Annales Univ. Sci. Budapest., Sect. Comp. 48 (2018) 169–179

PRIMALITY PROOFS WITH ELLIPTIC CURVES: FACTORING WITH POLLARD'S ρ METHOD

Gábor Román (Budapest, Hungary)

Communicated by Antal Járai

(Received February 3, 2018; accepted May 11, 2018)

Abstract. In this article, we are going to study the factoring probability of Pollard's ρ method and its efficiency during the elliptic curve primality proving. We will describe factoring strategies and heuristics involving the parameters controlling Pollard's ρ method.

1. Introduction

The elliptic curve primality proving algorithm (see [1]) proves the primality of an input number using a recursive procedure. During a recursive step, the recursive calls are based on the factorisation of computed curve orders. The success of the method hugely depends on the effectiveness of the factorisation method which we apply on these orders.

Naturally the first method which one applies is trial division. This removes the small prime factors from the curve orders. A good candidate for the next factorisation step is Pollard's ρ method, which we are going to study in this paper. One can describe the 'gain' during a factorisation involved in a recursive step, as the product of the small primes which we have factorised from the curve order. The authors of [3] gave a heuristic for the expected value of this 'gain' in the case of trial division. In the second section we are going to give a heuristic to estimate the expected value of the 'gain' and study the probability of success of

Key words and phrases: Elliptic curve primality proving, factorisation, Pollard's ρ method.
2010 Mathematics Subject Classification: 11Y11.
https://doi.org/10.71352/ac.48.169

extracting a prime during Pollard's ρ method, furthermore conduct experiments to observe how the 'gain' could behave in reality. In the third section we are going to present strategies for the application of Pollard's ρ method during the elliptic curve primality proving algorithm.

The algorithm which we are going to analyse is from Cohen [4], we have just changed the x^2+1 polynomial to the x^2+c polynomial with an integer constant c, and added a $B>0$ batching threshold. We are going to examine this particular algorithm, modified ones could yield different results.

Algorithm 1. (Pollard's ρ) Given a composite number n, an integer constant c and a batching threshold B, this algorithm tries to find a non-trivial factor of n.

1. (Initialize) Set $y \leftarrow 2$, $x \leftarrow 2$, $x_1 \leftarrow 2$, $k \leftarrow 1$, $l \leftarrow 1$, $P \leftarrow 1$, $b \leftarrow 0$.
2. (Accumulate product) Set $x \leftarrow x^2+c \pmod n$, $p \leftarrow P(x_1-x) \pmod n$ and $b \leftarrow b+1$. If $b=B$, compute $g \leftarrow \gcd(P,n)$, then if $g>1$ go to step 4 else set $y \leftarrow x$ and $b \leftarrow 0$.
3. (Advance) Set $k \leftarrow k-1$. If $k \neq 0$ go to step 2. Otherwise, compute $g \leftarrow \gcd(P,n)$. If $g>1$ go to step 4 else set $x_1 \leftarrow x$, $k \leftarrow l$, $l \leftarrow 2l$, then repeat k times $x \leftarrow x^2+c \pmod n$, then set $y \leftarrow x$, $b \leftarrow 0$ and go to step 2.
4. (Backtrack) Repeat $y \leftarrow y^2+c \pmod n$, $g \leftarrow \gcd(x_1-y,n)$ until $g>1$. If $g<n$ output g, otherwise output a message saying that the algorithm fails. Terminate the algorithm.

2. Factoring probability and the expected value of the 'gain'

The authors of [3] wrote that the expected value of the 'gain' is supposed to be $\sim \ln b(n)$ when one applies trial division up to a $b(n)$ bound in case of an n natural number. Their heuristic is the following: for each prime p the probability that p divides a given curve order m is $1/p$ and this results in a 'gain' of $\ln p$. Hence the expected value of the total 'gain' is

$$G(b(n)) = \sum_{p\in\mathbb{P}, p\leq b(n)} \frac{\ln p}{p} \sim \int_2^{b(n)} \frac{\ln x}{x}\frac{1}{\ln x}\,dx \sim \ln b(n).$$

The authors of the mentioned article expect that the 'gain' will still remain $\sim \ln b(n)$ for other factoring methods.

