On Generation of Probable Primes by Incremental Search

Jørgen Brandt and Ivan Damgård
Aarhus University

September 16, 1992

Abstract

This paper examines the following algorithm for generating a probable prime number: choose a random k bit odd number n, and test the numbers $n, n+2, \ldots$ for primality using t iterations of Rabin's test, until a probable prime has been found or some maximum number s of candidates have been tested.

We show an explicit upper bound as a function of k, t and s on the probability that this algorithm outputs a composite. From Hardy and Littlewoods prime r-tuple conjecture, an upper bound follows on the probability that the algorithm fails. We propose the entropy of the output distributrion as a natural measure of the quality of the output. Under the prime r-tuple conjecture, we show a lower bound on the entropy of the output distribution over the primes. This bound shows that as $k \to \infty$ the entropy becomes almost equal to the largest possible value.

Variants allowing repeated choice of starting points or arbitrary search length are also examined. They are guaranteed not to fail, and their error probability and output entropy can be analysed to some extent.

1 Introduction

Apart from being mathematically interesting, it is well-known that efficient generation of prime numbers is of extreme importance in modern cryptography.

Although prime numbers can be generated together with a proof of primality quite efficiently [8], using the Rabin test [10] remains in many cases the most practical method. This is true, even though this test allows some probability of error: it always accepts a prime number, but will sometimes accept a composite. The maximal probability with which this happens for a given composite is $1/4$, but in general the probability is much smaller.

Let $x = 2^k$, and let M_k be the set of odd numbers in the interval $[x/2..x[$, i.e. the set of odd numbers of bit length precisely k.

An obvious method for generating a probable prime number is to choose uniformly an element n from M_k and subject it to (at most) t independent iterations of the Rabin test. This is repeated until an n is found that passes t iterations. The error probability of this algorithm was studied in [4].

An often recommended alternative to this method uses incremental search from a randomly chosen starting point (see e.g. [6] or [9]). This alternative is more economical in

E.F. Brickell (Ed.): Advances in Cryptology - CRYPTO '92, LNCS 740, pp. 358-370, 1993.

its use of random bits, and can be optimized using test division etc. [2] to be significantly more efficient than the "uniform choice" method. Despite the practical advantages, an analysis of the incremental search method does not seem to have appeared before. Such an analysis will be the subject of this paper.

Before going into details, let us mention some related work: Bach [1] also uses the increment function in connection with primality testing. In his case, however, the increment function is used to generate the witnesses for the test, not the candidate numbers as in our case. Moreover, the results of [1] are concerned with the probability that the test accepts, *given that the input number is composite*, whereas we of course want the "reversed" probability, namely the probability that the input number is composite given that the test accepts. Finally, the general results of Cohen and Wigderson [3] on reducing the error probability of a BPP algorithm at the cost of a few more random bits would be applicable in our situation, more specifically to the Rabin test itself. This could be used to reduce the worst case probability of 1/4 dramatically. However, what the following shows is essentially that this is unecessary in our case: for most input numbers, the error probability is already exponentially small, and since we are interested in an algorithm for generating random prime numbers, not the decision problem as such, we only have to worry about the average case behavior.

We now give a more precise version of the algorithm we will look at. The description uses the notation above, and parameters s and t.

ALGORITHM PRIMEINC

1. Choose uniformly a number $n_0 \in M_k$. Put $n = n_0$.
2. Subject n to at most t iterations of the Rabin test. If n passes t iterations, output n and stop.
3. Otherwise (if n fails an iteration), put $n = n + 2$. If $n \geq n_0 + 2s$, output "fail" and stop, otherwise go to step 2.

It is well known that this algorithm can be optimized by test dividing by small prime numbers before applying the Rabin test to a candidate. [2] contains an analysis of the optimal number of primes to use. We have omitted this for simplicity because it is clear that, independently of the number of small primes used, the error probability of the optimized version would be at most that of PRIMEINC. This is because the test division can never reject a prime, and so can only give us a better chance of rejecting composites. It is clear that the error probability might even become much smaller by using test division, but this seems like a difficult problem to analyse.

