Evolutionary Construction of de Bruijn Sequences *

Meltem Sönmez Turan

National Institute of Standards and Technology,
Computer Security Division

Abstract. A binary de Bruijn sequence of order n is a cyclic sequence of period 2^n, in which each n-bit pattern appears exactly once. These sequences are commonly used in random number generation and symmetric key cryptography particularly in stream cipher design, mainly due to their good statistical properties. Constructing de Bruijn sequences is of interest and well studied in the literature. In this study, we propose a new randomized construction method based on genetic algorithms. The method models de Bruijn sequences as a special type of traveling salesman tours (TSP) and tries to find optimal solutions. We present some experimental results for $n \leq 14$.
Keywords: De Bruijn sequences, Genetic algorithms, Traveling salesman problem

1 Introduction

A binary de Bruijn sequence of order n is a cyclic sequence of period 2^n, in which each n-bit pattern appears exactly once. De Bruijn sequences are balanced, i.e., have same number of 1's and 0's, and have good randomness properties.

De Bruijn sequences are useful in random number generation, which is essential for secure communication especially for generation of cryptographic keys, nonces and salts. In stream cipher designs, feedback shift registers (FSRs) that can generate de Bruijn sequences are commonly used. All three hardware-oriented Ecrypt eStream competition [1] finalists, namely Trivium [4], Grain [13] and Mickey [3] use these registers as their main building block. However, due to the large size of these registers, it is not known whether the registers are capable of generating de Bruijn sequences.

One simple application of De Bruijn sequences is to attack digital lock systems that do not require using the *enter* key after each trial [7]. For example, a lock with four-digit combinations has 10,000 possible keys and the attacker is expected to type 20,000 digits to open the lock (40,000 digits for the worst case). Using de Bruijn sequences, the attacker can reduce the number of digits he is expected to type to 5,000.

*Previous version of this paper was published in AISec11, October 21, 2011, Chicago, Illinois, USA

De Bruijn sequences of order n exist for $n \geq 2$ and for a given n, the number of de Bruijn sequences is $2^{2^{n-1}-n}$ [11]. Efficiently generating de Bruijn sequences with large $n(\geq 64)$ has significance in stream cipher design, especially to have more provable security properties. For example, in the alternating step generator which is a generic stream cipher design approach, the security proofs of the system regarding period and linear complexity is done by the assumption that one of the registers generates a de Bruijn sequence [16].

There are various methods to construct de Bruijn sequences [7, 21]. Some of the construction methods start with an n-bit pattern and append a new bit to the sequence based on a pre-determined criteria, whereas some methods are recursive and use lower-order de Bruijn sequences as input. These construction methods are capable of generating a subset or all of the possible sequences for a given order n, with varying time and memory complexity. It is of interest to find an efficient construction method with a range that is not limited to a small subset of all possible sequences.

Genetic Algorithms, developed by Holland [14], are a part of evolutionary computation which is a subfield of artificial intelligence. These heuristic search algorithms are based on the survival of the fittest and natural selection concept of natural genetics. They simulate natural evolution with the goal of finding the best solution to problems having a large and non-linear search space.

In this study, we propose a new method to construct de Bruijn sequences using genetic algorithms. First, we define a special type of the traveling salesman problem (TSP), denoted as TSP_n^* with 2^n nodes and a predefined distance matrix. Then, we use our genetic algorithm to find an optimal tour for TSP_n^* and convert the output tour to a de Bruijn sequence.

The outline of the paper is as follows. In Sect. 2, literature surveys on de Bruijn sequence construction methods and genetic algorithms are provided. In Sect. 3, the definition and some basic properties of TSP_n^* are given. In Sect. 4, the details of the genetic algorithm are explained. The experimental results are summarized in Sect. 5. Finally, the results are discussed in the last section.

2 Preliminaries

The preliminaries part of this study consists of two main parts; constructions of de Bruijn sequences, and genetic algorithms especially focusing on their application to TSP.

