

Solving One-Dimensional Distance Geometry: a Comparison of GPU Annealing and Branch-and-Prune

A. Mucherino,^{*} F-T. Wu,[†] D.S. Gonçalves,[‡] J-H. Lin[†]

^{*}IRISA, Université de Rennes, Rennes, France

antonio.mucherino@irisa.fr

[†]Academia Sinica, Taipei, Taiwan

austin@as.edu.tw, jhlin@as.edu.tw

[‡]Universidade Federal de Santa Catarina, Florianópolis, SC, Brazil

douglas.goncalves@ufsc.br

Abstract—This paper explores the application of the Quantix GPU annealer for solving the Subset Sum Problem (SSP) and equivalent cycle instances of the one-dimensional Distance Geometry Problem (DGP). We present a comparative study against the exact Branch and Prune (BP) algorithm across various instance classes to evaluate their respective strengths and limitations. Our computational experiments reveal a significant divergence in performance, despite both solvers operating as 64-bit digital systems. While the BP algorithm efficiently solves standard, unstructured instances, it exhibits exponential complexity on highly structured instances where its pruning mechanisms fail. Conversely, the GPU annealer navigates deeply structured data with ease, but struggles with low-density instances. This difficulty arises because the necessary Quadratic Unconstrained Binary Optimization (QUBO) formulation quadratically increases the required bit precision, causing the annealer to hit representation limits faster than traditional methods. We empirically show that the GPU annealer’s efficacy seems to be dictated by the system’s capacity to represent the underlying QUBO model.

I. INTRODUCTION

THE Distance Geometry Problem (DGP) asks whether the vertices of a simple weighted undirected graph G can be embedded into a Euclidean space such that the distances between realized vertices perfectly match their assigned edge weights [10], [12]. When restricted to a single dimension, the DGP in dimension 1 seeks to map these vertices onto a real line. As long as the DGP graph is connected, these DGP instances can be solved by build-up algorithms such as the Branch and Prune (BP) algorithm [9]. This algorithm exploits part of the distance information to construct a binary tree, which is systematically explored and “pruned”, when possible, by using additional information (distances or other). While it has been empirically proven that the BP algorithm can perform well on specific classes of instances [11], it can exhibit an exponential complexity on some other instances. One example of DGP application in one dimension is the clock synchronization problem [17].

A particularly interesting subclass of the one-dimensional DGP arises when the underlying graph G is a cycle. In such a case, in fact, the distance information is mostly exploited for the tree construction, so that *almost* no additional distances are present for possibly verifying the feasibility of the current tree branches during the search. We say “almost” because one distance, but only one, is actually available to perform pruning: this is the final distance that closes the cycle. In other words, the algorithm’s pruning phase can only be applied when the search reaches the tree leaf nodes. This leads to a “paradoxical” situation where the exploration of the BP tree is trivial¹ but the verification of the final distance closing the cycle (the only pruning opportunity) requires exploring an exponentially large search space [4].

It is however important to point out that a different kind of pruning, even if less effective, can be implemented in the BP algorithm. During the search, we can in fact verify whether at the current tree level it is still possible, by using the remaining distances, to construct a feasible solution, i.e. a solution capable of satisfying the final distance that closes the cycle. As confirmed by our computational experiments, this additional pruning strategy can be rather effective for some specific classes of instances.

Cycle DGP instances directly correspond to instances of a very-well known problem in combinatorial optimization: the Subset Sum Problem (SSP) [7]. In fact, any instance of the SSP can be reduced to an instance of the one-dimensional DGP through a simple change of variables. This implies that both the DGP and SSP are NP-hard [15], and, as long as their inputs are restricted to integer values (and to dimension 1 for the DGP), they are both NP-complete.

¹Simply select a vertex and position it at the origin, and choose a walking direction on the graph. Then start to build-up, one by one, the positions for the other vertices by exploiting the only available distance. From one tree layer to the next, the number of branches doubles.