2 The Error Probability

In this section we will look at the probability that PRIMEINC outputs a composite. Let E be the event that this happens, and let $q_{k,t,s} = Prob(E)$. Let $C_m \subset M_k$ be the set of composites for which the probability of passing the Rabin test is larger than 2^{-m}.

In [4], the following upper bound is proved on the size of C_m:

Lemma 1

If $m \leq 2\sqrt{k-1} - 1$, we have

$$|C_m|/|M_k| < a \sum_{j=2}^{m} 2^{m-j-(k-1)/j},$$

where $a = 10.32$.

Let $D_m = \{n \in M_k | \, [n..n+2s[\cap C_m \neq \emptyset\}$, for $m > 2$ and put (for convenience) $D_2 = \emptyset$. We clearly have:

Lemma 2

$D_m \subset D_{m+1}$ and $|D_m| \leq s \cdot |C_m|$.

This implies the following result:

Theorem 1

Let $s = c \cdot \log(x)$ for some constant c. Then

$$q_{k,t,s} \leq ck(0.5ck \sum_{m=3}^{M} P(C_m) 2^{-t(m-1)} + 0.7\ 2^{-tM})$$

where $M \geq 3$, and $P(C_m)$ denotes the probability that a random number in M_k is in C_m.

Proof

Identify D_m with the event that the starting point n_0 is in D_m. Then by Lemma 2, $P(D_m) \leq s \cdot P(C_m)$. By the fact that $D_m \subset D_{m+1}$, we have

$$q_{k,t,s} = \sum_{m=3}^{M} P(E \cap (D_m \setminus D_{m-1})) + P(E \cap \neg D_M) \quad (1)$$

$$\leq \sum_{m=3}^{M} P(D_m) P(E | \, (D_m \setminus D_{m-1})) + P(E | \, \neg D_M) \quad (2)$$

Now consider the probability that E occurs given that some fixed $n_0 \notin D_m$ was chosen. Then no candidate we consider will be in C_m, and so for any composite candidate, the probability of accepting it will be at most 2^{-mt}. The probability of outputting a composite clearly is maximal when all candidates are composite, in which case we accept one of the cadidates with probability at most $s \cdot 2^{-mt}$. This means that

$$q_{k,t,s} \leq s^2 \sum_{m=3}^{M} P(C_m) 2^{-t(m-1)} + s \cdot 2^{-tM} \quad (3)$$

$$\leq 0.5(ck)^2 \sum_{m=3}^{M} P(C_m) 2^{-t(m-1)} + 0.7ck2^{-tM} \quad (4)$$

The theorem follows.

It is clear that we can get explicit bounds on $q_{k,t,s}$ by combining Lemma 1 and Theorem 1, where the value of M should be optimized to get the best possible result. Table 1 shows concrete results obtained for $c = 1, 5$ and 10. The following proposition gives a very rough idea of how the bound behaves for large k.

Proposition 1

For any constants c (where $c\log(x) = s$) and t, $q_{k,t,s}$ as a function of k satisfies

$$q_{k,t,s} \leq \delta k^3 2^{-\sqrt{k}}$$

for some constant δ.

Proof

It is sufficient to show the result for $t = 1$. From Lemma 1 we get that $P(C_m)$ is less than a constant times $2^m m 2^{-\sqrt{k}}$, by observing that $-j - (k-1)/j$ is always less than $-2\sqrt{k-1}$. The lemma now follows immediately by inserting this in the result of Theorem 1.

3 The Failure Probability

To get a good bound on the error probability, we should choose c to be as small as possible. The obvious disadvantage is that this is likely to increase the probability that the algorithm fails.