We are going to study the expected value of the 'gain' while applying Pollard's ρ method. Bach gave asymptotic description for the factoring probability of Pollard's ρ method in his article [2]. We present three theorems from this article now. Define polynomials $f_i \in \mathbb{Z}[x,y]$, by $f_0 = x$ and $f_{i+1} = f_i^2 + y$ for $i > 0$.

Theorem 1. *Fix $k \geq 1$. Choose x and y as random subjects to $0 \leq x, y < p$. Then the probability that for some $i, j < k$, $i \neq j$, $f_i(x,y) \equiv f_j(x,y) \pmod{p}$ is at least $\binom{k}{2}/p + O(p^{-3/2})$ as $p \to \infty$.*

Theorem 2. *Fix $k \geq 1$. Let n have two prime factors p and q with $p < q$. Then $gcd(f_{2i+1} - f_i, n) \neq 1, n$ for some $i < k$ with probability at least $\binom{k}{2}/p + O(p^{-3/2})$ as $p \to \infty$.*

Theorem 3. *Let n have two prime divisors p and q with $p < q$. Let $k(p) = \lfloor \frac{1}{4} \log_2 p \rfloor$. Then there is some $i < k(p)$ such that $gcd(f_{2i+1} - f_i, n) \neq 1, n$ with probability at least $\Omega(\ln^2 p)/p$.*

Based on this result, we can give a simple heuristic to estimate the expected value of the 'gain' for $n = pq$ semiprimes. There exists a $c > 0$ constant and a $p_0 \in \mathbb{N}$ limit, for which the probability of success is greater than $c(\ln^2 p)/p$ when p is greater than p_0. During a successful factorisation we get a divisor of n, which is not one and the number itself. In this case it could be p or q so the factorisation will result in a 'gain' at least $\ln p$. From these, and the previous heuristic, we get that

$$G(b(n)) \geq \sum_{p \in \mathbb{P}, p_0 < p \leq b(n)} \frac{c \ln^3 p}{p} \sim \int_{p_0}^{b(n)} \frac{\ln^3 x}{x} \frac{1}{\ln x}\, dx \sim \ln^3 b(n)$$

when the $b(n)$ bound is big enough.

Bach gave results for $n = pq$ numbers, which are of great interest from the viewpoint of cryptography. (Consider the RSA scheme for example.) The next step is to extend this result to square-free numbers. The factoring probability for arbitrary numbers is still an open question. We are going to build heavily onto the results of Bach.

Proposition 1. *Let $n = p_1 p_2 \ldots p_l$ with $p_1 < p_2 < \ldots < p_l$ primes, $l \geq 2$. Let $k(p_1) = \lfloor \frac{1}{4} \log_2 p_1 \rfloor$. Then there is some $i < k(p_1)$ such that $gcd(f_{2i+1} - f_i, n) \neq \neq 1, n$ with at least*

$$\frac{\Omega(\ln^2 p_1)}{p_1} \left(1 - \frac{1}{\sqrt{p_1}}\right)^{l-1}$$

probability.

Proof. (Proposition 1) The proof is the extension of the proof of theorem 3 in the article of Bach. First we have to sharpen a statement of Bach. Consider $f_{2i+1} - f_i \pmod q$ for given q prime. This polynomial splits into absolutely irreducible factors of degrees $d_1, \ldots, d_m$, where $\sum d_j = 2^{2i+1}$, so it vanishes modulo q with probability at most

$$\frac{1}{q^2}\sum_{j=1}^{m}\left[q + 2\sqrt{q}\binom{d_j - 1}{2}\right] \leq \frac{m}{q} + \frac{(\sum d_j)^2}{q^{3/2}} \leq \frac{2^{2i+1}}{q} + \frac{2^{4i+2}}{q^{3/2}}$$

according Weil's theorem. Summing this over $i = 0, \ldots, k-1$ results in

$$\frac{2}{3}\frac{4^k - 1}{q} + \frac{4}{15}\frac{16^k - 1}{q^{3/2}}$$

which is less than or equal to

$$\frac{10}{15}\frac{\sqrt{q} - 1}{q} + \frac{4}{15}\frac{q - 1}{q^{3/2}}$$

because $k \leq \frac{1}{4}\log_2 q$. So the probability for some $i < k$, $f_{2i+1} \equiv f_i \pmod q$ is less than $1/\sqrt{q}$.