TABLE I
EXPERIMENTS WITH PUBLICLY AVAILABLE BENCHMARK CONTAINING SEVEN SSP INSTANCES.
TIME IS GIVEN IN SECONDS.

instances	p01	p02	p03	p04	p05	p06	p07
n	8	10	21	10	9	6	10
ρ	1.60	0.91	1.00	1.87	1.42	1.20	2.04
Quantix ε	1	1	0	1	0	0	1
Quantix time	1.71	1.71	1.73	1.71	1.71	1.71	1.71
BP time	0.000004	0.000013	0.000093	0.000005	0.000004	0.000005	0.000008

The SSP asks the following question. Given a set $\mathcal{A}$ of n positive integer numbers, as well as an integer target t , is there any subset $\mathcal{S}$ of $\mathcal{A}$ such that the elements of $\mathcal{S}$ sum up exactly to the target t ? The *density index* for SSP instances can be computed as:

$$\rho = \frac{n}{\log_2(\max a_i)},$$

where the denominator represents the number of bits required to represent the largest integer in $\mathcal{A}$. In simple terms, the density index compares *how many integers are in $\mathcal{A}$* against *their magnitude*. For sets $\mathcal{A}$ containing unstructured integers, the absolute hardest SSP instances (and hence the hardest cycle DGP instances) are those having a density ρ equal to 1 [8], so that the number of bits required for representing the integers exactly corresponds to the number of integers in $\mathcal{A}$. However, for sets containing structured integers, as for example integers following a specific pattern, ρ may fail to provide the actual complexity of the instance.

This work aims at contributing to the growing body of research into alternative computing approaches for the SSP and the one-dimensional DGP. One recent major breakthrough is given in the field of optical computing, where two main publications ([5] and [16], authored independently by two research groups), propose the use of specifically designed optical processors. Our problem is here reformulated as a standard matrix-by-vector multiplication, where one of the sizes of the matrix grows exponentially with the instance size n . In both approaches, the information is encoded into the transverse spatial degrees of freedom of light beams using spatial light modulators, so that to perform all operations required by a matrix-by-vector operation in one single go. Some of us are co-authors of the approach proposed in [5], for which a partial implementation has recently been realized in [1], and for which a GPU-based emulation was previously proposed in [3].

Another highly explored alternative is adiabatic quantum computing. This requires the translation of the combinatorial problem into Quadratic Unconstrained Binary Optimization (QUBO) models, so that to solve it by using the so-called *quantum annealers* [14]. In this context, the work in [2] has recently explored the solution of cycle DGP instances on D-Wave machines². We remark that, although based on the artificial manipulation of two different physical phenomena, both optical processor and adiabatic quantum computing approach

have a common drawback: they both represent the instance data in an analog form, with a limited precision accuracy.

In this work, we investigate the possibility to solve cycle DGP and SSP instances through the use of the **Quantix GPU annealer**³ developed by COMPAL ELECTRONICS. Our study explores various instance classes where this digital annealer appears to be more or less effective, and we perform comparisons with the standard BP algorithm already mentioned above. We find this comparison particularly interesting because the digital nature of both approaches may suggest an equivalent behavior when dealing with instances where precision is an important factor. Instead, our experiments will show a divergence in performance of the two compared approaches.

The rest of the paper is organized as follows. Section II introduces the Quantix GPU annealer and details the necessary steps for reformulating the SSP into a QUBO model. Section III presents our computational experiments, providing a brief overview of the BP algorithm before detailing the comparative performance of both solvers across various instance classes. Finally, Section IV concludes the paper by summarizing our findings.

II. THE GPU ANNEALER FOR QUBOS

The Quantix GPU annealer is an advanced computing tool designed to solve large combinatorial optimization problems. While its underlying logic is inspired by quantum physics, Quantix is actually a classical transistor-based computing system, often simply categorized as a *GPU annealer*. Indeed, instead of using quantum bits governed by the laws of quantum physics, Quantix exploits the massive parallel-processing power of several GPU cores. This approach allows the system to achieve “quantum-like” problem-solving speeds using reliable and accessible hardware.

In order to solve a combinatorial problem using the GPU annealer, it is necessary to initially translate it into a QUBO, a specific mathematical model (see Introduction). Once the problem is reformulated, the GPU annealer digitally simulates the physical process of *annealing* (the way metals are gradually cooled to reach their lowest-energy state). In this context, this lowest energy state represents the optimal solution to the given problem. By evaluating thousands of potential solutions simultaneously, the GPU-driven system can swiftly navigate complex data landscapes and find accurate solution candidates in an efficient way.

