Universität Bielefeld

Technische Fakultät
Abteilung Informationstechnik
Forschungsberichte

The Money Changing Problem revisited: Computing the Frobenius number in time $O(k\,a_1)$

Sebastian Böcker Zsuzsanna Lipták

Report 2004-02

Impressum: Herausgeber:
Robert Giegerich, Ralf Hofestädt, Franz Kummert,
Peter Ladkin, Ralf Möller, Helge Ritter,
Gerhard Sagerer, Jens Stoye, Ipke Wachsmuth

Technische Fakultät der Universität Bielefeld,
Abteilung Informationstechnik, Postfach 10 01 31,
33501 Bielefeld, Germany

ISSN 0946-7831

THE MONEY CHANGING PROBLEM REVISITED: COMPUTING THE FROBENIUS NUMBER IN TIME $O(k\,a_1)$

SEBASTIAN BÖCKER AND ZSUZSANNA LIPTÁK

ABSTRACT. The Money Changing Problem is as follows: Let $a_1 < a_2 < \cdots < a_k$ be fixed positive integers with $\gcd(a_1, \ldots, a_k) = 1$. Given some integer n, are there non-negative integers $x_1, \ldots, x_k$ such that $\sum_i a_i x_i = n$? The *Frobenius number* $g(a_1, \ldots, a_k)$ is the largest integer n such that the above problem has no decomposition $x_1, \ldots, x_k$.

There exist algorithms that, for fixed k, compute the Frobenius number in time polynomial in $\log a_i$. For variable k, one can compute a residue table of a_1 words which, in turn, allows to determine the Frobenius number. The best known algorithm for computing the residue table has runtime $O(k\,a_1 \log a_1)$ using binary heaps, and $O(a_1(k + \log a_1))$ using Fibonacci heaps. In both cases, $O(a_1)$ extra memory in addition to the residue table is needed. Here, we present an intriguingly simple algorithm to compute the residue table in time $O(k\,a_1)$ and extra memory $O(1)$. In addition to computing the Frobenius number, we can use the residue table to solve the given instance of the Money Changing Problem in constant time, for any n.

1. INTRODUCTION

In the classical Money Changing Problem (MCP), we are given coins of k different values $a_1 < a_2 < \cdots < a_k$ with $\gcd(a_1, \ldots, a_k) = 1$. We want to know what change $n = \sum_i a_i x_i$ we can generate from these coins for non-negative integers x_i, assuming that we have an infinite supply of coins. Then, there exists an integer $g(a_1, \ldots, a_k)$ called the *Frobenius number* of $a_1, \ldots, a_k$, such that $g(a_1, \ldots, a_k)$ does not allow a decomposition of the above type, but all integers $n > g(a_1, \ldots, a_k)$ do.[1] Sylvester and Curran Sharp [9] show that for $k = 2$ we have $g(a_1, a_2) = a_1 a_2 - a_1 - a_2$. Kannan [4] developed an algorithm that for *fixed* k, computes the Frobenius number in time polynomial in $\log a_k$. For variable k, the runtime of this algorithm has a double exponential dependence on k, the dimension of the problem. Computing the Frobenius number is NP-hard [7] (and so is MCP [5]), so we cannot hope to find an algorithm polynomial in k and $\log a_i$ simultaneously.

If k is variable, the fastest algorithm to compute $g(a_1, \ldots, a_k)$ dates back to 1979: Nijenhuis' algorithm [6] has runtime $O(k\,a_1 \log a_1)$ using binary heaps, and $O(a_1(k+\log a_1))$ using Fibonacci heaps. It is based on an observation of Brauer and Shockley [2] to find integers n_p for $p = 0, \ldots, a_1 - 1$ such that for all integers $n \equiv p \mod a_1$, n_p is the smallest number that has a decomposition $n_p = \sum_i a_i x_i$. Note that the n_p's can be used to answer subsequent MCP questions for variable n in constant time. Nijenhuis' algorithm requires $O(a_1)$ extra memory in addition to the *residue table* n_p for $p = 0, \ldots, a_1 - 1$. Here, we present a simple algorithm to compute the values n_p and, hence, to find $g(a_1, \ldots, a_k)$ in time $O(k\,a_1)$ and extra memory $O(1)$.