As for the extension, define the

$$\alpha_k(p) := \exists i < j < k : f_i \equiv f_j \pmod p$$

and

$$\beta_k(p) := \exists i < k : f_i \equiv f_{2i+1} \pmod p$$

conditions. Using these, the probability of success during the factorisation of n is at least

$$P(\alpha_k(p_1) \wedge \neg\beta_k(p_2) \wedge \ldots \wedge \neg\beta_k(p_l))$$

where $P(\neg\beta_k(q)) \geq 1 - 1/\sqrt{q}$ according our previous calculations. These conditions are independent by the Chinese remainder theorem, so this is at least

$$P(\alpha_k(p_1))\prod_{i=2}^{l}\left(1 - \frac{1}{\sqrt{p_i}}\right) \geq P(\alpha_k(p_1))\left(1 - \frac{1}{\sqrt{p_1}}\right)^{l-1}$$

from where, using theorem 1, we get our result by substituting $\Omega(\ln^2 p_1)/p_1$ in the place of $P(\alpha_k(p_1))$. ■

2.1. Experiments for measuring the 'gain'

Taking into consideration the hidden c constant in the ordo and the p_0 prime limit, one sees that these results are merely theoretic. Conducting experiments

reveal that the 'gain' is around the logarithm of the factoring limit for smaller numbers in case of some polynomials. This coincide with what the authors expected in [3].

The experiment is the following. We take a factoring limit and apply the method to the odd primes below this limit. Every time when the method is successful for a p prime, we accumulate the $\ln p/p$ value in a sum. We compute separate sums for different factoring limits. We increase the factoring limit from 2^{13} up to 2^{20} with 2^{13} steps gaining 128 sums. The resulting sums reveal to us how the 'gain' behaves as the factoring limit increases.

We have conducted the experiment using the polynomials x^2+1, x^2+2 and x^2+3. The iteration limit (see section 3) is taken to be the square root of the factoring limit, and the batching parameter is taken from section (3.3). The results of the experiments were almost identical for every polynomial. One can be seen in figure 1.

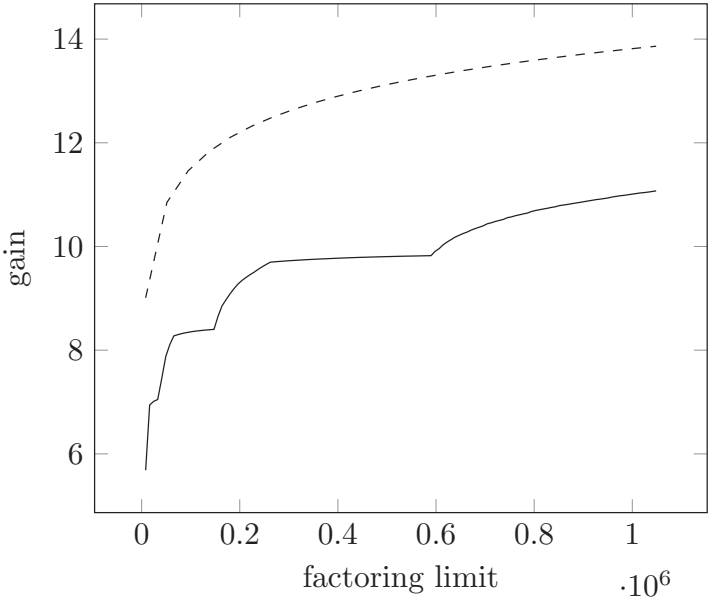

Figure 1: The behavior of the 'gain'. The dashed line is the natural logarithm function and the solid line represents the computed sums when the x^2+1 polynomial was applied during the method.

Of course in reality the method performs better, because the prime factors which are found by it aren't bounded by a factoring limit. Also one can vary the probability of success with the modification of the applied iteration limit.

3. Factoring strategy during ECPP with Pollard's ρ method

There are three parameters which one can vary during the application of Pollard's ρ method: the constant of the applied polynomial, a limit on the iteration count and the batching parameter. The method could compute for a decent amount of time, so it is mandatory to impose a limit on the iteration count.

One could apply the method in an iterative deepening fashion: taking the first curve order, we apply the method with one polynomial but using an iteration limit. If we were not successful, we move on to the next curve order, and factorise in the same fashion. If there are no more curve orders left, we can go back to a previously tried one, and continue where we left off, yet again with a limit on the iteration count. (This requires us to store the variables which are used during the previous call.) The manner in which we choose the curve order where we want to try harder is still an open question.