To say something conclusive about this, we should clearly know something about the gaps between consecutive primes. Unfortunately, no unconditional result is known about this that would be strong enough for our purposes. However, Gallagher [5] has shown, based on Hardy and Littlewoods prime r-tuple conjecture [7], the following result:

Lemma 3

Under the prime r-tuple conjecture, we have: For any constant λ, the number of n's in the interval $[1..x[$, such that the interval $[n..n + \lambda\log(x)]$ contains precisely k primes, is

$$x\frac{e^{-\lambda}\lambda^k}{k!} \quad as \quad x \to \infty$$

Let $d(n)$ denote the distance from n to the next, larger prime. Then Lemma 3 means in particular (by taking $k = 0$) that for n chosen uniformly between 1 and x, the probability that $d(n) > \lambda\log(x)$ is $exp(-\lambda)$, as $x \to \infty$. This is the only part of Gallaghers result we will need in the following. It implies that the expected distance to the next prime is $\log(x)$. We have confirmed this experimentally (see Figure 1).

The following lemma gives a corresponding result for the interval $[x/2..x[$.

c	$k\backslash t$	1	2	3	4	5	6	7	8	9	10
1	100	0	7	13	18	22	26	29	32	34	36
	150	3	12	20	26	31	35	39	43	46	49
	200	5	17	25	33	39	44	49	53	57	61
	250	7	21	31	39	45	51	57	62	66	70
	300	9	24	35	44	51	58	64	70	75	79
	350	11	28	40	49	57	64	71	77	82	88
	400	13	31	44	54	63	70	77	84	90	95
	450	15	34	47	58	68	76	83	90	97	103
	500	17	37	51	63	72	81	89	96	103	110
	550	18	40	54	67	77	86	95	102	110	116
	600	20	42	58	70	81	91	100	108	115	123
5	100	0	3	9	14	18	21	24	27	29	32
	150	0	8	15	21	26	31	35	38	42	45
	200	1	12	21	28	34	39	44	48	52	56
	250	3	16	26	34	41	47	52	57	62	66
	300	5	20	31	39	47	53	59	65	70	75
	350	7	23	35	44	53	60	66	72	78	83
	400	9	26	39	49	58	66	73	79	85	91
	450	11	29	43	54	63	71	79	86	92	98
	500	13	32	46	58	68	77	85	92	99	105
	550	14	35	50	62	72	82	90	98	105	112
	600	16	38	53	66	77	86	95	103	111	118
10	100	0	2	7	12	16	19	22	25	27	30
	150	0	6	13	19	24	29	33	36	40	43
	200	0	10	19	26	32	37	42	46	50	54
	250	2	14	24	32	39	45	50	55	60	64
	300	4	18	29	37	45	51	57	63	68	73
	350	5	21	33	42	51	58	64	70	76	81
	400	7	24	37	47	56	64	71	77	83	89
	450	9	27	41	52	61	69	77	84	90	96
	500	11	30	44	56	66	75	83	90	97	103
	550	12	33	48	60	70	80	88	96	103	110
	600	14	36	51	64	75	84	93	101	109	116

Table 1: Shows $-\log_2$ of the upper bound on $q_{k,t,s}$ as a function of k and t and for $s = 1\log(x), 5\log(x)$ and $10\log(x)$.

Lemma 4

Assume the prime r-tuple conjecture holds, and that PRIMEINC is executed with $s = c\log(x)$. Let $p(c)$ be the probability that the algorithm fails. Then for any ϵ,

$$p(c) \leq 2exp(-2c) - exp(-2c - \epsilon) \quad as \quad x \to \infty$$

Proof

Put $\lambda = 2c$, and let $p_y(\lambda)$ be the probability that $d(n) > \lambda\log(x)$, when n is uniform between 1 and y. Then

$$p_x(\lambda) = 1/2(p(c) + p_{x/2}(\lambda))$$

The lemma now follows from Lemma 3, since for any $\epsilon > 0$ and all large enough x, $(\lambda + \epsilon)\log(x/2) > \lambda\log(x)$.

This means that for large x, the failure probability is essentially $exp(-2c)$, and certainly less than $2exp(-2c)$. By Theorem 1, the error probability increases at most quadratically with c; we can therefore choose values of c for which both error and failure probability are small.

