log_2(l)/n$  remains fixed at 1. The percentage of problems with a perfect partition drops from 31% at  $n = 10$  to just 5.5% at  $n = 100$ . We suspect this may be the major cause of the apparent less than exponential growth in the higher percentiles. We conjecture that, if  $\log_2(l)/n$  is decreased as  $n$  is increased so that the percentage of perfect partitions remains constant at approximately 50%, then simple exponential growth might be observed. Problems are, however, much easier to solve than uniform data for which search cost grows as approximately  $2^{0.85n}$  [Gent and Walsh, 1998].



**Figure 7.** Percentiles in the search cost for the CKK algorithm to partition  $n = 20$  numbers drawn from the interval  $(0, l]$  using a uniform distribution. 1000 problems were generated at each integer value of  $\log_2(l)$ .



**Figure 8.** Percentiles in the search cost for the CKK algorithm to partition  $n$  numbers drawn from the interval  $(0, 2^n]$  using a distribution that follows Benford’s Law. 1000 problems were generated at each value of  $n$ .

## 5.2 Traveling salesperson (TSP) problems

To determine if these results were specific to number partitioning, we turned to the traveling salesperson (TSP) problem. To find optimal tours, we again use a branch and bound algorithm with the Hungarian heuristic for branching [Lawler *et al.*, 1985]. Table 2 demonstrates a large difference in the cost to find the optimal tour for input data obeying Benford’s Law compared to uniform random data. The Benford Law data was generated by computing each entry in an  $n$  by  $n$  symmetric inter-city distance matrix using the random variable,  $10^{rand(4)}$ . The uniform random data was generated by placing  $n$  cities at random on a square of side  $10^4$  and computing the integer Euclidean 2-D distance between them.

Why are TSP problems with inter-city distances obeying Benford’s Law easier to solve to optimality than uniform random problems? TSP problems generated to obey Benford’s Law will contain inter-city distances at all length scales. As a result, many of the legs leaving a particular city will be ruled

$n$	nodes visited	mean	median	max	min
10	Benford Law	32.55	29	116	31
	uniform data	112.15	46	1702	82
15	Benford Law	132.81	108	662	65
	uniform data	2617.79	322	75493	258
20	Benford Law	398.83	313	2049	650
	uniform data	13289.06	2859	373442	554
25	Benford Law	1289.37	912	9562	162
	uniform data	207055.97	23548	5528963	1369
30	Benford Law	3081.51	1477	3122	869
	uniform data	1087824.11	210054	32739476	218682
35	Benford Law	8376.13	2634	137136	1869
	uniform data	-	-	-	-

**Table 2.** Search cost to find optimal tour for 100 problems with  $n$  cities and an inter-city distance matrix generated either to obey Benford’s Law or from a random and uniform placement of the cities on a 2-D square.

out by an algorithm like branch and bound as they are obviously too long. Hence, the “effective” branching rate is lower than with uniform random problems where the inter-city distances follow many fewer length scales. The fact that random data obeying Benford’s Law proved easier than uniform random data in both these domains should be reassuring. By exploiting structure like that captured by Benford’s Law, we may be able to cope with problems that are intractable in the worst case.

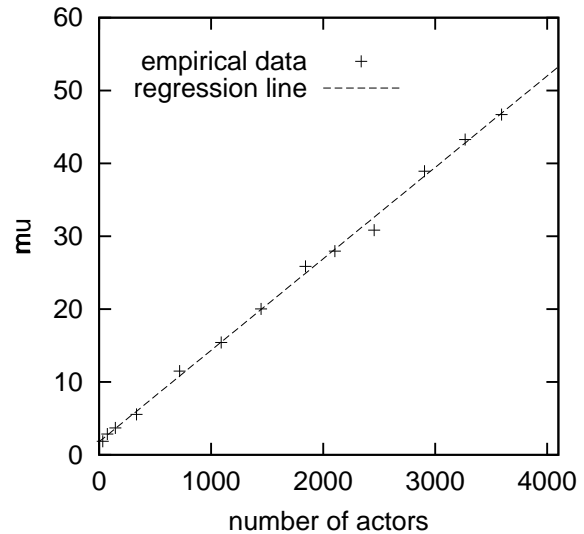
## 6 Small worlds

The clustering of leading digits modelled by Benford’s Law may be related to clustering in other structures like graphs. For example, Watts and Strogatz have shown that graphs that occur in many biological, social and man-made systems are often neither completely regular nor completely random but have instead a “small world” topology in which nodes are highly clustered and path lengths are small [Watts and Strogatz, 1998]. Walsh showed that graphs with a small world topology occur frequently in search problems and can have a large impact on search cost [Walsh, 1999].