But it is not reasonable to let the method run for too long time, as one can see in section (3.1). A better strategy involves the application of multiple polynomials to find prime factors up to a limit, see section (3.2). If this method fails, one could still try Pollard's $p-1$ or the elliptic curve factorisation.

To optimise the runtime of the method, one can fine tune the applied batching parameter as it is described in section (3.3).

3.1. Distribution of the iteration count

Pollard's ρ method computes with the $x_{m+1} = f(x_m)$ iteration, where f is our selected polynomial. Starting this iteration with an $x_0 = n$ natural number, sooner or later a cycle will appear in the $x_m \pmod n$ iteration. If n has a p prime factor, then there will be a cycle much sooner in the $x_m \pmod p$ iteration presumably. Now setting $x_0 = p$ for a p prime and executing the method until this prime is introduced in the greatest common divisor, we can obtain the iteration count for the given prime. Given that we have chosen our polynomial right, it can be sought as a random map, so this given prime will be factored out from any natural number which has it as a divisor, with nearly the same iteration count.

Let $m(p)$ be the iteration count of the examined method for a given p prime. We have executed the method with the x^2+1, x^2+2 and x^2+3 polynomials for odd primes below $l = 2^{20}$ resulting in $k = 82024$ samples, and investigated the distribution of these $m(p)/\sqrt{p}$ samples. We took the $16\sqrt{l}$ value as a limitation on the number of iterations during one execution of the method,

so the computations would not take so long. The appropriate bin width is calculated with $2(Q_3 - Q_1)/\sqrt[3]{k}$, where Q_3 and Q_1 are the third and the first quartiles of the ordered samples. The results can be seen in figure 2.