2. RESIDUE CLASSES AND THE FROBENIUS NUMBER

For integers a, b, let $a \bmod b$ denote the unique integer in $0, \ldots, b-1$ such that $(a \bmod b) \equiv a \mod b$ holds.

Let $a_1 < \cdots < a_k$ with $\gcd(a_1, \ldots, a_k) = 1$ be an instance of the Money Changing Problem. We denote by n_p the smallest integer with $n_p \equiv p \mod a_1$ that can be decomposed into a non-negative integer combination of $a_1, \ldots, a_k$. The n_p are well-defined: If n has a decomposition, so has $n + a_1$, and $n \equiv n + a_1 \mod a_1$. Clearly, $\sum_i a_i x_i = n_p$ implies $x_1 = 0$ because otherwise, $n_p - a_1$ has a decomposition, too. If the n_p are known, then we can test in constant time whether some number n can be decomposed: Set $p = n \bmod a_1$, then n can be decomposed if and only if $n \geq n_p$.

Date: June 21, 2004; Technical Report no. 2004-2, University of Bielefeld, Technical Faculty.

[1] There has been considerable work on bounds for Frobenius numbers (see [8] for a survey) but here we concentrate on *exact* computations.

Given the values n_p for $p = 0, \ldots, a_1 - 1$ we can compute the Frobenius number $g(a_1, \ldots, a_k)$ and the number of *omitted* values ω that cannot be decomposed using $a_1, \ldots, a_k$ [2]:

$$g := g(a_1, \ldots, a_k) = \max_p\{n_p\} - a_1 \quad \text{and} \quad \omega = \sum_p \left\lfloor \frac{n_p}{a_1} \right\rfloor = \frac{1}{a_1} \sum_p n_p - \frac{a_1 - 1}{2}.$$

To compute the values n_p for $p = 0, \ldots, a_1 - 1$, Nijenhuis [6] gave an algorithm with runtime $O(k\, a_1 \log a_1)$, where the $\log a_1$ factor is due to a binary heap structure that must be updated in every step. This algorithm improved on older algorithms with worse runtime bounds [3, 10], see [8] for a survey. We can modify Nijenhuis' algorithm by using a Fibonacci heap instead of a regular heap, thereby reaching a $O(a_1(k + \log a_1))$ runtime bound. It should be understood that the constant factor overhead (runtime and memory) is much higher for a Fibonacci heap.

3. The Round Robin Algorithm

We compute the values n_p iteratively for the sub-problems "Find n_p for the instance $a_1, \ldots, a_i$" for $i = 1, \ldots, k$. For $i = k = 1$ we start with $n_0 = 0$ and $n_p = \infty$ for $p = 1, \ldots, a_1 - 1$.

Suppose we know the correct values n'_p for the sub-problem $a_1, \ldots, a_{k-1}$ and want to calculate those of the original problem $a_1, \ldots, a_k$. We first concentrate on the simple case that $\gcd(a_1, a_k) = 1$. We initialize $n_p \leftarrow n'_p$ for all $p = 0, \ldots, a_1 - 1$ and $n \leftarrow n_0 = 0$. In every step of the algorithm, set $n \leftarrow n + a_k$ and $p \leftarrow n \bmod a_1$. Let $n \leftarrow \min\{n, n_p\}$ and $n_p \leftarrow n$. We repeat this loop until n equals 0.

In case all $a_2, \ldots, a_k$ are coprime to a_1, this short algorithm is already sufficient to find the correct values n_p. We have displayed a small example in Fig. 1, where every column corresponds to one step of the algorithm as described in the previous paragraph. Focus on the column $a_3 = 9$, we start with $n = 0$. In the first step, we have $n \leftarrow 9$ and $p = 4$. Since $n < n_4 = 24$ we update $n_4 \leftarrow 9$. Second, we have $n \leftarrow 9 + 9 = 18$ and $p = 3$. In view of $n > n_3 = 8$ we set $n \leftarrow 8$. Third, we have $n \leftarrow 8 + 9 = 17$ and $p = 2$. Since $n < n_2 = 32$ we update $n_2 \leftarrow 17$. Fourth, we have $n \leftarrow 17 + 9 = 26$ and $p = 1$. In view of $n > n_1 = 16$ we set $n \leftarrow 16$. Finally, we return to $p = 0$ via $n \leftarrow 16 + 9 = 25$. The Frobenius number for this example is $g(5, 8, 9, 12) = 16 - 5 = 11$.