As a realistic example, suppose we put $k = 300$ and $c = 10$. Then we fail with probability about 2^{-28}, or 1 in 200 million times, and with $t = 6$ we still get an error probability of at most 2^{-51}. It seems reasonable to make the error probability much smaller than the failure probability: a failure is detectable and can be recovered from, whereas as error is never detected, at least not by PRIMEINC itself.

4 The Output Distribution

This section is concerned with the quality of the output from PRIMEINC, in particular how the output is distributed over the possible primes. This is a critical point in cryptographic applications (e.g. RSA), where the output prime is to be kept secret. One should expect that an enemy knows which algorithm is being used to generate the primes, and it is natural to demand that this will not give him any significant advantage.

We suggest to use the entropy of the output distribution as a measure of its quality. This is natural, since it measures the enemy's uncertainty about the prime generated. From this point of view it is clear that the optimal output distribution is the uniform distribution over the primes in M_k, since it has maximal entropy. By the prime number theorem, this maximal value is about $H_u(x) = \log(x/2\log(x))$ for large x. Below, we will show that the entropy of the distribution output by PRIMEINC is very close to $H_u(x)$ for large x.

Since we will be using the prime r-tuple conjecture directly in the following, we quote it here:

Prime r-tuple conjecture

For a fixed r-tuple of integers $d = (d_1, ..., d_r)$, let $\pi_d(x)$ be the number of n's less than or equal to x, such that $n + d_i$ are all primes. For a prime p, let $\nu_d(p)$denote the number

of distinct residue classes modulo p occupied by numbers in d. The conjecture now says that

$$\pi_d(x) \sim S_d \frac{x}{\log(x)^r} \quad as \quad x \to \infty$$

where

$$S_d = \prod_p (\frac{p}{p-1})^{r-1} \frac{p - \nu_d(p)}{p-1}$$

For $r = 1$, this is just the prime number theorem. For $d = (0, 2)$, it is a statement about prime twins, and here $S_{0,2} \approx 1.32$.

By an argument similar to that of Gallagher, one can show that under this conjecture, the following holds:

Lemma 5

Let $F_h(x)$ denote the number of primes p such that $p \leq x$ and $p - q \leq h$, where q is the largest prime less than p. Then for any constant λ,

$$F_{\lambda \log(x)}(x) = \frac{x}{\log(x)}(1 - e^{-\lambda})(1 + o(1))$$

as $x \to \infty$.

Proof

By definition of the π_d's, it is clear that $F_h(x)$ can be found using inclusion/exclusion:

$$F_h(x) = \sum_{1 \leq d_1 \leq h} \pi_{0,d_1}(x) - \sum_{1 \leq d_1 < d_2 \leq h} \pi_{0,d_1,d_2}(x) + \sum_{1 \leq d_1 < d_2 < d_3 \leq h} \pi_{0,d_1,d_2,d_3}(x) - \cdots$$

Using the prime r-tuple conjecture, we get that

$$F_h(x) \sim \sum_{1 \leq d_1 \leq h} S_{0,d_1} \frac{x}{\log(x)^2} - \sum_{1 \leq d_1 < d_2 \leq h} S_{0,d_1,d_2} \frac{x}{\log(x)^3} \cdots$$

as $x \to \infty$. Using a modification of Gallagher's method from [5], one can show that

$$\sum_{1 \leq d_1 < \cdots < d_r \leq h} S_{0,d_1,\cdots,d_r} \sim \frac{h^r}{r!}$$

as $h \to \infty$, with a error term of $O(h^{r-1/2+\epsilon})$ for any $\epsilon > 0$. Inserting this in the above and choosing $h = \lambda \log(x)$ for a constant λ gives us that

$$F_{\lambda \log(x)}(x) \sim \frac{x}{\log(x)}(\frac{\lambda}{1!} - \frac{\lambda^2}{2!} + \frac{\lambda^3}{3!} - \cdots)$$

which immediately implies the lemma.

