Computing partial information out of uncomputable one – The first digit of 2^n at base 3 as an example

Mika Hirvensalo
Juhani Karhumäki

Turku Centre for Computer Science

TUCS Technical Reports

No 446, February 2002

Computing partial information out of uncomputable one – The first digit of 2^n at base 3 as an example

Mika Hirvensalo
Juhani Karhumäki

Turku Centre for Computer Science and
Department of Mathematics,
University of Turku,
FIN-20014 Turku, Finland
{mikhirve,karhumak}@cs.utu.fi

Supported under the grant 44087 of the Academy of Finland.

Turku Centre for Computer Science
TUCS Technical Report No 446
February 2002

ISBN 952-12-0958-5
ISSN 1239-1891

Abstract

The problem of finding 2^n when n is given is mathematically trivial, but computationally *intractable*. We show how to compute the most significant digit of the ternary representation of 2^n in polynomial time.

TUCS Research Group
Theory Group

1 Introduction

Exponent functions are mathematically, in many aspects, among the simplest ones. For example, their derivatives are particularly easy to compute. On the other hand, from algorithmic point of view they are terrible. Although the value of an exponent function, say 2^n for example, is very easy to compute in theory, it is out of the capacity of modern computers in practice. More precisely, the value of such a function cannot be computed by an algorithm running in polynomial time with respect to the size of the input, i.e. the length of n. This, indeed, follows since the output is exponential in terms of the length of n. The same applies for many other numbers, for example, for the well-known Fibonacci numbers.

Having this situation a natural question is: Can we compute some partial information about an exponent function in polynomial time, for example its length at a given base? This is the problem area we pose here. We concentrate to a very simple function: for a given n consider the ternary representation of 2^n, i.e., the function $n \mapsto f(n) = (2^n)_{\mathbf{3}}$. We believe this already captures difficulties encountered in our questions.

Let $f(n)$ be defined as above. What can be computed easily is the last, or even last k, for a fixed k, digits of $f(n)$. Indeed, such values form an ultimately periodic sequence of numbers (cf. Figure 1), so that the threshold and the period can be computed in constant time. Hence the problem reduces to the computation $n \mapsto n \pmod p$, where p is the precomputed period. Therefore it can be done in linear time.

On the other hand, to compute even the length of $f(n)$ in polynomial time is not obvious. We shall show, that this can be done, and moreover, that the first digit of $f(n)$ can also be computed in polynomial time. The algorithm itself is not complicated, nor difficult to analyze. However, to prove its correctness seems to require deep results on transcendental numbers.

To compute a fixed number of first digits of $f(n)$ does no seem to be essentially more difficult question. The situation, however, changes if we ask to compute the "middle" digit of $f(n)$. We do not know how to do it in polynomial time. A more general question is: given $n \in \mathbb{N}$ and $i \leq \lfloor \log_3 f(n) \rfloor + 1$, can we compute the ith digit of $f(n)$ in polynomial time?

Our problems can be seen as examples of interesting general algorithmic questions. How much partial information about intractable algorithmic problem can be computed in practice, that is in polynomial time?

2 Preliminaries

In this section we introduce some definitions and notations used in the following sections.

For any real number x, notation $\lfloor x \rfloor$ stands for the largest integer M

satisfying $M \leq x$, and $\log_d x$ stands for the d-ary logarithm of x. We also use the standard notation $\ln x$ for the natural logarithm of x.

Let $d > 1$ be an integer. Each natural number M admits a unique representation as

$$M = a_0 d^{\ell-1} + a_1 d^{\ell-2} + \ldots + a_{\ell-2} d^1 + a_{\ell-1}, \tag{1}$$

where $a_i \in \{0, 1, \ldots, d-1\}$ and $a_0 \neq 0$. The string $a_0 a_1 \ldots a_{\ell-1}$ is called the *d-ary representation of M*, and denoted by $M_{\mathbf{d}}$. The *length* of d-ary representation of M is defined to be the length of string $M_{\mathbf{d}}$ and denoted by $|M_{\mathbf{d}}|$. Notice that we use boldface subscripts to separate between the numbers and their representations. Especially, $|M|$ stands for the *absolute value* of M whereas $|M_{\mathbf{d}}|$ stands for the length of the string representing M.