2.1 Constructions of de Bruijn Sequences

Simplest construction method is the Prefer-one method which starts with n zeros and adds the bit 1 to the sequence whenever possible. For $n = 4$, the prefer-one method generates the following sequence;

$$0000111101100101.$$

Prefer-same [7] and prefer-opposite [2] methods are similar to the prefer-one method using different bit insertion criteria. These methods are deterministic and given the same initial state, they generate only one de Bruijn sequence.

De Bruijn sequences can also be generated using feedback shift registers (FSRs). A *FSR* is a device that shifts its contents into adjacent positions within the register and fills the position on the other end with a new value generated by the *feedback function*. A FSR is uniquely determined by its length n and feedback function f. The output sequence $\mathbf{S}= \{s_0, s_1, s_2, \ldots\}$ of a FSR satisfy the following recursion

$$s_{n+i} = f(s_i, \ldots, s_{i+n-1}),\ i \geq 0 \tag{1}$$

given the initial state $(s_0, s_1, \ldots, s_{n-1})$. To guarantee that every state has a unique predecessor and successor, f should be written in the form $f(x_1, \ldots, x_n) = x_1 + g(x_2, \ldots, x_n)$ [11]. Some necessary conditions on f and g to generate a de Bruijn sequence are given as follows;

1. To avoid all zero cycle, $f(0, \ldots, 0) = 1$.
2. To avoid all one cycle, $f(1, \ldots, 1) = 0$.
3. To avoid the cycle $(00\ldots01)$, not all of the linear terms exists in f [12].
4. The parity of the truth table of g is 1 [11].
5. The function g is non-symmetric [5], i.e.

$$g(x_2, \ldots, x_n) \neq g(x_n, \ldots, x_2).$$

These conditions are not sufficient and are only related to a very small portion of feedback functions, therefore they are not very useful while searching a feedback function for large n. One way to construct de Bruijn sequences using FSRs is to use linear feedback shift registers (LFSRs) with period $2^n - 1$. De Bruijn sequences are constructed by simply appending 0 to the $(n-1)$-bit $(00\ldots0)$ pattern in LFSR output. The number of distinct de Bruijn sequences generated by this method is bounded by the number of degree n primitive polynomials over $GF(2)$ which is equal to $\phi(2^n - 1)/n$, where ϕ is the Euler-phi function.

Another way of constructing de Bruijn sequences is by taking a Hamiltonian path of n-dimensional de Bruijn graph, also known as the Good's diagram [11]. De Bruijn graph with dimension n is a directed graph with 2^n nodes having an edge from node $(a_1, \ldots, a_n)$ to $(b_1, \ldots, b_n)$ if and only if $a_i = b_{i+1}$ for $1 \leq i \leq n-1$ (See Fig. 1).

Fredricksen and Maiorana [8] proposed an efficient method to generate the lexicographically minimal de Bruijn sequence. Etzion and Lempel [6] provided construction methods that can generate de Bruijn sequences with minimal linear complexity. Games [9] provided a recursive construction that inputs two de Bruijn sequence of order n to produce a de Bruijn sequence of order $n + 1$. Fredricksen [7] and Ralston[21] give a survey of these construction methods.

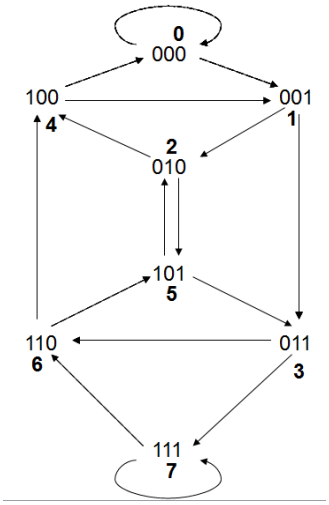

Fig. 1. De Bruijn graph for $n = 3$

2.2 Genetic Algorithms

Genetic algorithms operate on population members by an iterative procedure. Each member represents a candidate solution for the problem of interest, and the ability of each member to survive over generations is measured by a fitness function. As given in Figure 2, a genetic algorithm starts by generating an initial population that contains N_p members. In each generation, population members undergo selection, crossover, and mutation to generate new children, according to predetermined crossover and mutation probabilities. Some of the members are removed from the population in order to reduce the population size back to its initial size. This is repeated until given stopping conditions are satisfied. When genetic algorithm stops, the best solution found so far is given as the output.

```
Generate initial population;
Until stopping condition is satisfied
  Select parents from the population;
  Apply crossover operator to produce children;
  Apply mutation operator to the children;
  Extend the population by adding the children;
  Reduce the population back to its original size;
Output the best population member;
```