²<https://www.dwavequantum.com/>

³<https://www.compall.com/zh-hant/>

From the Quantix package, we use in our experiments the standard `QUBO_Solver` function. This function simulates the physical annealing process mentioned above, and we made the choice to use this one because the simulation it performs is close to the real quantum system it takes its inspiration from. We need to point out, however, that the Quantix package also proposes the implementations of other methods for solving QUBOs, where the standard annealing is hybridized with Genetic Algorithms (GAs) to prevent the search from getting trapped in local minima.

We formulate our SSP instances (or our cycle DGP instances in dimension 1) in QUBO form, i.e. as a minimization problem where the objective is $x^T Q x$, where $x \in \{0, 1\}^n$ is a vector of binary variables, as follows. Given the set $\mathcal{A} = \{a_1, \dots, a_n\}$ and the target t of an SSP instance, the problem can be cast as the linear equation:

$$\sum_{i=1}^n a_i x_i = t.$$

Since the binary vector x solves the previous equation if it is a zero for the quadratic function:

$$q(x) = (a^T x - t)^2,$$

where a is a column vector with elements a_i , we can rewrite:

$$q(x) = x^T a a^T x - (2ta)^T x + t^2 = x^T H x + b^T x + c,$$

with $H = a a^T$, $b = -2ta$ and $c = t^2$. By ignoring the constant term c , and by exploiting the fact that $x_i = x_i^2$ (our x_i 's can either hold 0's or 1's), we obtain the following correspondence with a QUBO:

$$Q_{i,i} = H_{i,i} + b_i \quad \text{and} \quad Q_{i,j} = H_{i,j}.$$

Since we can suppose that all integers a_i are positive, our specific QUBO formulation corresponds to a Q matrix where all its non-diagonal elements are strictly positive. Moreover, our QUBO exhibits a linear growth w.r.t. the original number of variables of the SSP (this is not always the case for other NP-hard problems). Finally, we can remark that every element of the Q matrix is formed by only one term of the original SSP, because it is based on one single constraints. In brief, the obtained problem belongs to the class of the easiest QUBO to tackle.

III. COMPUTATIONAL EXPERIMENTS

This section presents in detail our computational experiments. For the BP algorithm (the reader can find a brief explanation of the algorithm below), the C code is available on a public GitHub repository⁴.

The Quantix system used in this work, in its version 1, is installed on a local machine with 20 Intel Xeon Silver 4416+ (800 MHz) with the cache size of 38400KB. Each processor has 20 CPU cores. On this machine, 3 Nvidia GeForce RTX 4090 GPU cards are installed, each GPU card has 16,000 Compute Unified Device Architecture (CUDA) cores. Quantix

is implemented according to the diverse adaptive bulk search (DABS) framework [6], [13], which implements multiple solution pools maintained by a host running on the system's CPUs. Each solution pool is associated with a GPU, as well as with a table keeping information of the encountered good-quality solutions.

With the aim of studying the performances of both BP algorithm and GPU annealer for a variety of situations, the experiments are proposed on three different sets of SSP instances.

The BP algorithm

Since its initial proposal in the context of protein structure determination [9], various descriptions of the BP algorithm have been reported in several publications. This short section has the only aim to recall the main search mechanism employed by the algorithm, in order to make the present publication as self-contained as possible.

For simplicity, we specifically take into consideration an SSP instance, for which the explanation of the pruning phase is easier to present. Basically, the BP algorithm relies on a systematic exploration of a binary tree, where every unique path from the root to a leaf node represents a potential subset $\mathcal{S}$ derived from the initial set $\mathcal{A}$ of the SSP. Each level of the tree corresponds to an integer in $\mathcal{A}$: moving to the left branch signifies the exclusion of that integer from the subset, while moving to the right signifies its inclusion. Although the theoretical size of this tree reaches 2^n nodes, the algorithm implements the so-called *pruning phase* to avoid an exhaustive tree search. However, as already remarked in the Introduction, most of the data (the values of the integer numbers in the SSP) is exploited for the tree construction, and only one distance is left at the lowest tree layer to perform the final feasibility check. In order to improve performance, our implementation of the BP algorithm also relies on an additional pruning phase: if at the current branch level, the subset sum is already larger than the target, or if it is rather impossible to attain the target with all remaining integers, then the current branch can be pruned.