Equation (1) gives inequalities $M \geq d^{\ell-1}$ and

$$M \leq (d-1)(d^{\ell-1} + d^{\ell-2} + \ldots + d + 1) = d^{\ell} - 1 < d^{\ell}.$$

Combining these two estimates we see that $d^{\ell-1} \leq M < d^{\ell}$, or, equivalently, $\ell - 1 \leq \log_d M < \ell$, which is to say that $|M_{\mathbf{d}}| = \ell = \lfloor \log_d M \rfloor + 1$. In the case $d = 3$ we say that $M_{\mathbf{3}}$ is the *ternary* representation of M.

Let $q = \frac{a}{b}$ be a nonzero rational number such that $\gcd(a, b) = 1$. The *height* of q is defined as $H(q) = \max\{|a|, |b|\}$, and the *size* of q as $S(q) = |a_{\mathbf{d}}| + |b_{\mathbf{d}}|$, where d is some fixed base where both a and b are represented.

As defined above, the size of a rational number $q = \frac{a}{b}$ depends on the number system on which a and b are represented. On the other hand, since $\log_d M < |M_{\mathbf{d}}| \leq \log_d M + 1$ holds for any integer M, and because $\log_d M = (\ln d)^{-1} \ln M$, we see that the length of M is $\Theta(\ln M)$ in any representation system. Here and hereafter, we exclude the unary representations.

3 The algorithms

In this section we represent the algorithms for computing $|(2^n)_{\mathbf{3}}|$ and the first symbol of $(2^n)_{\mathbf{3}}$, and analyze their complexities. In the remaining sections we prove that the algorithms represented work correctly. We begin with some observations.

Without loss of generality we can assume that the input n is given in binary representation, so the *size* of input is $|n_{\mathbf{2}}|$, which, as seen in the previous section, is $\Theta(\ln n)$. We can therefore state our problem rigorously as follows: given an input $n_{\mathbf{2}} \in \{0,1\}^*$, compute $(2^n)_{\mathbf{3}} \in \{0,1,2\}^*$.

As seen in the previous sections, the size of the input is $\Theta(\ln n)$, whereas the output size is $\Theta(n)$, which is exponential in the input size. It follows that the problem is intractable by necessity.

On the other hand, computing $n_{\mathbf{2}} \mapsto |(2^n)_{\mathbf{3}}|$ is a very different problem. Now the output should be the ternary length of 2^n, or, from the algorithmic

n	$(2^n)_{\mathbf{3}}$	$\lvert(2^n)_{\mathbf{3}}\rvert$	$\lfloor 0.6n \rfloor + 1$
1	2	1	1
2	11	2	2
3	22	2	2
4	121	3	3
5	1012	4	4
6	2101	4	4
7	11202	5	5
8	100111	6	5(!)
9	200222	6	6
10	1101221	7	7
11	2210212	7	7
12	12121201	8	8
13	102020102	9	8(!)

Figure 1: Ternary representations of 2^n for $n = 1, 2, \ldots, 13$.

point of view, a *word* which represents the length. Again, without loss of generality, we can require the output in binary, which means that the output size would be

$$\begin{aligned} \lvert\lvert(2^n)_{\mathbf{3}}\rvert_{\mathbf{2}}\rvert &= \lfloor \log_2 \lvert(2^n)_{\mathbf{3}}\rvert \rfloor + 1 \leq \log_2(\lfloor n \log_3 2 \rfloor + 1) + 1 \\ &\leq \log_2(n \log_3 2 + n) + 1 = \log_2 n + \log_2(\log_3 2 + 1) + 1, \end{aligned}$$

in particular, it is in $O(\ln n)$.

Equation

$$\lvert(2^n)_{\mathbf{3}}\rvert = \lfloor n \log_3 2 \rfloor + 1 \tag{2}$$

gives a good starting point for computing the length, but the straightforward utilization of (2) contains at least two problematic features.

First, knowing n and $\log_3 2$ precisely enough allows us to compute their product, but it must be noted that we should be able to compute $\log_3 2 \approx 0.6$ at least up to precision $\frac{1}{n}$, since for larger imprecisions the outcome would be incorrect. This is illustrated in the rightmost column of Figure 1. This problem is easy to handle, and the beginning of the next section is devoted to this.