Fig. 2. Pseudocode of a generic genetic algorithm

Genetic Algorithms for TSP Genetic algorithms can be used to solve the TSP which is an NP-hard sequence-based combinatorial optimization problem. Given n nodes and their pairwise distances, TSP aims to find a shortest closed tour visiting each node exactly once. Although the exact methods (such as evaluating all possible tours) fall short when the problem size gets large ($n > 30$), heuristic methods obtain near-optimal solutions for real-world applications in a reasonable computation time.

In a genetic algorithm, TSP tours can be represented in various forms, such as path, adjacency, ordinal and rank representation [18]. Most common representation is the path representation, in which a number assigned to each node and the solutions are represented as an ordered sequence of nodes. The TSP has a very natural fitness function, which is given by the tour length.

The success and the performance of genetic algorithms highly depend on the selection of the crossover operator. Nearest neighbor crossover NNX, defined in [22], randomly selects the starting node and finds the neighbors of this node in each parent. The closest unused neighbor is used as the next node, if possible. If all neighbors are used before, NNX randomly selects an unused node. Other crossover examples can be given as; PMX [10], CX [20], EAX [19], NX [15], NEX [17]. It seems that the crossover operators that make use of problem specific information perform better.

3 A Special Type of TSP: TSP$^*_n$

Let $(a_1, \ldots, a_n)$ be the binary representation of integer A $(< 2^n)$. We define $msb_s(A)$ and $lsb_s(A)$ to be the first and last s $(s \leq n)$ bits of $(a_1, \ldots, a_n)$, respectively, i.e.,

$$msb_s(A) = (a_1, a_2, \ldots, a_s),$$
$$lsb_s(A) = (a_{n-s+1}, a_{n-s+2}, \ldots, a_n).$$

Definition 1. *TSP$^*_n$ is a special type of TSP with 2^n nodes where the distance between two distinct nodes $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ is defined to be*

$$d_{A,B} = n - max_s\{s | msb_s(A) = lsb_s(B)\}, \quad s \geq 0.$$

TSP$^*_n$ is constructed using a complete directed graph whose edge weights is calculated using the de Bruijn graph. The distance between nodes A and B corresponds to the number of edges in the shortest path from A to B in the de Bruijn's graph and can take values between 1 and n. As an example, the distance from node 2 and node 7 is 3, since the shortest path $(2 \rightarrow 5 \rightarrow 3 \rightarrow 7)$ has three edges (See Figure 1). Since the graph is directed, the distances are asymmetric.

The tour lengths for TSP$^*_n$ vary between 2^n and $n2^n$. Each cyclically-distinct optimal solution (i.e., the tours with length 2^n) corresponds to a de Bruijn sequence. Given an optimal tour $(A^{(1)}, A^{(2)}, \ldots, A^{(2^n)})$, de Bruijn sequence is generated as

$$(a_1^{(1)}, a_1^{(2)}, \ldots, a_1^{(2^n)})$$

where $a_1^{(i)}$ corresponds to the first bit in the binary representation of the node $A^{(i)}$. Selecting different bit positions only results in the rotation of the same de Bruijn sequence.

Example 1. TSP_3^* has 8 nodes $\{0, 1, \ldots, 7\}$, with the distance matrix given in Figure 3. The length of the tour (0 1 2 5 3 7 6 4) = (000, 001, 010, 101, 011, 111, 110, 100) is 8 and corresponds to the following de Bruijn sequence (00010111). The optimal tour is depicted in Figure 4.