This lemma shows that the gaps between consecutive primes loosely speaking follows an exponential distribution. It corresponds nicely with Gallaghers result since a

Poisson distribution in statistics results from counting random events with exponentially distributed gaps occurring in a certain time slot.

Let $F'_h(x)$ denote the number of primes p such that $x/2 < p \leq x$ and $p - q \leq h$, where q is the largest prime less than p. It is a trivial consequence of Lemma 5 that

$$F'_{\lambda \log(x)}(x) = \frac{x/2}{\log(x)}(1 - e^{-\lambda})(1 + o(1))$$

as $x \to \infty$.

We now return to PRIMEINC and consider the restriction of the output distribution to the cases where the algorithm does not fail. For simplicity, we will look at an "ideal PRIMEINC" that never accepts a composite; we let $H(x)$ denote the resulting distribution over the primes in M_k. By Proposition 1, the distribution of the real PRIMEINC only assigns a negligible amount of probability mass [1] to composite numbers, and hence the difference between the entropy of this distribution and $H(x)$ will be negligible.

Theorem 2

If ideal PRIMEINC is executed with $s = c \cdot \log(x)$ for any constant c, then under the prime r-tuple conjecture,

$$\frac{H(x)}{H_u(x)} \sim 1$$

as $x \to \infty$.

Proof

The number of starting points n_0 that lead to non-failure will be called N. For a prime p, let $d(p)$ be the distance to the largest prime smaller than p, and $P(p)$ the probability that p is produced as output. Then

$$P(p) = \begin{cases} d(p)/2N, & \text{if } d(p) \leq 2c\log(x) \\ 2c\log(x)/2N, & \text{if } d(p) > 2c\log(x) \end{cases}$$

Now choose $n+1$ constants $0 = \lambda_0 < \lambda_1 < \ldots < \lambda_{n+1} = 2c$. By Lemma 5, the number of primes with $\lambda_i \log(x) \leq d(p) < \lambda_{i+1} \log(x)$ is

$$\frac{x}{2\log(x)}(e^{-\lambda_i} - e^{-\lambda_{i+1}})(1 + o(1))$$

This gives us the following:

$$\begin{aligned} H(x) &= \sum_{x/2<p\leq x} P(p)\log(1/P(p)) \\ &= \sum_{i=0}^{n} \sum_{\lambda_i \log(x) \leq d(p) < \lambda_{i+1}\log(x)} P(p)\log(1/P(p)) \quad + \sum_{d(p)\geq 2c\log(x)} P(p)\log(1/P(p)) \end{aligned}$$

[1] Here, a probability is called negligible if, as a function of k, it converges to 0 faster than any polynomial fraction

k	$H(x)$	$H_u(x)$	$H(x)/H_u(x)$
17	8.38	8.65	0.97
31	17.4	17.7	0.98
64	38.9	39.8	0.98
128	82.5	83.5	0.99
257	171	172	0.99

Table 2: Shows estimates for the entropy of the output from PRIMEINC with infinite search length compared with the maximal entropy value. The values are exact for $k = 17, 31$, and are based on a sample interval of length about 2^{27} for the rest of the values. Note that the entropy for any finite search length will be larger than the $H(x)$ value shown.

$$\begin{aligned} &\geq (1+o(1)) \cdot \left(\sum_i \frac{\lambda_i \log(x)}{2N} \log\left(\frac{2N}{\lambda_{i+1}\log(x)}\right) \frac{x}{2\log(x)} (e^{-\lambda_i} - e^{-\lambda_{i+1}}) \right. \\ &\quad \left. + \frac{2c\log(x)}{2N} \log \frac{2N}{2c\log(x)} \frac{x}{2\log(x)} e^{-2c} \right) \\ &= (1+o(1)) \frac{x}{4N} \left(\log\left(\frac{2N}{\log(x)}\right) \left(\sum_i \lambda_i (e^{-\lambda_i} - e^{-\lambda_{i+1}}) + 2ce^{-2c} \right) + A \right) \end{aligned}$$

where A is a constant. By Gallaghers result, $N \sim x/4(1-e^{-2c})$ as $x \to \infty$. We therefore get that

$$\frac{H(x)}{H_u(x)} \gtrsim \frac{1}{(1-e^{-2c})} \left(\sum_i \lambda_i (e^{-\lambda_i} - e^{-\lambda_{i+1}}) + 2ce^{-2c} \right)$$