Benchmark instances

Our first experiments are performed on instances of a public benchmark,⁵ where instances are marked with the labels **p01**, **p02**, ..., **p07**. These instances were already used in previous works (see for example [6]) for testing quantum and quantum-inspired technologies.

Table I shows our comparisons between our implementation of the BP algorithm, and the GPU annealer. For every instance, we provide the size n , as well as its density index ρ . For each result, we provide the total time in seconds (for both solvers). Only for the GPU annealer, we also provide the absolute error ε on the target. Indeed, when BP is able to find a solution, the subset sum must precisely match with the given target,

⁵https://people.sc.fsu.edu/~jburkardt/datasets/subset_sum/; see file `subset_sum.html` in folder `~jburkardt/datasets/subset_sum/`; last accessed on 2026-03-29.

⁴<https://github.com/mucherino/DistanceGeometry>

TABLE II
EXPERIMENTS WITH GENERATED SSP INSTANCES HAVING SIZE n EQUAL TO 24 AND WITH VARIABLE DENSITY INDEX ρ .

instances	d171	d150	d133	d114	d104	d100	d092
ρ	1.71	1.50	1.33	1.14	1.04	1.00	0.92
Quantix ε	0	0	0	2	4	2	8
Quantix time	1.52	2.52	4.52	18.64	17.43	16.93	16.83
BP time	0.000005	0.002291	0.000966	0.011205	0.006143	0.006294	0.008851

so that the value of ε is always 0. This is not the case for the GPU annealer, which rather implements an optimization strategy, that might get trapped into local minima.

The BP algorithm is able to find a solution for all instances in a computational time smaller than one millisecond. These instances are indeed not very hard, because of their small size, and because, for most of them, the density ρ is not 1. The only exception is given by the instance labeled **p03**, but there is only a tiny impact on the computational time of BP, because of the relatively small size of the instance. The instance **p03** is particularly interesting because, as pointed out in [6], the number of bits necessary to represent the original integers is large enough to challenge the analog representations of quantum machines such as D-Wave. To this comment, we can also add that this is direct consequence of its density ρ being equal to 1.

Since the GPU annealer is a digital system, it is not directly affected by this issue. Indeed, we can see from Table I that the GPU annealer can find good-quality solutions for all instances. For some instances, however, the solver is not able to identify the optimal subset, but it rather provided a solution very close to the optimal ($\varepsilon = 1$). We have verified that all these sub-optimal solutions are *local* optima: one single bit flip from the given solutions is not able to make us closer to the target. Essentially, the GPU annealer “got stuck” at these sub-optimal solutions. We remark that the results presented in [6] with the GPU annealer, on this same benchmark, exhibit better performances: this is probably because the Genetic Algorithm (GA) method variant was there used. In spite of this, the GPU annealer generally converges within the minimal time budget that can be allocated (about 1 second), for all instances in the benchmark.

Instances with variable density index

The instances we use in these experiments are generated with the aim of stressing the observation in [6] for which instances requiring a higher number of bits for their representation are the hardest to solve by alternative computing approaches. Indeed, the representation limits observed for D-Wave, which is an analog machine, actually also exist for digital systems, but they are attained much farther. To this purpose, our instances are generated with a constant size ($n = 24$), while their density ρ varies. As the density decreases, indeed, more and more bits for the representation of the integers are required. We generate the instances with the C function `ssp_gen_hard` available in the same GitHub repository of the BP implementation.

Even if both systems are based on the nowadays 64-bit standard for their registers, it is important to remark that the GPU annealer works with the QUBO formulation of the SSP. In particular, when the elements of the Q matrix are computed (see Section II), the original integer numbers are either squared, or multiplied with others. As a consequence, the number of bits necessary to represent the elements of Q grows quadratically with the size of the instances. This is the reason why the GPU annealer hits its representation limit faster than a standard CPU core, where the BP algorithm runs.

Table II shows some experiments where we compare the BP algorithm with the GPU annealer. It is evident that the GPU annealer is much more sensitive to the decrease in density of our instances. The instance **d092** is not considered to be a “very hard” instance in general, because of its low density, but still the GPU annealer struggles to converge. This observation makes us consider that the traditional density index may not be the correct measure for the *hardness* of instances when solved with the GPU annealer.