From \ To	0	1	2	3	4	5	6	7
0	-	1	2	2	3	3	3	3
1	3	-	1	1	2	2	2	2
2	2	2	-	2	1	1	3	3
3	3	3	3	-	2	2	1	1
4	1	1	2	2	-	3	3	3
5	3	3	1	1	2	-	2	2
6	2	2	2	2	1	1	-	3
7	3	3	3	3	2	2	1	-

Fig. 3. Distance matrix for the TSP_3^*

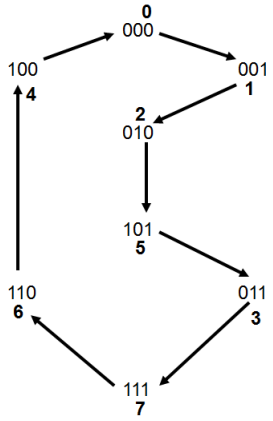

Fig. 4. An optimal TSP_3^* tour

4 Proposed Genetic Algorithm

In this section, we describe the main components of the proposed genetic algorithm.

A natural way to represent de Bruijn sequences is to use the binary representation, however since our underlying problem is the TSP, we used the path representation which is more suitable to solve TSPs.

The fitness function is selected to be the tour length, for which the optimal value is known to be 2^n. Since the optimal solution for TSP_n^* is known unlike other random TSP instances, the genetic algorithm stops whenever the optimal solution is found. Limiting the iteration number to 500 is used as an alternative stopping condition, to stop the genetic algorithm if the population converges to a non-optimal solution.

To generate the initial population, two different methods are used;

- *Method I* simply generates random tours, therefore the quality of the initial population is low in terms of the tour lengths.
- *Method II* uses an heuristic approach which results in higher quality members. This method first starts with a random node and selects the next node randomly from the nodes whose distance is 1 to the current node, if possible, otherwise, an unused node is selected randomly.

At each iteration, N_p couples are selected randomly, and from each couple, one offspring is produced. The new offsprings are added to the population and the population size doubles. Then, population members are sorted based on their fitness values. The best N_p members are moved to the next generation.

NNX[22] is originally proposed for the symmetric TSP, where the distances between nodes are independent of the direction of the edges. Since in TSP_n^*, the distances are asymmetric, selection of the nodes are slightly modified as given in the following example.

Example 2. Let *Parent I* = (1 2 4 7 6 3 0 5) and *Parent II* = (5 2 6 7 0 3 4 1) with tour lengths 17 and 19, respectively. The initial node is randomly chosen to be 3,

$$(3 * * * * * * *).$$

The successors of 3 are 0 and 4 from *Parent I* and *Parent II*, respectively. Since $d_{3,4} < d_{3,0}$, the next node is selected as 4,

$$(3\ 4 * * * * * *).$$

Continuing this way, the offspring is obtained as

$$(3\ 4\ 1\ 2\ 6\ 7\ 0\ 5)$$

with tour length 16, with better fitness value compared to its parents.

To avoid premature convergence, two different mutation operators are used.

- *Mutation I* simply randomly selects two nodes and swaps their positions.
- *Mutation II* is an improvement of *Mutation I*, in which one of the nodes to be swapped (let's say $A^{(i)}$), is selected when $d_{A^{(i-1)},A^{(i)}} + d_{A^{(i)},A^{(i+1)}} > 2$, which means that

$$(\dots A^{(i-1)} \; A^{(i)} \; A^{(i+1)} \dots)$$

part of the member should be updated in order the member to be optimal. Second node to be swapped is selected randomly. The swap operation is applied to the member, if it results in an improvement in the fitness value.

5 Experimental results

In this section, we provide the details of our parameter settings and experimental results. To investigate the effect of population size, initial population generation method and mutation type on the solution quality, the variations given in Table 1 are used.

Population size, N_p	2^5 2^8
Initial Population	Method I - Random Method II - Heuristic
Mutation	No Mutation Mutation I with probability 0.05 Mutation II with probability 0.05

Table 1. Parameter selection of the experiments for $n = 3, \dots, 10$

For $n = 3, \dots, 10$, our experiments are repeated 100 times for each choice of population size, initial population generation method and mutation operator. The algorithm is considered to be successful, if it outputs a tour with length 2^n, i.e. an optimal TSP_n^* tour.