p	$a_1 = 5$	$a_2 = 8$	$a_3 = 9$	$a_4 = 12$
0	0	0	0	0
1	∞	16	16	16
2	∞	32	17	12
3	∞	8	8	8
4	∞	24	9	9

Figure 1. Table n_p for the MCP instance $5, 8, 9, 12$.

It is straighforward how to generalize the algorithm for $d := \gcd(a_1, a_i) > 1$: In this case, we do the updating independently for every residue $r = 0, \ldots, d - 1$. Only those n_p for $p \in \{0, \ldots, a_1 - 1\}$ are updated that satisfy $p \equiv r \mod d$. To guarantee that the round-robin loop completes updating after a_1/d steps, we have to start the loop from a minimal n_p with $p \equiv r \mod d$. For $r = 0$ we know that $n_0 = 0$ is the unique minimum, while for $r \neq 0$ we search for the minimum first.

Round Robin Algorithm

```
1  initialize n_0 = 0 and n_p = ∞ for p = 1, ..., a_1 − 1
2  for i = 2, ..., k do
3      d = gcd(a_1, a_i);
4      for r = 0, ..., d − 1 do
5          Find n = min{n_q : q = r, r + d, r + 2d, ..., r + (a_1 − d)};
6          If n < ∞ then repeat a_1/d times
7              n ← n + a_i; p = n mod a_1;
8              n ← min{n, n_p}; n_p ← n;
9          done;
10     done;
11 done.
```

The inner loop (lines 6–9) will be executed only if the minimum $\min\{n_q\}$ is finite, in case some residue class cannot be decomposed by $a_1, \ldots, a_i$ because of $\gcd(a_1, \ldots, a_i) > 1$.

Lemma 1. *Suppose that n'_p for $p = 0, \ldots, a_1 - 1$ are the residues for the MCP instance $a_1, \ldots, a_{k-1}$. Initialize $n_p \leftarrow n'_p$ for $p = 0, \ldots, a_1 - 1$. Then, after one step of the outer loop (for $i = k$) of the Round Robin Algorithm, the values n_p for $p = 0, \ldots, a_1 - 1$ are the residues for the MCP instance $a_1, \ldots, a_k$.*

Since for $k = 1$, $n_0 = 0$ and $n_p = \infty$ for $p \neq 0$ are the correct residues of MCP with one coin, we can use induction to show the correctness of the algorithm. To prove the lemma, we first note that for all $p = 0, \ldots, a_1 - 1$,

$$n_p \leq n'_p \quad \text{and} \quad n_p \leq n_q + a_k \text{ for } q = (p - a_k) \bmod a_1$$

after termination. Assume that for some n, there exists a decomposition $n = \sum_{i=1}^{k} a_i x_i$. We have to show $n \geq n_p$ for $p = n \bmod a_1$. Then, $\sum_{i=1}^{k-1} a_i x_i = n - a_k x_k$ is a decomposition of the MCP instance $a_1, \ldots, a_{k-1}$ and for $q = (n - a_k x_k) \bmod a_1$ we have $n - a_k x_k \geq n'_q$. We conclude

$$n_p \leq n_q + a_k x_k \leq n'_q + a_k x_k \leq n.$$

By an analogous argument, we infer $n_p = n$ for *minimal* such n. One can easily show that $n_p = \infty$ if and only if no n with $n \equiv p \mod a_1$ has a decomposition with respect to the MCP instance $a_1, \ldots, a_k$.