2D-Knapsack SSP instances

SSP is traditionally seen as a special case of the one-dimensional Knapsack problem (every item’s weight is exactly equal to its value, and the capacity perfectly matches the target value). Since they both belong to the NP-complete complexity class, it is possible to map one into another. The instances we use in this final set of experiments are 2D Knapsack instances which are then *disguised* as SSP instances. We proceed this way because the generated instances are particularly hard for the BP algorithm to solve, while they appear to be particularly easy for the GPU annealer.

The 2D Knapsack problem can be stated as follows. Given n items, we know for each of them a value v_i and two distinct resource weights (e.g. weight and volume), denoted as w'_i and w''_i . We are also given a target value V and two maximum capacities W' and W'' . The decision variant of 2D Knapsack asks whether we can select a subset of items such that the sum of their values is V while the maximum capacities W' and W'' are not attained. Without losing generality, we can suppose the two capacities W' and W'' need to be exactly satisfied (so that they can be combined with V to construct a unique target t).

Thus, in order to map 2D Knapsack instances into an SSP, we verify the maximal value for the integers v_i , w'_i , w''_i and define the value

$$b = 1 + \max_i \{v_i, w'_i, w''_i\},$$

TABLE III

EXPERIMENTS WITH 2D KNAPSACK SSP INSTANCES. BOTH THE BP ALGORITHM AND THE GPU ANNEALER PROVIDED A SUBSET FOR WHICH THE SUM OF THE ELEMENTS EXACTLY CORRESPONDS TO THE TARGET ($\varepsilon = 0$).

instances	k35	k40	k45	k50
n	35	40	45	50
Quantix time	10.74	10.74	1.73	2.75
BP time	6.10	82.94	1825.55	1633.87

which we then use to construct the integers in the set $\mathcal{A}$:

$$a_i = v_i + b \cdot w'_i + b \cdot b \cdot w''_i,$$

as well as the target of our SSP:

$$t = V + b \cdot W' + b \cdot b \cdot W''.$$

Notice that this is a classic approach used in complexity theory for problem reduction [7].

Table III shows that the GPU annealer is absolutely the “winner” on the generated set of instances (not only it converges much faster, but it always finds solutions with $\varepsilon = 0$). Instead, as the size n of the instances grows, we can notice that the BP algorithm struggles more and more to find a solution. This is likely to be due to the particular structure of the generated instances. Because of the use of b and b^2 in the formula defining the elements a_i , the magnitude of these integers remains similar while the less significant bits in their representation (the v_i part of a_i) play a fundamental role: they need to sum up exactly to V . As a consequence, the pruning phase in the BP algorithm (see the short description of the BP algorithm above) becomes much less effective, by forcing the BP exploration to approach an exhaustive search.

IV. CONCLUSIONS

In this work, we compared the Quantix GPU annealer and the BP algorithm for solving one-dimensional cycle DGP and SSP instances. Despite both being 64-bit digital systems, they exhibited divergent performance profiles depending on the instance class.

For standard, unstructured SSP instances, BP was exceptionally efficient, whereas the GPU annealer exhibited larger computational times and occasionally stalled at local minima. In fact, as the density ρ of the instances decreases, the GPU annealer hits a representation limit much faster than BP. The annealer, indeed, relies on a QUBO formulation that squares and multiplies integers to construct the Q matrix, so that the required bit precision grows quadratically. Thus, for QUBO-based digital solvers, computational hardness seems to be dictated heavily by the system’s capacity to represent the mathematical model, rather than just the traditional density index.

Conversely, the GPU annealer excels on highly structured data, such as 2D Knapsack instances mapped to the SSP. In these scenarios, the particular structure of the data neutralizes BP’s primary pruning mechanism, forcing an inefficient, near-exhaustive search. The GPU annealer easily navigates this

structural complexity, converging quickly to exact solutions where the sum of the subset elements exactly matches the target t .

Future works will be devoted to the definition and study of other instance classes, with an extended comparison to D-Wave machines, as well as to other types of digital annealers. In parallel, we also plan to perform a theoretical study on the appropriate measures for actual instance hardness in different experimental setups.

ACKNOWLEDGMENTS

The authors wish to thank the 4 reviewers for the fruitful comments. This work was partially supported by the ANR project EVARISTE (ANR-24-CE23-1621).