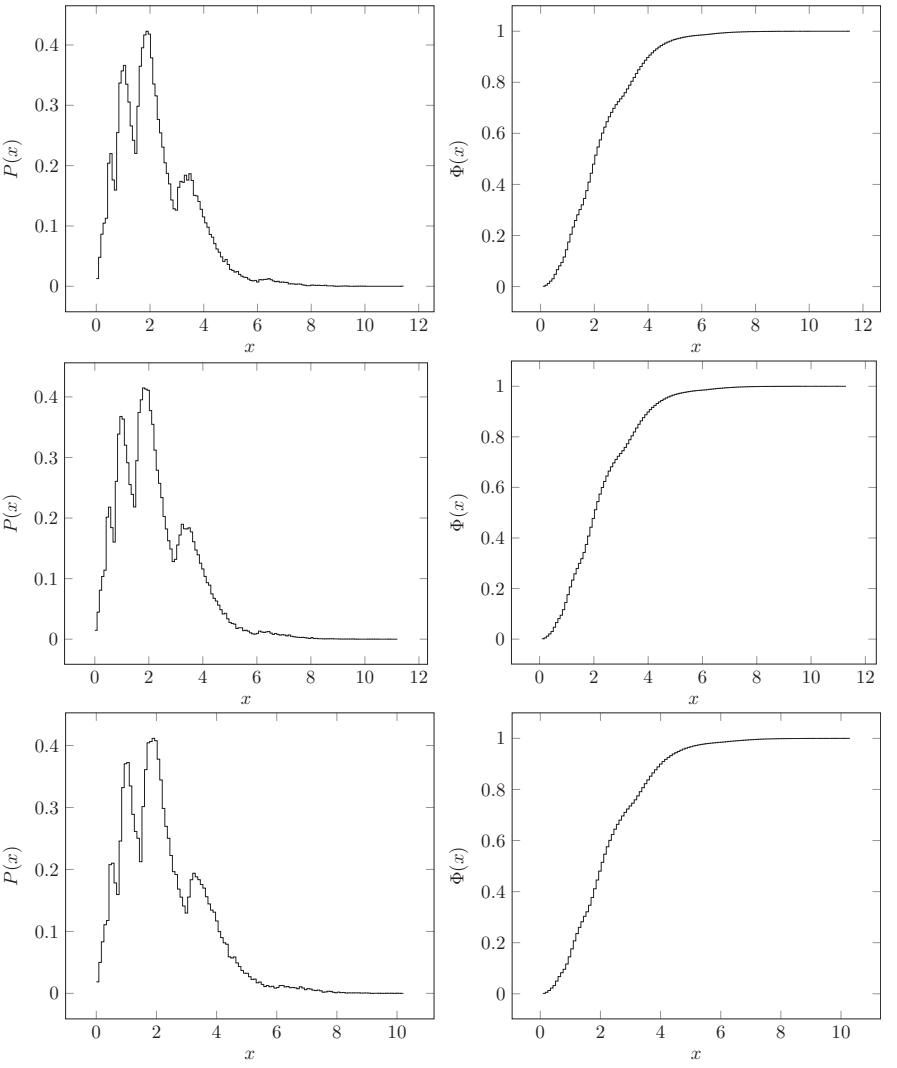

Figure 2: The histograms and the distributions of the computed data for the polynomials $x^2 + 1, x^2 + 2$ and $x^2 + 3$ from the top to the bottom.

Repeating the computations with different polynomials yielded near identical distributions. The results coincide with the results of Knuth [5], who wrote that $m(p)$ has an average value of about $2\sqrt{p}$. From these observations, one can see that it is reasonable to carry out the computations until $c\sqrt[4]{n}$, where $c = 6$ or $c = 8$ is an acceptable choice.

3.2. Finding prime factors up to a limit

One could find every prime factor up to a given limit with Pollard's ρ method, similarly as it can be seen in Maple$^{\text{TM}}$ with Pollard's $p - 1$ method. However, in the case of Pollard's ρ method, we achieve this by executing the method multiple times with different polynomials. This is useful when one wants to obtain every possible prime factor of a number up to a given limit. Of course, because of the nature of Pollard's ρ method, one could gain larger primes than the posed limit, but up to this limit every prime factor will be found. The process is the following.

If one performs Pollard's ρ on the p primes up to an n natural number applying $2\sqrt{p}$ iteration limit, then after processing every prime approximately half of the them will be factorised successfully, independently from the iterated polynomial. The result will be similar if one takes $2\sqrt{n}$ for the iteration limit through this process, though there will be more primes which are factorised successfully.

Based on this, first we "sieve" the primes with an f_1 polynomial up to n, then we sieve the remaining primes with an $f_2 \neq f_1$ polynomial, and so on. In every round, we apply $2\sqrt{n}$ iteration limit and we use a polynomial which we haven't used so far. With every sieving, the number of the remaining primes halves, so we quickly arrive to a point where only a few primes remain. Now we multiply these remaining primes to get an N constant.

Finding prime factors up to n is now done in the following fashion. We did the aforementioned process for this n natural number, we used the polynomials $f_1, f_2, \ldots, f_k$, and gained the product of the remaining primes N. Now let's say, that we have an m natural number and we want to find every prime factor of m which is smaller than n. We perform Pollard's ρ method on this number with the $2\sqrt{n}$ iteration limit using the $f_1, f_2, \ldots, f_k$ polynomials. If we weren't successful up until this point, then we compute $\gcd(m, N)$, which is supposed to factorise out the remaining primes from m which are smaller than n.

We have done the described computations for $n = 2^{20}$. After applying $f(x) = x^2 + c$ with $c = 1, 2, \ldots, 6$, only 34 primes remained. Of course, one can vary the iteration limit, to adjust the number of the required polynomials and the size of N.

3.3. Heuristic for a batching parameter

Denote the running time of our multiplication algorithm with $\mu \in \mathbb{N} \to \mathbb{R}$ and the running time of our gcd algorithm with $\gamma \in \mathbb{N} \to \mathbb{R}$, where the arguments of these functions should be interpreted in bits. We will only concentrate on these operations during the analysis of algorithm 1, because these are the dominant ones from the viewpoint of runtime.

Let's look at a part of the execution, which starts at the "accumulate product" step and ends after the gcd check and the multiplications in the "advance" step. Denote this part of the execution as a cycle. Now the algorithm performs the accumulate product step k times in a cycle, where k is a power of two and increases to the consecutive power of two at the end of every cycle, except the first. There are gcd checks interleaved in these cycles at every Bth step and at the end. Depending on the size of the B threshold, there could be cycles where we do not perform any gcd operations in the accumulate product step, only in the advance step.