To identify graphs with a small world topology, Walsh introduced the scale independent parameter  $\mu$  [Walsh, 1999]. This is defined as the clustering coefficient (the average fraction of your neighbours connected to each other) divided by the characteristic path length (the average path length between any two nodes), both normalized by their values for a random graph with the same number of nodes and edges. By definition, random graphs have  $\mu = 1$ , while graphs with a small world topology have  $\mu \gg 1$ . To demonstrate the ubiquity of graphs with a small world topology, Watts & Strogatz [Watts and Strogatz, 1998] used graphs derived from three sources: the online film actors database, the power grid in the western half of the United States, and the neural network of the nematode worm. In the rest of this section, we focus on the actors database. We conjecture that similar arguments could be made for other graphs with a small world topology.

To generate a graph from the actors database, we associate actors with nodes and add an edge between nodes whenever the two actors associated with these nodes appear together in a film. We wish to simulate this process but control the frequency with which actors appear so, for example, we can fit

it to a Benford Law distribution. We begin by fixing the total number of times each actor appears in all films. We then write out a long list of their names, each name appearing the appropriate number of times. We randomise this list, and break it into lengths determined by the cast size of each movie. Using these cast lists, we then construct the associated graph. This model ignores certain features of casting films. For example, there is no provision for pairs of actors, such as Laurel and Hardy, who appeared in many films together. We do, however, allow the same actor to appear more than once in a film, as Alec Guinness did in ‘Kind Hearts and Coronets’.



**Figure 9.** Mu parameter  $\mu$  (i.e. small worldiness) of randomised film data graphs. All films had a cast length of 5. The number of appearances of each actor ranges from 1 to 25 and fits Benford’s Law. The total number of cast appearances ranged from 250 to 25,000, (i.e. we simulated from 50 to 5000 films). A linear regression fit is also plotted.

Some results are shown in Figure 9. We observe linear growth in the small-worldiness as the number of actors is increased. We see similar linear growth with other generation parameters. The real actors database used by [Watts and Strogatz, 1998] has  $n \approx 250,000$  and  $\mu = 2396$ . Extrapolation of Figure 9 suggests that at  $n = 250,000$ ,  $\mu \approx 3,100$ . The closeness of these numbers is probably not significant, but it is significant that this simple random process based on a distribution obeying Benford’s Law gives graphs of similar small worldiness to the real graph.

This method of generating graphs creates a small world topology comparable to that generated by the rewiring process proposed in [Watts and Strogatz, 1998] or the “morphing” process proposed in [Gent *et al.*, 1999]. As many real world graphs have a significant small world topology, Benford’s Law may also be useful for benchmarking algorithms involving graphs. Applications of Benford’s Law are thus not restricted to purely numerical domains.

## 7 Related work

Benford's Law can be traced back to Newcomb [Newcomb, 1881] who observed that tables of logarithms were dirtier at the front than the back, and proposed without explanation the equation  $\log_{10}(1 + 1/i)$  for predicting the frequency of the leading digit. The law was then largely forgotten until [Benford, 1938]. From a computational perspective, Knuth has suggested [Knuth, 1981] that numerical computations be optimized to deal with data following Benford's Law. Recently, Nigrini has shown that many aspects of financial accounts like expenses claims follow Benford's Law [Nigrini, 1996]. Using standard statistical tests, he is able to detect fraudulent or erroneous data which deviates from the law.

Hill's theorem [Hill, 1995] offers an explanation for the ubiquity of Benford's Law in natural phenomena. This theorem proves that a random mix of different distributions follows Benford's Law even when the individual distributions themselves do not follow or approximate Benford's Law. Our results show that many statistics associated with algorithms like runtime often obey Benford's Law. This suggests that the performance of algorithms may be best modelled as the synthesis of several different distributions and not by any one distribution. This concurs with the arguments of [Hoos and Stutzle, 1998] for studying run-time distributions for Las Vegas style algorithms on *single* problem instances in an attempt to separate out different distributions.