REFERENCES

- [1] M.G. Damaceno, A. Mucherino, R. Medeiros de Araújo, P.H. Souto Ribeiro, N. Rubiano da Silva, *Exploring the Scalability of Parallel Processing in SLM-based Optical Systems*, Optics and Laser Technology **202**, 115346, 2026.
- [2] C.A. Freitas Bernardes, P. Castellucci, D.S. Gonçalves, E. Duzzioni, A. Mucherino, *Adiabatic Quantum Computing for the Subset Sum Problem: Preliminary Studies*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedC-SIS25), Workshop on Computational Optimization (WCO25), Krakow, Poland, 635–639, 2025.
- [3] S.B. Hengeveld, A. Mucherino, *A GPU Approach to Distance Geometry in 1D: an Implementation in C/CUDA*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedCSIS22), Workshop on Computational Optimization (WCO22), Sofia, Bulgaria, 333–336, 2022.
- [4] S.B. Hengeveld, A. Mucherino, *The Discrete Side of Distance Geometry: a Focus on the 1-dimensional Case*, Proceedings of Discrete Math Days (DMD22), Santander, 301–305, 2022.
- [5] S.B. Hengeveld, N. Rubiano da Silva, D.S. Gonçalves, P.H. Souto Ribeiro, A. Mucherino, *An Optical Processor for Matrix-by-Vector Multiplication: an Application to the Distance Geometry Problem in 1D*, Journal of Optics **24**(1), 015701, 2022.
- [6] J.-R. Jiang, Y.-C. Shu, Q.-Y. Lin, *Benchmarks and Recommendations for Quantum, Digital, and GPU Annealers in Combinatorial Optimization*, IEEE Access **12**, 125014–125031, 2024.
- [7] R.M. Karp, *Reducibility Among Combinatorial Problems*. In: R.E. Miller, J.W. Thatcher, J.D. Bohlinger (Eds.), “Complexity of Computer Computations”. New York: Plenum. 85–103, 1972.
- [8] J.C. Lagarias, A.M. Odlyzko, *Solving Low-Density Subset Sum Problems*, Journal of the ACM **32**(1), 229–246, 1985.
- [9] L. Liberti, C. Lavor, N. Maculan, *A Branch-and-Prune Algorithm for the Molecular Distance Geometry Problem*, International Transactions in Operational Research **15**, 1–17, 2008.
- [10] L. Liberti, C. Lavor, N. Maculan, A. Mucherino, *Euclidean Distance Geometry and Applications*, SIAM Review **56**(1), 3–69, 2014.
- [11] L. Liberti, C. Lavor, A. Mucherino, *The Discretizable Molecular Distance Geometry Problem seems Easier on Proteins*. In: [12], Springer, 47–60, 2013.
- [12] A. Mucherino, C. Lavor, L. Liberti, N. Maculan (Eds.), *Distance Geometry: Theory, Methods and Applications*, 410 pages, Springer, 2013.
- [13] K. Nakano, D. Takafuji, Y. Ito, T. Yazane, J. Yano, Sh. Ozaki, R. Katsuki, R. Mori, *Diverse Adaptive Bulk Search: a Framework for Solving QUBO Problems on Multiple GPUs*, IEEE conference proceedings, International Parallel and Distributed Processing Symposium Workshops (IPDPSW23), 314–325, 2023.
- [14] A. Rajak, S. Suzuki, A. Dutta, B.K. Chakrabarti, *Quantum Annealing: An Overview*, Philosophical Transactions of the Royal Society A, Mathematical, Physical and Engineering Sciences **381**(2241), 2023.
- [15] J. Saxe, *Embeddability of Weighted Graphs in k -Space is Strongly NP-hard*, Proceedings of 17th Allerton Conference in Communications, Control and Computing, 480–489, 1979.

- [16] J. Spall, X. Guo, T.D. Darrett, A.I. Lvovsky, *Fully Reconfigurable Coherent Optical Vector-Matrix Multiplication*, Optics Letters **45**(20), 5752–5755, 2020.
- [17] P. Verissimo, M. Raynal, *Time in Distributed System Models and Algorithms*. In: “Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems”, S.K. Shrivastava, S. Krakowiak (Eds.), Springer, 1–32, 1999.