The second, and more severe problem is, that an approximation for $\log_3 2$, does not directly offer any tools to compute $\lfloor n \log_3 2 \rfloor$, no matter how precise the approximation is! To see this, let β_n, $n = 1, 2, 3, \ldots$ be a sequence of irrational numbers, and b_n, $n = 1, 2, 3, \ldots$, be a sequence of their very precise rational approximations, $\lvert b_n - \beta_n \rvert \ll 1$ for each n. Let us take some n, and assume, for instance, that $b_n < \beta_n$. If the interval (b_n, β_n) happens to contain an integer M, then $\lfloor b_n \rfloor = M - 1$, whereas the correct value $\lfloor \beta_n \rfloor$ is equal to M. In other words, if we do not have apriori knowledge on the distance between β_n and the nearest integer M, we cannot certainly find the value $\lfloor \beta_n \rfloor$ by using only an approximation b_n of β_n. In the next section, we use a deep result of Alan Baker to solve this problem.

When computing the most significant (leftmost) digit of $(2^n)_{\mathbf{3}}$, we note that an estimation similar to that one used in the previous section shows that if $(2^n)_{\mathbf{3}} \in 1\{0,1,2\}^*$, then

$$3^{\ell-1} \leq 2^n < 2 \cdot 3^{\ell-1},$$

whereas $(2^n)_{\mathbf{3}} \in 2\{0,1,2\}^*$ implies

$$2 \cdot 3^{\ell-1} \leq 2^n < 3^{\ell},$$

where $\ell = |(2^n)_{\mathbf{3}}|$. Thus, to recover the most significant digit of $(2^n)_{\mathbf{3}}$ is to decide whether or not the inequality

$$2 \cdot 3^{\ell-1} \leq 2^n \iff 3^{\ell-1} \leq 2^{n-1} \tag{3}$$

holds. Inequality (3) is equivalent to

$$\ell - 1 \leq (n-1)\log_3 2, \tag{4}$$

and (4) will be used for finding the first digit of $(2^n)_{\mathbf{3}}$.

Now we are ready for the algorithms. The constant C occurring in the algorithms will be explained in the next section.

Algorithm 1.
Input: A natural number n in binary representation.
Output: $|(2^n)_{\mathbf{3}}|$ in binary representation.

1. Compute a rational approximation q of $\log_3 2$ so precise that
$$|q - \log_3 2| \leq \frac{1}{\ln 3} \frac{1}{n^{C+1}}.$$
2. Compute qn.
3. Compute $\lfloor qn \rfloor$.
4. Output $\lfloor qn \rfloor + 1$.

In the next section we show that step 1 can be accomplished in time $O(\ln^3 n\,(\ln\ln n)^2)$, and, moreover, the approximating q has size $O(\ln n \ln\ln n)$. Especially, the numerator of $q = \frac{a}{b}$ has size $O(\ln n \ln\ln n)$, and hence the multiplication of q by n in step 2 can be done in time $O((\ln n \ln\ln n)^2)$, and the resulting number $qn = \frac{an}{b}$ has size $O(\ln n \ln\ln n + \ln n) = O(\ln n \ln\ln n)$. For the third step, an ordinary division of an by b is enough to reveal $\lfloor qn \rfloor$, and because of the numerator and the nominator sizes, it can be done in time $O((\ln n \ln\ln n)^2)$. As verified earlier, the outcoming number has size $O(\ln n)$, which implies that the last computation included in the fourth step can be performed in time $O(\ln n)$.

As a conclusion: Computation $n_\mathbf{2} \mapsto |(2^n)_\mathbf{3}|_\mathbf{2}$ can be performed in time $O(\ln^3 n(\ln\ln n)^2)$, or, to put it into other format, in time $O(|n_\mathbf{2}|^3 \ln^2 |n_\mathbf{2}|)$.

Algorithm 2.
Input: A natural number n in binary representation.
Output: The leftmost digit of string $(2^n)_\mathbf{3}$.

1. Compute $\ell = |(2^n)_\mathbf{3}|$ by using Algorithm 1.
2. Compute a rational approximation q of $\log_3 2$ so precise that
$$|q - \log_3 2| \le \frac{1}{\ln 3}\frac{1}{(n-1)^{C+1}}.$$
3. Compute numbers $\ell - 1$ and $(n-1)q$.
4. Decide, whether $\ell - 1 \le (n-1)q$.
5. If $\ell - 1 \le (n-1)q$, output 2, otherwise output 1.

