in: Nanotechnology 7(1), pp. 307-314, 1996

Emergent Computation by Catalytic Reactions

Wolfgang Banzhaf*†

and Peter Dittrich

and Hilmar Rauhe

Department of Computer Science, Dortmund University

Baroper Str. 301, 44221 Dortmund, GERMANY

banzhaf,dittrich,rauhe@LS11.informatik.uni-dortmund.de

*Author to whom correspondence should be addressed

†Informatik Centrum Dortmund (ICD), Dortmund, Germany

Abstract

Recently, biochemical systems have been shown to possess interesting computational properties. In a parallel development, the chemical computation metaphor is becoming more and more frequently used as part of the emergent computation paradigm in Computer Science. We review in this contribution the idea behind the chemical computational metaphor and outline its relevance for nanotechnology. We set up a simulated reaction system of mathematical objects and examine its dynamics by computer experiments. Typical problems of computer science, like sorting, parity checking or prime number computation are placed within this context. The implications of this approach for nanotechnology, parallel computers based on molecular devices and DNA-RNA-protein information processing are discussed.

1 Introduction

The idea of using natural systems for computational purposes has long been pondered. Haken, for instance, has proposed to use laser mode competition as a way to recognize patterns [1]. Others have proposed chemical and biological systems as useful for computational purposes [2], [3], [4]. Recently, biochemical systems have been shown to possess interesting computational capabilities [5], [6], [7], [8]. For the first time, biochemical systems have been used to do some actual computation.

The pioneering usage of DNA for computation in [7] gave an idea of the potential power of molecular computation. Its strength is based on the high density of information storage in DNA and the amount of molecules that are usable for parallel computation. Molecular computation in vitro, however, is still at its infancy and the computational concepts which take advantage of its potential power are still under exploration. Suggestions have been made already for universal molecular computers based on graphs [9], [10] or molecular Turing machines [6], [11], [12].

The properties of DNA, i.e. its ability to store information and use it for the specifity of molecular interactions (DNA-DNA or DNA-Enzyme based) seem to predestine DNA for the usage in information processing at a molecular level. On the other hand the methods for the manipulation of DNA currently available via molecular biology [13] have very limited programming capability and computational concepts will have to adapt to the characteristics of the molecular environment.

In a second, and nearly simultaneous development natural metaphors for computation are more and more frequently used in computer science. Already established fields like artificial neural networks [14] and evolutionary computation [15, 16] have recently got company by approaches using the chemical metaphor to computation as their central notion. In this way concepts are being developed which might well in the future converge with the technological abilities to realize molecular computers. In this contribution, we shall concentrate on this new concept in computer science.

The chemical metaphor for computation has been discussed at various places in the literature. Haken proposed to use chemical reactions for information processing [1] (chapter 9.8). In 1988 the Γ-language was introduced into Computer Science by Banatre et al. [17, 18]. Their primary motivation was the inadequacy of the imparative programming paradigm for massively parallel computers of the coming decades. Γ is based on a single data structure called multiset. A computation can be seen as a transformation of the multiset by consuming elements of the multiset and by producing new members according to specific rules. Banatre et al. [17, 18] also discuss similar applications of the paradigm as we do here, although their point of view is parallelizability of the approach. Rather, our point of view will be the dynamics of such a system.

Later, the idea was further developed into the "CHemical Abstract Machine (CHAM)" by Berry and Boudol [19]. The CHAM adds new features to the paradigm by considering solutions of multisets and allowing for the definition of membrane-like encapsulation of subsolutions. The authors also examine more theoretical aspects of the approach, e.g. its relation to concurrent λ-calculi [20] and to Milner's calculus of mobile processes [21, 22].

Another contribution to the chemical computation metaphor stems from Fontana et al. who study organizational aspects of resulting interaction webs. In a series of papers [23, 24] a computational system was devised which is based on the λ-calculus. Elements are interpreted as functions which react with each other. Through intermediate states, λ-expressions are evaluated with the reaction partner as an argument and the result of this computation is considered as an analog of a reaction product. The work of Fontana et al. is mainly concerned with the static aspects of organization, however, and does not focus on the resulting macroscopic dynamics.