Table 2 and 3 summarizes the results of our experiments using the number of successful experiments (out of 100 trials), average number of iterations for successful trials, and the number of distinct de Bruijn sequences generated in all trials.

For the experiments using Method I with $N_p = 2^5$, the success rate is strongly affected by the mutation operator. Using the mutation operator increases the success rate by 12 and 20 percent with Mutation I and II, respectively. However, this setting is unsuccessful for problems with order $n > 6$. Using higher population size increases the success rates for problems up to order of $n = 8$ (See Table 2).

For the experiments using Method II, the success rates of the algorithms are more than 90 percent for all $n \leq 10$. For small n values, even the initial population generation method manages to generate optimal solutions, however as n gets larger, the heuristic method seems to fall short. The genetic algorithm

manages to generate optimal solutions after small number of iterations, usually less than 10 iterations on the average. Mutation operator seems to have a very limited effect on the results of the problems starting with high quality initial population.

For larger n values, the initial population is generated using Method II, since Method I fails to be successful for $n \geq 9$. Although having a higher population size significantly increases the success rate, due to the large memory requirements, the population size is fixed to 2^5. Since the mutation operators do not have a significant effect on the results when the initial population is generated using Method II, mutation operator is not used for larger problems. All experiments are replicated 10 times and the results are summarized in Table 4.

6 Discussion and Conclusion

In this study, we construct de Bruijn sequences using a new evolutionary method that models the sequences as a special type of traveling salesman tours. The method is randomized and capable of generating all possible de Bruijn sequences. We presented some experimental result for $n \leq 14$.

The proposed genetic algorithm requires n, $n2^n$ and $n2^nN_p$ bits to represent each node, member and the population, respectively. For $n > 14$, the algorithm becomes inefficient mainly due to the memory requirement. More efficient representations such as binary representation will be studied as a future work.

7 Acknowledgments

The author would like to thank Çağdaş Çalik for his help while implementing the code. The author would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

		No Mutation			Mutation I			Mutation II		
n	N_p	Success Rate	Avr. Iter.	Distinct	Success Rate	Avr. Iter.	Distinct	Success	Avr. Iter.	Distinct
3	2^5	91	2.16	2	100	3.28	2	100	1.17	2
4	2^5	77	16.35	16	88	22.46	16	98	22.52	16
5	2^5	51	36.64	51	58	59.08	59	64	43.98	68
6	2^5	0	-	0	0	-	0	0	-	0
3	2^8	100	0.02	2	100	0.47	2	100	0	2
4	2^8	97	6.46	16	97	3.23	16	99	5.16	16
5	2^8	96	8.56	143	99	9.64	132	97	11.07	137
6	2^8	88	22.92	119	88	22.82	111	89	27.46	130
7	2^8	79	39.82	92	73	41.15	87	77	51.23	91
8	2^8	45	116	49	31	95.46	35	33	99.63	36
9	2^8	0	-	0	0	-	0	0	-	0

Table 2. Summary of results obtained using Method I to generate the initial population

		No Mutation			Mutation I			Mutation II		
n	N_p	Success Rate	Avr. Iter.	Distinct	Success Rate	Avr. Iter.	Distinct	Success	Avr. Iter.	Distinct
3	2^5	100	0	2	100	0	2	100	0	2
4	2^5	100	0	16	100	0.06	16	100	0.02	16
5	2^5	100	0.54	199	100	0.61	194	100	0.42	219
6	2^5	100	2.81	155	100	2.69	128	100	2.9	131
7	2^5	100	6.8	113	100	7.74	117	100	6.96	111
8	2^5	100	14.39	103	100	16.83	107	99	16.67	105
9	2^5	100	32.4	100	96	34.19	98	99	27.32	99
10	2^5	97	59.27	97	95	63.84	95	90	60.26	90
3	2^8	100	0	2	100	0	2	100	0	2
4	2^8	100	0	16	100	0	16	100	0	16
5	2^8	100	0	1190	100	0	1191	100	0	1163
6	2^8	100	0	881	100	0	867	100	0	901
7	2^8	100	0.07	208	100	0.04	428	100	0.05	420
8	2^8	100	0.85	205	100	1.19	222	100	1.06	203
9	2^8	100	4.32	129	100	4.21	144	100	3.37	142
10	2^8	100	7.98	110	100	8.49	117	100	9.63	110