The required operations in an "empty" cycle will consist of two modular multiplications in every execution of the accumulate product step, finished with a gcd operation and k modular multiplications in the advance step. This gives us

$$t_1(k) := 4k\mu(\ln n) + \gamma(\ln n) + 2k\mu(\ln n),$$

because a modular multiplication consists of a multiplication and a modular correction. The required operations in a not-empty cycle will consist of the operations in the empty cycle plus the interleaved gcd operations. In this case, we have

$$t_2(k) := 6k\mu(\ln n) + \left(\left\lfloor \frac{k}{B} \right\rfloor + 1\right)\gamma(\ln n),$$

because there will be $\lfloor k/B \rfloor$ interleaved gcd operations.

Let's presume that there will be m cycles before the algorithm starts backtracking. There will be two cycles where $k = 1$, so the required operations for these cycles will be

$$t_1(1) + \sum_{l=0}^{\lfloor \log_2 B \rfloor} t_1(2^l) + \sum_{l=\lfloor \log_2 B \rfloor + 1}^{m-1} t_2(2^l)$$

at most, because the number of empty cycles will be $\lfloor \log_2 B \rfloor$. This is equal to

$$(m+1)\gamma(\ln n) + 6(2^m - 1)\mu(\ln n) + \gamma(\ln n) \sum_{l=\lfloor \log_2 B \rfloor + 1}^{m-1} \left\lfloor \frac{2^l}{B} \right\rfloor$$

where

$$\sum_{l=\lfloor \log_2 B \rfloor + 1}^{m-1} \left\lfloor \frac{2^l}{B} \right\rfloor < \frac{2^m - 1}{B},$$

although the sum could be empty. This happens in the case when we split the input during the empty cycles. For this end, let's presume from now on that $\lfloor \log_2 B \rfloor + 1 \leq m - 1$ holds.

As for the backtracking, one should note that the required number of iterations during the backtrack step will be at most B. During one of these repetitions, a modular multiplication and a gcd operation occurs, so

$$B(2\mu(\ln n) + \gamma(\ln n))$$

operations will be performed in the backtrack step at most. Summing the parts, we get that for an algorithm execution with m cycles

$$\left(\frac{2^m - 1}{B} + B + m + 1\right)\gamma(\ln n) + (6(2^m - 1) + 2B)\mu(\ln n)$$

operations are required at most. Here we have control over B only, so the

$$\left(\frac{2^m - 1}{B} + B + 2B\frac{\mu(\ln n)}{\gamma(\ln n)}\right)\gamma(\ln n)$$

part is interesting to us. Here we have to minimise B while n and m are fixed. Examining the expression inside the parentheses, we get that it has a minimum when

$$B = \sqrt{\frac{2^m - 1}{2\frac{\mu(\ln n)}{\gamma(\ln n)} + 1}}$$

if B is positive. From the analysis of Cohen, we know that as the smallest p prime factor of n tends to infinity, the required cycles performed will be asymptotically $\log_2 \sqrt{p}$ which is at most $\log_2 \sqrt[4]{n}$, so

$$B = \sqrt{\frac{\sqrt[4]{n} - 1}{2\frac{\mu(\ln n)}{\gamma(\ln n)} + 1}} \sim \sqrt[8]{n}$$

because γ grows much faster than μ. By these heuristics it seems better to choose B to be $\sqrt[8]{n}$ instead of a constant value which we might have fixed by feeling. Take note that this choice of B parameter makes our $\lfloor \log_2 B \rfloor + 1 \leq m - 1$ requirement valid if n is sufficiently big.

References

[1] **Atkin, A.O.L. and F. Morain,** Elliptic curves and primality proving, *Math. Comp.*, **61(203)** (1993), 29–68.

[2] **Bach, E.,** Toward a theory of Pollard's rho method, *Information and Computation*, **90** (1991), 139–155.

[3] **Bosma, W., E. Cator, A. Járai and Gy. Kiss,** Primality proofs with elliptic curves: Heuristics and analysis, *Annales Univ. Sci. Budapest., Sect. Comp.*, **44** (2015), 3–27.

[4] **Cohen, H.,** A Course In Computational Algebraic Number Theory, *Springer–Verlag*, third, corrected printing (1996).

[5] **Knuth, D.E.,** *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, Addison Wesley Longman, third edition (1998).

G. Román
Department of Computer Algebra
Faculty of Informatics
Eötvös Loránd University
H-1117 Budapest
Pázmány Péter sétány 1/C
Hungary
romangabor@caesar.elte.hu