Certain dynamical aspects in systems based on chemical computation have been considered in our own work [25, 26, 27], where a system of computational objects has been set up and its evolution was studied. Our point of view on chemical computation is to consider information processing in algorithmic reactors. Simply spoken, an *algorithmic reactor* is a computer which computes by the analog of chemical reactions. The algorithmic reactors that will be presented in this paper implement single algorithms rather than universal computers.

The principle of all information reactors is the implementation of parallel algorithms which require relatively simple local interactions of their computational "molecules". The computation of a reactor is encoded in its hardware, since the construction of the molecules allows only limited interactions. Through their interactions, the "molecules" execute the algorithm. Though the presented algorithmic reactors lack programmability, they do computation rather efficently. This approach could serve as a reasonable compromise between universal computation and what will be implementable in vitro. We do, however, expect programmable molecular computers to arrive at the scene sooner or later.

The rest of this paper is organized as follows: Section 2 gives an informal description of the chemical computation metaphor as used by us. Section 3 discusses some implications of this view for nanotechnology in general. Section 4 discusses three examples of algorithms, starting with a very simple problem: parity checking, followed by a more complicated problem, sorting. The final problem is prime number computation. Section 5 summarizes our results and gives our conclusions.

2 The chemical metaphor

Chemical reactions take place under specific physical and structural conditions. An event that is reproducible under the same conditions, such as an interaction between molecules, is potentially able to carry information. The information is stored in the composition of the molecules, therefore a reaction, as it causes changes of the composition of the reacting molecules, can be seen as an act of information processing. The capacity of information processing increases with the speed of the reaction and the complexity and the number of the reactands. Beyond the microscopic events a reaction system which contains many reactands should be able to show a trend in its macroscopic behavior that can be interpreted in a meaningful and unique sense. The strategy to create an algorithmic reactor out of a system of chemical or quasi-chemical reactions is to establish local interactions in a manner that they are elementary operations of an algorithm whose computation emerges as macroscopic behavior of the whole system. In principle this can be done by selection of the physical conditions, the composition of the molecules and the specificity of their reactions. Preferably, algorithmic reactors implement parallel algorithms based on relatively simple local interactions with low interdependence in order to make full use of the number of reactands in the system. In comparison to in vitro systems the "molecules" of the presented reactors and their interactions have a complexity which suggests molecules like DNA for possible in vitro implementations.

In today's desktop computers the information content of a stored bit sequence depends heavily on its position in memory. The interpretation of data is done by an instruction flow which refers to specific pre-defined memory locations. Changing the positon of data randomly will disrupt the computational process.

In the chemical computation metaphor used here, no spatial structure is used. Data are viewed as certain particles (objects) which collide arbitrarily with other particles. When they collide they are able to interact and may produce other objects or they may be destroyed. If nothing happens the collision is called *elastic* [23].

It is important to consider a collision as a simple microscopic event. The desired computation emerges out of many collisions of a vast number of objects. This requires that the structure of the object and the interaction mechanism must be simple, e.g. describable by a small finite state machine or by simple rules.

The metaphor is summarized in the following table.

data	substances or molecules
processing	chemical reaction
algorithm	substances and their reaction laws

3 Molecular computing and nanotechnology

Feynman was the first to consider molecular computing seriously with his seminal work [28, 29] from 1959. An entire field has developed since, with computation using quantum effects as its main concern [30]. Thermodynamic considerations and the possibility of computation to become reversible were of high importance, too [31], [32].

In the early Eighties, nanotechnology was incepted as a new field [33]. It has since developed into an engineering discipline [34]. Although Drexler considers biotechnology as an important path toward nanotechnology as it demonstrated the feasibility of such an approach, mainstream studies have focussed on engineering realizations in the traditional top-down approach.

Our emphasis will be somewhat different. In our oppinion, chemistry with molecules diffusing and colliding in solutions offers enough potential for molecular computing to go along with. Therefore we are concerned about signals in the form of concentration peaks and do not consider immobility as a precondition for achieving molecular computation.