Table 3. Summary of results obtained using Method II to generate the initial population

n	Success rate	Avr. Iter.	Distinct
11	8 /10	136.00	8
12	5 /10	118.40	5
13	2 /10	152.00	2
14	3 /10	262.33	3

Table 4. Summary of results obtained using Method II with $N_p = 2^5$, for larger n values

References

1. eSTREAM: ECRYPT Stream Cipher Project. IST-2002-507932 http://www.ecrypt.eu.org/stream, 2004.
2. A. Alhakim. A Simple Combinatorial Algorithm for de Bruijn Sequences. *American Mathematical Monthly*, 117(8):728–732, 2010.
3. S. Babbage and M. Dodd. The Stream Cipher MICKEY. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/015, 2005.
4. C. D. Cannière and B. Preneel. Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005.
5. Çağdaş Çalik, M. Sönmez-Turan, and F. Özbudak. On Feedback Functions of Maximum Length Nonlinear Feedback Shift Registers. *IEICE Transactions*, 93-A(6):1226–1231, 2010.
6. T. Etzion and A. Lempel. Construction of de Bruijn Sequences of Minimal Complexity. *IEEE Transactions on Information Theory*, 30(5):705–708, 1984.
7. H. Fredricksen. A Survey of Full Length Nonlinear Shift Register Cycle Algorithms. *j-SIAM-REVIEW*, 24(2):195–221, 1982.
8. H. Fredricksen and J. Maiorana. Necklaces of Beads in k colors and k-ary de Bruijn Sequences. *Discrete Mathematics*, 23:207–210, 1978.
9. R. A. Games. A Generalized Recursive Construction for de Bruijn Sequences. *IEEE Transactions on Information Theory*, 29(6):843–849, 1983.
10. D. E. Goldberg and R. Lingle. Alleles, Loci and the TSP. In *In Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159. (Edited by J. J. Grefenstette), Lawrence Erlbaum, Hillsdale, NJ, 1985.
11. S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.
12. R. Gonzalo, D. Ferrero, and M. Soriano. Some Properties of Nonlinear Feedback Shift Registers with Maximum Period. *Proceedings of the Sixth International Conference on Telecommunications Systems*, 1998.
13. M. Hell, T. Johansson, and W. Meier. Grain - A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010, 2005.
14. J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
15. S. Jung and B. R. Moon. Toward Minimal Restriction of Genetic Encoding and Crossovers for the Two-Dimensional Euclidean TSP. *IEEE Transactions on Evolutionary Computation*, 6(6):557–565, 2002.
16. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1996.
17. P. Merz. A Comparison of Memetic Recombination Operators for the Traveling Salesman Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 472–479. Morgan Kaufmann, San Francisco, 2002.
18. Z. Michalewicz and D. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, New York, 2000.
19. Y. Nagata and S. Kobayashi. Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 450–457. Morgan Kaufmann, San Mateo, 1997.

20. I. M. Oliver, D. J. Smith, and J. R. C. Holland. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In *Proceedings of the Second International Conference on Genetic Algorithms and their application*, pages 224–230, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.
21. A. Ralston. De Bruijn Sequences-A Model Example of the Interaction of Discrete Mathematics and Computer Science. *Mathematics Magazine*, 55(3):131–143, 1982.
22. H. Süral, N. E. Özdemirel, I. Önder, and M. Sönmez-Turan. An Evolutionary Approach for the TSP and the TSP with Backhauls. In L. M. Hiot, Y. S. Ong, Y. Tenne, and C.-K. Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, volume 2 of *Adaptation, Learning, and Optimization*, pages 371–396. Springer Berlin Heidelberg, 2010.