## 8 Conclusions

We have shown that Benford's Law describes a wide range of computational phenomena. We first identified Benford's Law in benchmark TSP and time-series data. We then observed Benford's Law in a number of other computational statistics including file size and runtime. We proved that scattering numbers uniformly on a log scale gives data that both obeys Benford's Law and is scale invariant. We then studied the impact of data obeying Benford's Law on algorithms for the TSP and number partitioning problem. We observed phase transition behaviour similar to that seen with uniform data. However, the cost of optimization with data obeying Benford's Law was significantly smaller than the cost with uniform random data. Finally, we identified a connection between the clustering of digits in Benford's Law and clustering in other structures like graphs with a small world topology. This offers a new way of generating graphs with a small world topology.

What general lessons can be learnt from this study? First, many different computational phenomena appear to follow Benford's Law. This observation could be used to improve algorithm performance (e.g. by load balancing), as well as to model algorithm behaviour. Second, generating data to obey Benford's Law may be useful for benchmarking algorithms. For example, partitioning integers that obey Benford's Law is much easier than partitioning integers that are distributed uniformly. And finally, many other complex systems may follow Benford's Law. For example, the size of files transferred over the Internet and their transmission times may follow Benford's Law. The previously reported Zipf-like distributions found in such data [Huberman *et al.*, 1998] suggest a

certain amount of scale invariance, and scale invariance leads to Benford's Law [Hill, 1995]. We should therefore be on the lookout for Benford's Law in other computational domains.

## Acknowledgements

The second author is supported by an EPSRC Advanced Research Fellowship. Thanks to the other members of the APES Group for their comments and feedback (see <http://apes.cs.strath.ac.uk>).

## References

- [Benford, 1938] F. Benford. The law of anomalous numbers. *Proc. of the American Philosophical Society*, 78:551–572, 1938.
- [Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *CACM*, 5:394–397, 1962.
- [Gent *et al.*, 1999] I.P. Gent, H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proceedings of the 16th National Conference on AI*. 1999.
- [Gent and Walsh, 1998] I.P. Gent and T. Walsh. Analysis of heuristics for number partitioning. *Computational Intelligence*, 14(3):430–451, 1998.
- [Hill, 1995] T.P. Hill. Base-invariance implies Benford's law. *Proc. of the American Math. Soc.*, 12:887–895, 1995.
- [Hoos and Stutzle, 1998] H. Hoos and T. Stutzle. Evaluating las vegas algorithms - pitfalls and remedies. In *Proc. of 14th Annual Conf. on Uncertainty in AI (UAI-98)*, 1998.
- [Huberman *et al.*, 1998] B.A. Huberman, P.L.T. Pirolli, J.E. Pitkow, and R.M. Lukose. Strong regularities in world wide web surfing. *Science*, 280:95–97, 1998.
- [Knuth, 1981] D. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms (3rd edition)*. Addison-Wesley, Reading, MA., 1981.
- [Korf, 1995] R. Korf. From approximate to optimal solutions: A case study of number partitioning. In *Proc. of the 14th IJCAI*. 1995.
- [Lawler *et al.*, 1985] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons, Chichester, 1985.
- [Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. of the 10th National Conference on AI*, pages 459–465. 1992.
- [Newcomb, 1881] S. Newcomb. Note on the frequency of the use of digits in natural number. *American Journal of Mathematics*, 4:39–40, 1881.
- [Nigrini, 1996] N. Nigrini. A taxpayer compliance application of benford's law. *Journal of the American Taxation Association*, 18:72–91, 1996.
- [Pemberton and Zhang, 1996] J.C. Pemberton and W. Zhang. Epsilon-transformation: exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81:297–325, 1996.

- [Walsh, 1999] T. Walsh. Search in a small world.  
In *Proc. of 16th IJCAI*. 1999. available from  
<http://apes.cs.strath.ac.uk/reports/apes-07-1998.ps.gz>.
- [Watts and Strogatz, 1998] D.J. Watts and S.H. Strogatz.  
Collective dynamics of 'small-world' networks. *Nature*,  
393:440–442, 1998.