The approach, therefore, stands between what is now envisioned in nanotechnology and what is already realized in bulk-matter electronics. Our reasoning is based on the fact that the majority of molecular machinery in biological systems is working in solution. Concepts of self-organisation, self-assembly and self-programming are at the center of our attention. It goes without saying that emergent phenomena [35] play a central role in this argument.

4 Three examples of algorithms

4.1 The general set-up

For the following examples we use an experimentation system with these components:

- *A soup (population, reactor vessel) of objects.*
  These objects may be character sequences [36], lambda-expressions [23], binary strings [25], [37] or numbers. In a basic setting, the soup has no spatial structure so that its state can be noted as a concentration vector.

- *A reaction or collision rule.*
  The reaction rule defines the interaction among two objects s_1 and s_2 which may lead to the generation of a new object s_3. This is denoted as:

$$s_1 + s_2 \Longrightarrow s_3 \tag{1}$$

- *An algorithm to run the system.*
  In this contribution we use two algorithms that can also be found with minor modifications in [23, 25, 36, 37].

Reactor algorithm I

1. Randomly select two objects s_1, s_2 from the soup, without removing them.

2. If there exists a reaction $s_1 + s_2 \Longrightarrow s_3$ and the filter condition $f(s_1, s_2, s_3)$ holds, replace a randomly selected object of the soup by s_3.

The replaced objects form the *dilution flux* of the system. The *filter* f can be used to block lethal objects and to introduce elastic collisions. An object is said to be *lethal* if it is able to replicate in an unproportionally large number in almost any ensemble configuration. The algorithm instantiates the interaction scheme:

$$s_1 + s_2 + X \longrightarrow s_1 + s_2 + s_3 \tag{2}$$

This means, that s_1 and s_2 are not consumed and act as catalysts of the reaction. The *raw material* X is used to balance the equation and does not appear explicitly in the system. It could be interpreted as computational resources like processing time or memory ([38]. pp 373). Reaction systems which exhibit this kind of reaction scheme are able to form *hypercyclic* organisations [39, 40, 41].

Reactor algorithm II

1. Randomly select two objects s_1, s_2 from the soup, without removing them.
2. If there exists a reaction $s_1 + s_2 \Longrightarrow s_3$ and the filter condition $f(s_1, s_2, s_3)$ holds, replace the object s_1 of the soup by s_3.

This algorithm instantiates the interaction scheme:

$$s_1 + s_2 \longrightarrow s_2 + s_3 \tag{3}$$

This means that the object s_1 is transformed into s_3 with the catalytic help of s_2. An important difference to algorithm I is that there is no dilution flux, e.g. an object s_1 that does not react with any object will stay in the soup forever.

To measure the running time of an algorithm we shall call M iterations of the two steps of the algorithm a *generation*,. M is also the reactor or soup size. Thus, during one generation, every object of the reactor will have had a chance to react with another object .

4.2 Parity checking

In this first example a reaction system should decide whether a given binary sequence contains an odd or even number of ones. This problem has a linear complexity with respect to the length of the given sequence. It should illustrate how a linear loop can be performed.

The objects for this task are binary strings of arbitrary length and an operator object s_w. In order to prepare the reaction system, the soup P is initialized with n_i units of the input sequence $s_i \in \{0,1\}^*$ and n_w units of the operator object s_w. Then the reactor-algorithm I is iterated for some generations.

For the reaction $s_1 + s_2 \Longrightarrow s_3$ the following collision rules are applied:

- If the length of $s_1 = \dots b_2 b_1 b_0$ is greater than 1 set $s_3 = \dots b_2 b$, with $b = (b_1 \neq b_0)$ and $b_2, b_1, b_0, b \in \{0,1\}$. That means that s_3 becomes s_1 where the first two bits b_1, b_0 are replaced by their exclusive-or product $(b_1 \neq b_0)$.
- If s_1 is equal to $0, 1$ or s_w, replicate s_1 $(s_3 = s_1)$. This amplifies the result when it appears.