As time and space complexity of the algorithm are quite obvious, we reach:

Theorem 1. *The Round Robin Algorithm computes the residue table of an instance $a_1, \ldots, a_k$ of the Money Changing Problem, in runtime $O(k\, a_1)$ and extra memory $O(1)$.*

To obtain a decomposition of any n in k steps, we save for every $p = 0, \ldots, a_1 - 1$ an index i such that a decomposition $x_1, \ldots, x_k$ of n_p has $x_i > 0$, and we also save the maximal x_i for any such decomposition. This can be easily incorporated into the algorithm retaining identical time complexity, and requires $2a_1$ additional words of memory.

We can improve the algorithm in the following two ways: First, we do not have to explicitly compute the greatest common divisor $\gcd(a_1, a_i)$. Instead, we do the first round-robin loop (lines 6–9) for $r = 0$ with $n = n_0 = p = 0$ until we reach $n = p = 0$ again. We count the number of steps t to this point. Then, $d = \gcd(a_1, a_i) = \frac{a_1}{t}$ and for $d > 1$, we do the remaining round-robin loops $r = 1, \ldots, d - 1$. Second, for $r > 0$ we do not have to search for the minimum in $\{n_q : q = r, r+d, r+2d, \ldots, r+(a_1-d)\}$. Instead, we start with $n = n_r$ and do exactly t steps of the round-robin loop. Here, $n_r = \infty$ may hold, so we initialize $p = r$ (line 5) and update $p \leftarrow (p + a_i) \bmod a_1$ separately in line 7. Afterwards, we continue with this loop until we first encounter some $n_p \leq n$ in line 8, and stop there.

It should be noted that we can modify the above algorithm to find *all* decompositions of some input n: To this end, we generate a data structure with runtime and space $O(\sum_i a_i)$. If we denote the number of decompositions of n by $\gamma(n)$, then this data structure allows us to generate all decompositions in time linear in $\gamma(n)$ and independent of n. The details will be published elsewhere.

4. Computational results

We tested our algorithm on some instances of the Money Changing Problem that are known to be "hard": Twenty-five such examples with $5 \leq k \leq 10$ and $3719 \leq a_1 \leq 48709$ were taken from [1] and presented to our algorithm. The runtime (900 MHz UltraSparc III processor, C++) for every instance was below 16 ms, and compared well to those of [1] as well as standard linear programming based branch-and-bound search.[2]

In addition, we generated 100 000 random instances of MCP, with $3 \leq k \leq 50$ and $10^3 \leq a_i \leq 10^7$. We have plotted the runtime of our algorithm against $k\, a_1$ in Fig. 2. As one can see, the runtime of the algorithm is mostly independent of the structure of the underlying instance. The processor cache, on the contrary, is responsible for major runtime differences. So, we split the instances into two groups, depending on a_1 and, thus, the size of the residue table: The left plot contains all instances with $a_1 \leq 10^6$; these appear to fit into the processor cache. The right plot, on the other hand, contains those instances with $a_1 > 10^6$; here, the residue table has to be stored in main memory. In the left plot, the 25 problems from the previous paragraph are indicated by circles.

[2]More precisely, Aardal and Lenstra [1] do not compute the Frobenius number g, but only verify that g cannot be decomposed.