The complexity analysis of Algorithm 2 is similar to that of the Algorithm 1, and the outcoming complexity is $O(|n_\mathbf{2}|^3 \ln^2 |n_\mathbf{2}|)$.

4 Validity of the algorithms

Approximating $\log_3 2$

It is a well-known fact that the series

$$\ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \dots$$

converges if $|x| < 1$. By substituting $x = -y$ we see that

$$\ln\frac{1}{1-y} = -\ln(1-y) = y + \frac{1}{2}y^2 + \frac{1}{3}y^3 + \frac{1}{4}y^4 + \dots,$$

and, a further substitution $y = \frac{\alpha-1}{\alpha}$ shows us that

$$\ln\alpha = \sum_{i=1}^{\infty}\frac{1}{i}\left(\frac{\alpha-1}{\alpha}\right)^i \tag{5}$$

converges for any fixed $\alpha > 1$. Hereafter we assume that $\alpha > 1$ is a fixed integer.

The remainder of (5) can be easily estimated as

$$\begin{aligned}
& \left| \sum_{i=M+1}^{\infty} \frac{1}{i} \left(\frac{\alpha-1}{\alpha} \right)^i \right| \\
< & \left(\frac{\alpha-1}{\alpha} \right)^{M+1} \sum_{i=0}^{\infty} \left(\frac{\alpha-1}{\alpha} \right)^i \\
= & \alpha \left(\frac{\alpha-1}{\alpha} \right)^{M+1}.
\end{aligned}$$

Thus, by choosing

$$M > 1 + \frac{\ln \frac{\alpha}{\epsilon}}{\ln \frac{\alpha}{\alpha-1}} \tag{6}$$

the remainder is less than ϵ.

In the case $\epsilon = \frac{1}{n^T}$ we see that in order to approximate $\ln \alpha$ within precision $\frac{1}{n^T}$, it is sufficient to take

$$M = 1 + \frac{T \ln n + \ln \alpha}{\ln \frac{\alpha}{\alpha-1}} = \Theta(T \ln n)$$

first summands of the series (5). Notice that all the summands in the series are positive, which implies that an approximation obtained by taking M first terms is always smaller than the actual value of the logarithm.

Let us choose $M = \Theta(T \ln n)$. Then, by writing

$$\sum_{i=1}^{M} \frac{1}{i} \left(\frac{\alpha-1}{\alpha} \right)^i = \frac{1}{M!\alpha^M} \sum_{i=1}^{M} \frac{M!}{i} \alpha^{M-i} (\alpha-1)^i \tag{7}$$

we see that the nominator of (7) is at most $M!\alpha^M$, and its length is of order

$$\begin{aligned}
\ln(M!\alpha^M) &= M \ln M + O(M) + M \ln \alpha = O(M \ln M) \\
&= O(T \ln n \ln(T \ln n)) = O(\ln n \ln \ln n).
\end{aligned}$$

An estimate for the numerator can be found in the similar way: First, a trivial estimate shows that the numerator is at most $M \cdot M!\alpha^M(\alpha-1)^M$, whose length is of order

$$\begin{aligned}
& \ln M + M \ln M + O(M) + M \ln \alpha + M \ln(\alpha - 1) = O(M \ln M) \\
= & O(\ln n \ln \ln n)
\end{aligned}$$

similarly as in the case of the nominator.

Number $M!$, or, to be precise, its representation, can be computed in time $O(M(\ln M!)^2) = O(M^3 \ln^2 M) = O(\ln^3 n (\ln \ln n)^2)$. On the other hand, number α^M can be found in time $O(M \cdot (M \ln \alpha)^2) = O(M^3) =$

$O(\ln^3 n)$, and the product $M!\alpha^M$ in time $O(\ln^2 M!) = O((\ln n \ln\ln n)^2)$. It follows that the nominator of (7) can be computed in time $O(\ln^3 n(\ln\ln n)^2)$. A similar estimate can be found also for the numerator.