By choosing a larger number of λ_i's with smaller intervals, we can make the sum over i arbitrarily close to $\int_0^{2c} y e^{-y} dy = 1 - e^{-2c} - 2ce^{-2c}$, which implies the result of the theorem since $H(x)/H_u(x)$ is always smaller than 1.

Thus, for large x, the entropy is very close to maximal independently of the value of c. In fact one can show by an argument similar to the one for Theorem 2 that if we take $c = \infty$, i.e. allow the algorithm to run indefinitely until a prime is found, the resulting entropy would still be close to maximal in the same sense as in Theorem 2. Taking any finite value of c means that we are limiting the probability for any single prime to a certain maximum. Intuitively, this should make the distribution closer to the uniform one, and so the entropy should be close to maximal for any finite value of c; Theorem 2 confirms this intuition.

We have done some numerical experiments to estimate how fast the convergence in Theorem 2 is. Table 2 shows the entropy for various values of $k = log_2(x)$ and an infinite search length. For $k \geq 64$ the values are estimates based on primes in an interval of length about 2^{27}. Already for small values of k, the entropy is close to maximal, and the value clearly tends to increase with increasing k, in accordance with Theorem 2.

To illustrate the exponential distribution of gaps between primes, we plotted the frequency of gaps between primes in various intervals against their length. A logarithmic scale was used, such that a straight line should result, according to the exponential distri-