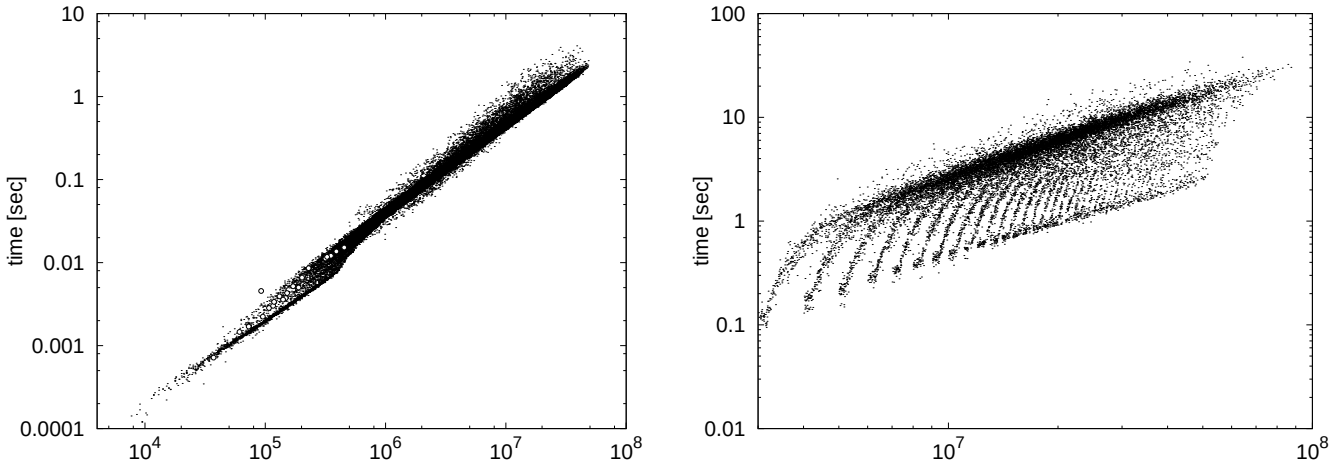

FIGURE 2. Runtime vs. $k\,a_1$ on a log-log scale, for $a_1 \leq 10^6$ (left) and $a_1 > 10^6$ (right).

ACKNOWLEDGMENTS

Sebastian Böcker and Zsuzsanna Lipták are currently supported by "Deutsche Forschungsgemeinschaft" (BO 1910/1-1 and 1-2) within the Computer Science Action Program. Implementation and simulations by Henner Sudek. We thank Stan Wagon for helpful discussions.

REFERENCES

[1] K. Aardal and A. K. Lenstra. Hard equality constrained integer knapsacks. *Lect. Notes Comput. Sc.*, 2337:350–366, 2002.
[2] A. Brauer and J. E. Shockley. On a problem of Frobenius. *J. Reine Angew. Math.*, 211:215–220, 1962.
[3] B. R. Heap and M. S. Lynn. A graph-theoretic algorithm for the solution of a linear diophantine problem of Frobenius. *Numer. Math.*, 6:346–354, 1964.
[4] R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12:161–177, 1991.
[5] G. S. Lueker. Two NP-complete problems in nonnegative integer programming. Technical Report TR-178, Department of Electrical Engineering, Princeton University, March 1975.
[6] A. Nijenhuis. A minimal-path algorithm for the "money changing problem". *Amer. Math. Monthly*, 86:832–835, 1979. Correction in *Amer. Math. Monthly* **87** (1980) 377.
[7] J. L. Ramírez-Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16(1):143–147, 1996.
[8] J. L. Ramírez-Alfonsín. *The Diophantine Frobenius Problem*. Oxford University Press, 2004. To appear.
[9] J. J. Sylvester and W. J. Curran Sharp. Problem 7382. *Mathematical Questions and Solutions from the Educational Times*, 41:171–178, 1884.
[10] H. S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *Amer. Math. Monthly*, 85:562–565, 1978.

AG GENOMINFORMATIK, TECHNISCHE FAKULTÄT, UNIVERSITÄT BIELEFELD, PF 100 131, 33501 BIELEFELD, GERMANY
E-mail address: {boecker,zsuzsa}@CeBiTec.uni-bielefeld.de

Bisher erschienene Reports an der Technischen Fakultät
Stand: 2004-06-15

94-01 Modular Properties of Composable Term Rewriting Systems
(Enno Ohlebusch)

94-02 Analysis and Applications of the Direct Cascade Architecture
(Enno Littmann, Helge Ritter)

94-03 From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction
(Robert Giegerich, Stefan Kurtz)

94-04 Die Verwendung unscharfer Maße zur Korrespondenzanalyse in Stereo Farbbildern
(André Wolfram, Alois Knoll)

94-05 Searching Correspondences in Colour Stereo Images – Recent Results Using the Fuzzy Integral
(André Wolfram, Alois Knoll)

94-06 A Basic Semantics for Computer Arithmetic
(Markus Freericks, A. Fauth, Alois Knoll)

94-07 Reverse Restructuring: Another Method of Solving Algebraic Equations
(Bernd Bütow, Stephan Thesing)

95-01 PaNaMa User Manual V1.3
(Bernd Bütow, Stephan Thesing)