As a conclusion, we get the following theorem:

Theorem 1. *Let $\alpha > 1$ and T be fixed natural numbers. It is possible to compute a rational number $q < \ln\alpha$ such that $\alpha - q < \frac{1}{n^T}$ in time $O(\ln^3 n(\ln\ln n)^2)$. Moreover, the size of q is $O(\ln n \ln\ln n)$.*

Now, a rational approximation for $\log_3 2 = \frac{\ln 2}{\ln 3}$ can be found by finding rational approximations r and s for both $\ln 2$ and $\ln 3$, respectively. Assume that $\ln 2 = r + \epsilon_1$ and $\ln 3 = s + \epsilon_2$, where $0 < \epsilon_1,\ \epsilon_2 \leq \epsilon$. If we further assume that s and r are so good approximations that $1 \leq s < \ln 3$ and $r > 0$, we can estimate their ratio as

$$\left|\frac{\ln 2}{\ln 3} - \frac{r}{s}\right| = \frac{|s\ln 2 - r\ln 3|}{s\ln 3} \leq \frac{|\epsilon_1 s - \epsilon_2 r|}{\ln 3}.$$

If $\epsilon_1 s - \epsilon_2 r \geq 0$, then $|\epsilon_1 s - \epsilon_2 r| = \epsilon_1 s - \epsilon_2 r < \epsilon_1 s < \epsilon \ln 3$, whereas $\epsilon_1 s - \epsilon_2 r < 0$ implies $|\epsilon_1 s - \epsilon_2 r| = \epsilon_2 r - \epsilon_1 s \leq \epsilon_2 r < \epsilon \ln 2$. In both cases,

$$\left|\frac{\ln 2}{\ln 3} - \frac{r}{s}\right| \leq \frac{|\epsilon_1 s - \epsilon_2 r|}{\ln 3} < \frac{\epsilon \ln 3}{\ln 3} = \epsilon.$$

The above inequality states that if r and s are approximations of $\ln 2$ and $\ln 3$ at most ϵ apart from the actual values, then their ratio $\frac{r}{s}$ is an approximation of $\log_3 2$ at most ϵ apart from the real value. Moreover, it is clear that $\frac{r}{s}$ has size $O(\ln n \ln\ln n)$, if both r and s have.

Baker's result

The extra information we use for computing $\ell = \lfloor n \log_3 2\rfloor + 1$ is provided in the following theorem, the proof can be found in [1].

Theorem 2 (A. Baker, 1966). *Let $\alpha_1, \ldots, \alpha_k$ be non-zero algebraic numbers with degrees at most d and heights at most A. Further, let $\beta_0, \ldots, \beta_k$ be algebraic numbers with degrees at most d and heights at most $B \geq 2$. Then for*

$$\Lambda = \beta_0 + \beta_1 \ln\alpha_1 + \ldots + \beta_k \ln\alpha_k$$

we have either $\Lambda = 0$ or $|\Lambda| > B^{-C}$, where C is an effectively computable number depending only on k, d, A, and the original determinations of the logarithms.

Remark 1. In the case when $\beta_0 = 0$ and $\beta_1, \ldots, \beta_k$ are rational integers, it has been shown that the theorem holds with $C = C'(\ln A)^k \ln(\ln A)^k$ and C' depends only on k and d [1].

Now we choose $k = 2$, $\beta_0 = 0$, $\beta_1 = m \in \mathbb{N}$, $\beta_2 = -n \in -\mathbb{N}$, $\alpha_1 = 3$, and $\alpha_2 = 2$. Then we can take $d = 1$, $A = 3$, and $B = \max\{|m|, |n|\}$. It is easy to show that $m \ln 3 - n \ln 2 \neq 0$, so we have the following theorem as a special case of Theorem 2.

Theorem 3. *There exists a constant C such that*

$$|m \ln 3 - n \ln 2| > \frac{1}{n^C},$$

whenever $m, n \in \mathbb{N}$ and $m \leq n$.

Remark 2. In fact, the previous theorem would already follow from a result of Gelfond, see [2]. It can be also shown that the above theorem holds with $C = 13.3$ [3].

The above theorem directly implies that

$$|m - n \log_3 2| > \frac{1}{\ln 3} \frac{1}{n^C} \tag{8}$$

for any natural numbers $m < n$. Important points here are that C is independent of n and m, and can be computed.