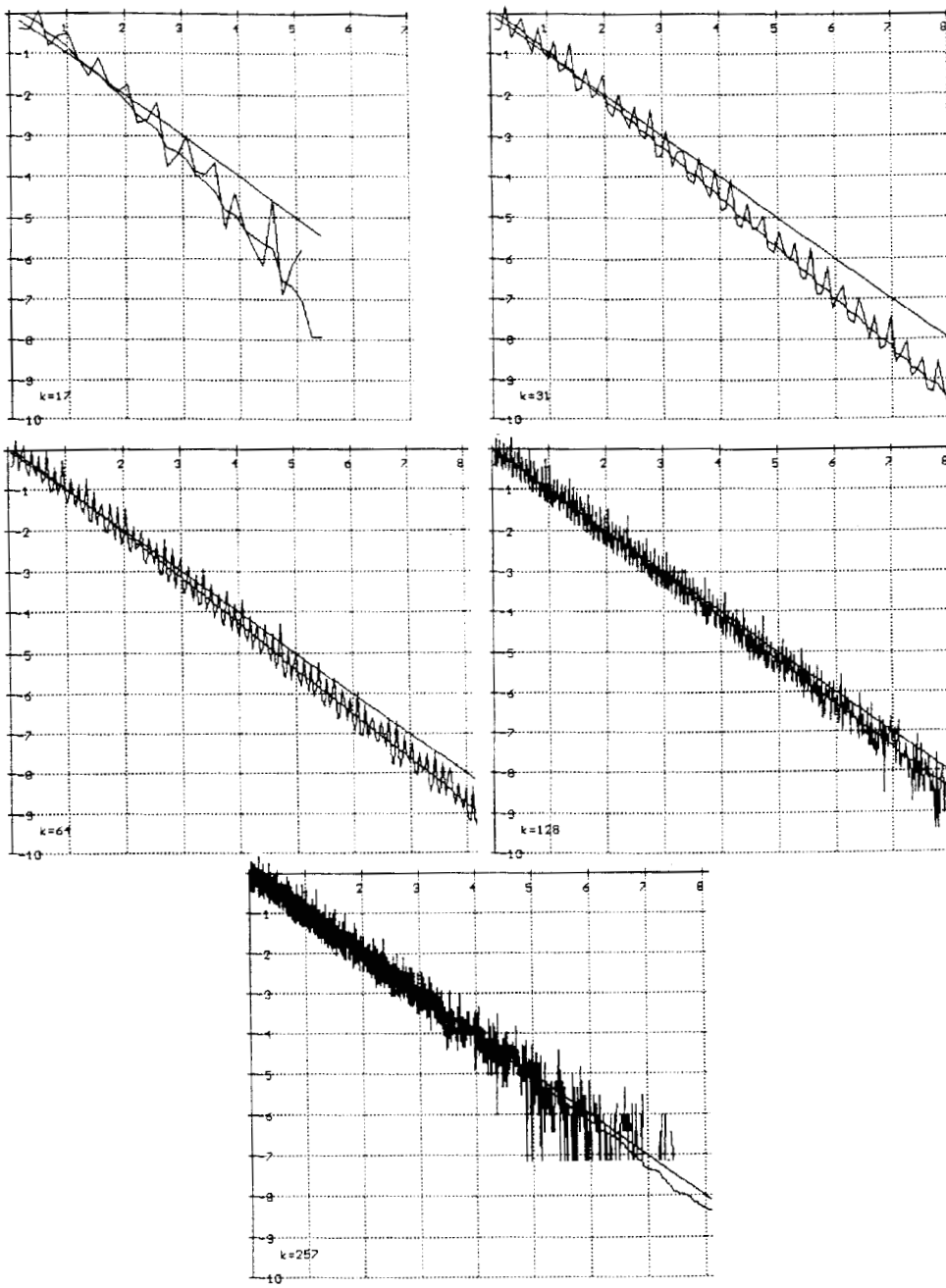

Figure 1: The graphs here show the distribution of gaps between primes in M_k and distances to the next prime from a random starting point. The "sawtooth" graphs represent the gaps. Results for $k = 17$ and 31 are exact, the rest are based on samples of length about 2^{27}. We plotted the distances divided by $\log(x)$ on the horizontal scale against the logarithm of the frequences of distances. This normalizes the graphs such that they can be directly compared. The distributions predicted by Lemma 4 and 5 are representred by the straight lines

bution. The result can be seen in Figure 1. The straight line shown represents the values predicted by Lemma 5.

5 Getting rid of Failures

There are two obvious ways to remove the (unlikely) failures in PRIMEINC:

Choose a new random starting point

- and try a new search. The question is, however, how many times we will have to do this to make (almost) sure that we find a probable prime, and what will happen to the error probability in that case?

Consider therefore an algorithm that simply iterates PRIMEINC with some fixed $k, t, s = c \cdot \log(x)$ until a probable prime is output.

By the prime number theorem, each starting point n_0 is prime with probability $O(k^{-1})$. Therefore, there is exponentially small (in k) probability that the number of iterations is larger than, say, k^2. This implies an upper bound on the expected running time. On the other hand, the error probability for k^2 iterations is at most

$$k^2 q_{k,t,s} \leq O(k^5 2^{-\sqrt{k}})$$

by Proposition 1, and so still asymptotically smaller than any polynomial fraction.

These arguments only give a very rough upper bound on the error probability and running time, but they add some theoretical justification to the algorithm, as they are unconditional results (independent of the prime r-tuple conjecture).

One disadvantage of this approach, however, is that is seems very difficult to analyse the entropy of the output distribution.

Let the search go on indefinitely

- until a probable prime has been found. Let $q_{k,t,\infty}$ denote the error probability of this algorithm. It is clear that $q_{k,t,\infty}$ can be no larger than the error probability of an algorithm that first runs PRIMEINC with some finite $s = c\log(x)$, outputs the number PRIMEINC produces (if one is found) and outputs a composite in the event of a failure. By Lemma 4, this implies that under the prime r-tuple conjecture,

$$q_{k,t,\infty} \leq p_{k,t,c\log(x)} + e^{-2c}(1 + o(1))$$

for any constant c. Thus we cannot say anything unconditional about the error probability, on the other hand we can in this case analyse the entropy of the output distribution, as mentioned in the remarks following Theorem 2.