95-02 Computer Based Training-Software: ein interaktiver Sequenzierkurs
(Frank Meier, Garrit Skrock, Robert Giegerich)

95-03 Fundamental Algorithms for a Declarative Pattern Matching System
(Stefan Kurtz)

95-04 On the Equivalence of E-Pattern Languages
(Enno Ohlebusch, Esko Ukkonen)

96-01 Static and Dynamic Filtering Methods for Approximate String Matching
(Robert Giegerich, Frank Hischke, Stefan Kurtz, Enno Ohlebusch)

96-02 Instructing Cooperating Assembly Robots through Situated Dialogues in Natural Language
(Alois Knoll, Bernd Hildebrand, Jianwei Zhang)

96-03 Correctness in System Engineering
(Peter Ladkin)

96-04 An Algebraic Approach to General Boolean Constraint Problems
(Hans-Werner Güsgen, Peter Ladkin)

96-05 Future University Computing Resources
(Peter Ladkin)

96-06 Lazy Cache Implements Complete Cache
(Peter Ladkin)

96-07 Formal but Lively Buffers in TLA+
(Peter Ladkin)

96-08 The X-31 and A320 Warsaw Crashes: Whodunnit?
(Peter Ladkin)

96-09 Reasons and Causes
(Peter Ladkin)

96-10 Comments on Confusing Conversation at Cali
(Dafydd Gibbon, Peter Ladkin)

96-11 On Needing Models
(Peter Ladkin)

96-12 Formalism Helps in Describing Accidents
(Peter Ladkin)

96-13 Explaining Failure with Tense Logic
(Peter Ladkin)

96-14 Some Dubious Theses in the Tense Logic of Accidents
(Peter Ladkin)

96-15 A Note on a Note on a Lemma of Ladkin
(Peter Ladkin)

96-16 News and Comment on the AeroPeru B757 Accident
(Peter Ladkin)

97-01 Analysing the Cali Accident With a WB-Graph
(Peter Ladkin)

97-02 Divide-and-Conquer Multiple Sequence Alignment
(Jens Stoye)

97-03 A System for the Content-Based Retrieval of Textual and Non-Textual Documents Based on Natural Language Queries
(Alois Knoll, Ingo Glöckner, Hermann Helbig, Sven Hartrumpf)

97-04 Rose: Generating Sequence Families
(Jens Stoye, Dirk Evers, Folker Meyer)

97-05 Fuzzy Quantifiers for Processing Natural Language Queries in Content-Based Multimedia Retrieval Systems
(Ingo Glöckner, Alois Knoll)

97-06 DFS – An Axiomatic Approach to Fuzzy Quantification
(Ingo Glöckner)

98-01 Kognitive Aspekte bei der Realisierung eines robusten Robotersystems für Konstruktionsaufgaben
(Alois Knoll, Bernd Hildebrandt)

98-02 A Declarative Approach to the Development of Dynamic Programming Algorithms, applied to RNA Folding
(Robert Giegerich)

98-03 Reducing the Space Requirement of Suffix Trees
(Stefan Kurtz)

99-01 Entscheidungskalküle
(Axel Saalbach, Christian Lange, Sascha Wendt, Mathias Katzer, Guillaume Dubois, Michael Höhl, Oliver Kuhn, Sven Wachsmuth, Gerhard Sagerer)

99-02 Transforming Conditional Rewrite Systems with Extra Variables into Unconditional Systems
(Enno Ohlebusch)

99-03 A Framework for Evaluating Approaches to Fuzzy Quantification
(Ingo Glöckner)

99-04 Towards Evaluation of Docking Hypotheses using elastic Matching
(Steffen Neumann, Stefan Posch, Gerhard Sagerer)

99-05 A Systematic Approach to Dynamic Programming in Bioinformatics. Part 1 and 2: Sequence Comparison and RNA Folding
(Robert Giegerich)

99-06 Autonomie für situierte Robotersysteme – Stand und Entwicklungslinien
(Alois Knoll)

2000-01 Advances in DFS Theory
(Ingo Glöckner)

2000-02 A Broad Class of DFS Models
(Ingo Glöckner)