Theorem 4. *Let C be the constant mentioned in (8). If $q \in \mathbb{Q}$ is an approximation of $\log_3 2$ so precise that $|q - \log_3 2| \leq \frac{1}{\ln 3} \frac{1}{n^{C+1}}$, then $\lfloor n \log_3 2 \rfloor = \lfloor nq \rfloor$.*

Proof. Let $M = \lfloor nq \rfloor$. The assumption on q verifies that $M + 1 \leq n$, so the premise of Theorem 3 is satisfied. Now $M \leq nq < M + 1$, and we must show that also $M \leq n \log_3 2 < M + 1$. For that purpose, we first assume, for the sake of contradiction, that $n \log_3 2 < M$. But then we would have

$$|M - n \log_3 2| = M - n \log_3 2 \leq nq - n \log_3 2,$$

which is at most $\frac{1}{\ln 3} \frac{1}{n^C}$ by the assumption. In virtue of inequality (8) this is a contradiction, and hence $M \leq n \log_3 2$. On the other hand, if $n \log_3 2 \geq M + 1$, then

$$|M + 1 - n \log_3 2| = n \log_3 2 - (M + 1) < n \log_3 2 - nq \leq \frac{1}{\ln 3} \frac{1}{n^C},$$

which is absurd by inequality (8). Thus $M \leq n \log_3 2 < M + 1$, which shows that $\lfloor n \log_3 2 \rfloor = M = \lfloor qn \rfloor$. $\square$

Theorem 4 directly implies the correctness of Algorithm 1. We emphasize again that the required approximation q can be computed in time $O(|\mathbf{n_2}|^3 \ln^2 |\mathbf{n_2}|)$. For Algorithm 2, we state the following theorem.

Theorem 5. *Let $q \in \mathbb{Q}$ be an approximation of $\log_3 2$ so precise that*

$$|q - \log_3 2| \leq \frac{1}{\ln 3} \frac{1}{(n-1)^{C+1}}, \tag{9}$$

Then $\ell - 1 \leq (n-1)q$ if and only if $\ell - 1 \leq (n-1)\log_3 2$.

Proof. Assume first that $\ell - 1 \leq (n-1)\log_3 2$, but $\ell - 1 > (n-1)q$. Then

$$\begin{aligned} & |(n-1)\log_3 2 - (\ell - 1)| = (n-1)\log_3 2 - (\ell-1) \\ < \quad & (n-1)\log_3 2 - (n-1)q. \end{aligned}$$

By (9), the latest expression is at most $\frac{1}{\ln 3}\frac{1}{(n-1)^C}$, which contradicts inequality (8). Similarly, assuming $\ell - 1 \leq (n-1)q$ and $\ell - 1 > (n-1)\log_3 2$ leads into contradiction. $\square$

The above theorem verifies the correctness of Algorithm 2.

5 Conclusion

We have shown how to compute $|(2^n)_{\mathbf{3}}|$ and the first symbol of $(2^n)_{\mathbf{3}}$ in polynomial time with respect to $|n_{\mathbf{2}}|$. We believe that our approach, together with more general results of [1], allows also to compute k first symbols of $(2^n)_{\mathbf{3}}$ for any fixed k, or more generally, k first symbols of $(p^n)_{\mathbf{q}}$ for $p, q \geq 2$ and $p \neq q$.

On the other hand we do not see how to compute the middle symbol of $(2^n)_{\mathbf{3}}$ in polynomial time. This motivates the following problem.

Problem. Does there exist a polynomial time algorithm computing, for a given n and $i \leq \lfloor n \log_3 2 \rfloor + 1$, the ith symbol of string $(2^n)_{\mathbf{3}}$ in polynomial time with respect to $|n_{\mathbf{2}}| = \Theta(\ln n)$?

Other problems are to consider our questions for some other exponentially growing sequences of numbers, such as the Fibonacci sequence.

Acknowledgement

We wish to thank Tapani Matala-aho for providing the reference to the work of Georges Rhin [3].

References

[1] A. Baker: *Transcendental Number Theory.* Cambridge University Press (1975).

[2] A. Gelfond: *Sur les approximations des nombres transcendants par des nombres algébriques.* Dokl. Acad. Sci. URSS 2, 177–182, (in russian and french) (1935).

[3] G. Rhin: *Approximants de Padé et mesures effectives d'irrationalité.* Séminaire de Théorie des Nombres, Paris 1985–86, 155–164, Progr. Math., 71, Birkhäuser Boston, Boston, MA (1987).

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.fi

University of Turku
- **Department of Mathematical Sciences**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**