The numerical evidence we collected leads us to conjecture that e^{-2c} is a good overestimate of the failure probability already for quite small values of k, say, larger than 64. Figure 1 shows the actual distribution of the distance to the next prime in some example intervals, compared with the values predicted by Lemma 4. Assuming e^{-2c} as an upper bound on the probability, we get that $q_{k,t,\infty} \leq p_{k,t,c\log(x)} + e^{-2c}$.

Using Theorem 1 and optimizing for the value of c, one can get concrete estimates for $q_{k,t,\infty}$. The results are shown in Table 3.

$k\backslash t$	1	2	3	4	5	6	7	8	9	10
100	0	4	9	13	16	20	22	24	27	29
150	1	8	14	20	24	28	32	35	38	41
200	3	12	19	26	31	36	40	44	48	52
250	4	15	24	31	37	43	48	53	57	61
300	6	18	28	36	43	49	55	60	65	70
350	8	21	32	41	49	55	61	67	73	78
400	9	24	36	45	53	61	67	73	79	85
450	11	27	39	49	58	66	73	80	87	92
500	12	29	43	53	63	72	79	86	93	99
550	14	32	46	58	67	76	84	92	99	105
600	15	35	49	61	72	81	90	98	105	111

Table 3: Shows $-\log_2$ of the upper bound on $q_{k,t,\infty}$ as a function of k and t *assuming* that the failure probability of PRIMEINC with $s = c\log(x)$ is at most e^{-2c}.

6 Conclusion

We have shown some explicit upper bounds on the error probability of the PRIMEINC algorithm. Together with the prime r-tuple conjecture, these bounds show that we can choose the maximal length of the search such that both the error probability and the failure probability are in practice negligible, even for quite small values of t.

Moreover, under the prime r-tuple conjecture, we have seen that the uncertainty about the prime produced by PRIMEINC is very close to maximal, for any value of c, including $c = \infty$. This strongly suggests that, compared to a uniform choice, there no significant loss of security when using PRIMEINC in cryptographic applications where secret primes are required.

For the case where one iterates PRIMEINC if no probable prime is found, we have seen unconditional results on the running time and error probability. In the case where in stead the search is allowed to go on indefinitely, the prime r-tuple conjecture implies results on both running time, error probability and entropy.

References

[1] E.Bach: *Realistic analysis of som randomized algorithms*, Proc. of STOC 87.

[2] J.Brandt, I.B.Damgård and P.Landrock: *Speeding up prime number generation*, Proc. of Asiacrypt 91, Springer Verlag Lecture Note Series.

[3] L.Cohen and A.Wigderson: *Dispersers, deterministic amplification, and weak random sources*, Proc. of FOCS 89.

[4] I.B.Damgård, P.Landrock and C.Pomerance: *Average case bounds for the strong probable prime test*, submitted to Math. Comp., available from authors.

[5] P.X.Gallagher: *On the distribution of primes in short intervals*, Mathematica 23 (1976) pp.4-9.

[6] J.Gordon: *Strong primes are easy to find*, Proc. of Crypto 84, Springer Verlag Lecture Note Series.

[7] G.H.Hardy and J.E.Littlewood: *Some problems of 'Partitio Numerorum': III. On the expression of a number as a sum of primes*, Acta Mathematica 44 (1922), pp.1-70.

[8] U.Maurer: *The generation of secure RSA products with almost maximal diversity*, Proc. of EuroCrypt 89, Springer Verlag Lecture Note Series.

[9] FIPS publications XX, Digital Signature Standard (DSS).

[10] M.O.Rabin: *Probabilistic algorithms for testing primality*, J. Number Theory 12 (1980) pp.128-138.