2000-03 An Axiomatic Theory of Fuzzy Quantifiers in Natural Languages
(Ingo Glöckner)

2000-04 Affix Trees
(Jens Stoye)

2000-05 Computergestützte Auswertung von Spektren organischer Verbindungen
(Annika Büscher, Michaela Hohenner, Sascha Wendt, Markus Wiesecke, Frank Zöllner, Arne Wegener, Frank Bettenworth, Thorsten Twellmann, Jan Kleinlützum, Mathias Katzer, Sven Wachsmuth, Gerhard Sagerer)

2000-06 The Syntax and Semantics of a Language for Describing Complex Patterns in Biological Sequences
(Dirk Strothmann, Stefan Kurtz, Stefan Gräf, Gerhard Steger)

2000-07 Systematic Dynamic Programming in Bioinformatics (ISMB 2000 Tutorial Notes)
(Dirk J. Evers, Robert Giegerich)

2000-08 Difficulties when Aligning Structure Based RNAs with the Standard Edit Distance Method
(Christian Büschking)

2001-01 Standard Models of Fuzzy Quantification
(Ingo Glöckner)

2001-02 Causal System Analysis
(Peter B. Ladkin)

2001-03 A Rotamer Library for Protein-Protein Docking Using Energy Calculations and Statistics
(Kerstin Koch, Frank Zöllner, Gerhard Sagerer)

2001-04 Eine asynchrone Implementierung eines Microprozessors auf einem FPGA
(Marco Balke, Thomas Dettbarn, Robert Homann, Sebastian Jaenicke, Tim Köhler, Henning Mersch, Holger Weiss)

2001-05 Hierarchical Termination Revisited
(Enno Ohlebusch)

2002-01 Persistent Objects with O2DBI
(Jörn Clausen)

2002-02 Simulation von Phasenübergängen in Proteinmonoschichten
(Johanna Alichniewicz, Gabriele Holzschneider, Morris Michael, Ulf Schiller, Jan Stallkamp)

2002-03 Lecture Notes on Algebraic Dynamic Programming 2002
(Robert Giegerich)

2002-04 Side chain flexibility for 1:n protein-protein docking
(Kerstin Koch, Steffen Neumann, Frank Zöllner, Gerhard Sagerer)

2002-05 ElMaR: A Protein Docking System using Flexibility Information
(Frank Zöllner, Steffen Neumann, Kerstin Koch, Franz Kummert, Gerhard Sagerer)

2002-06 Calculating Residue Flexibility Information from Statistics and Energy based Prediction
(Frank Zöllner, Steffen Neumann, Kerstin Koch, Franz Kummert, Gerhard Sagerer)

2002-07 Fundamentals of Fuzzy Quantification: Plausible Models, Constructive Principles, and Efficient Implementation
(Ingo Glöckner)

2002-08 Branching of Fuzzy Quantifiers and Multiple Variable Binding: An Extension of DFS Theory
(Ingo Glöckner)

2003-01 On the Similarity of Sets of Permutations and its Applications to Genome Comparison
(Anne Bergeron, Jens Stoye)

2003-02 SNP and mutation discovery using base-specific cleavage and MALDI-TOF mass spectrometry
(Sebastian Böcker)

2003-03 From RNA Folding to Thermodynamic Matching, including Pseudoknots
(Robert Giegerich, Jens Reeder)

2003-04 Sequencing from compomers: Using mass spectrometry for DNA de-novo sequencing of 200+ nt
(Sebastian Böcker)

2003-05 Systematic Investigation of Jumping Alignments
(Constantin Bannert)

2003-06 Suffix Tree Construction and Storage with Limited Main Memory
(Klaus-Bernd Schürmann, Jens Stoye)

2003-07 Sequencing from compomers in thepresence of false negative peaks
(Sebastian Böcker)

2003-08 Genalyzer: An Interactive Visualisation Tool for Large-Scale Sequence Matching – Biological Applications and User Manual
(Jomuna V. Choudhuri, Chris Schleiermacher)

2004-01 Sequencing From Compomers is NP-hard
(Sebastian Böcker)