For instance, the reactions performed in the run of Figure 1 with the input $s_i = 01101$ are:

$$s_w + 01101 \Longrightarrow 0111 \tag{4}$$

$$s_w + 0111 \Longrightarrow 010 \tag{5}$$

$$s_w + 010 \Longrightarrow 01 \tag{6}$$

$$s_w + 01 \Longrightarrow 1 \tag{7}$$

$$s_w + 1 \Longrightarrow 1 \tag{8}$$

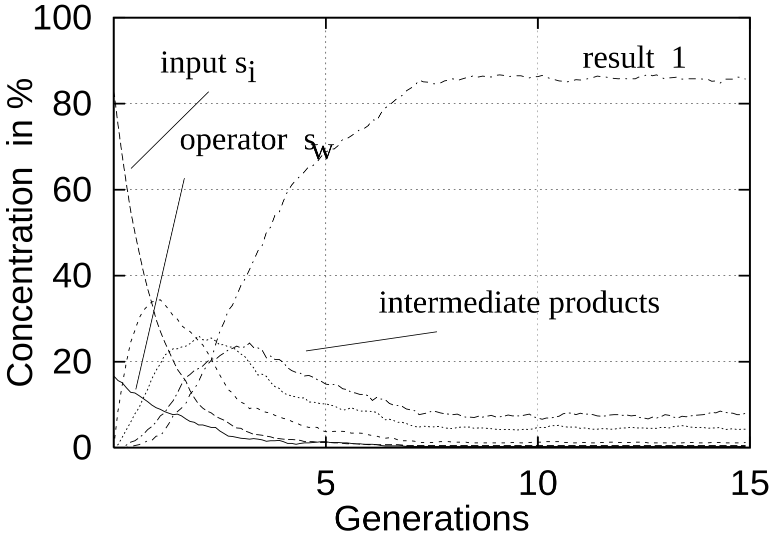

Figure 1: Parity computation by an algorithmic reactor. The reactor is filled with $n_i = 800$ objects of the input $s_i = 01101$ and n_w objects of the operator objects s_w. During the computation intermediete substances are produced and displaced. Their concentration peaks indicate different steps of the linear loop performed.

$$s_w + s_w \Longrightarrow s_w \tag{9}$$

The last two reactions constitute a replication of the result (former) and a self-replica tion of the operator strings s_w (latter), respectively.

Figure 1 shows the results of a typical simulation. The different steps of the loop become noticeable by concentration peaks of intermediate substances. The result is the substance with a high concentration at the end of the simulation. If it consists of 1s the answer to the decision problem becomes ”yes”.

4.3 A sorting algorithm

In the second example, we would like to devise an algorithm for a real number sequence to become sorted. The property of being sorted should *emerge* from the interactions of participating subsequences and numbers. The algorithm should be tolerant to the addition of new numbers during the computation process as well as to other perturbations.

Our starting point is again the chemical computation methaphor. In analogy to elements in a reaction vessel, consider a pool of numbers realized in the memory of our computer. These numbers come in multiple copies and have the possibility to react with each other upon encounter. In case of a reaction, the numbers stick together forming (part of) a sequence that grows in length with successful further reactions. One could say that a polymerization reaction between numbers takes place. Such a reaction, however, is only possible if certain conditions are fulfilled by its reaction partners. Otherwise, the encounter is elastic and results in an unchanged pool.

The idea is to devise a reaction condition which incorporates the sorting criterion. A naïve approach to this end is to provide the possibility of a number to ”react” with a sequence if the last number of the already existing sequence is, say, smaller than the number trying to react with it. In this way, sorted sequences grow longer only by adding (sequences of) larger numbers.

Formally, if we consider elementary numbers to be symbolized by $n_1, n_2, ..., n_L$, and $N_1(k), N_2(k), ...N_L(k)$ to be subsequences of arbitrary length k with last (rightmost) element $n_1, n_2, ..., n_L$ then such a reaction (collision) rule might be described as:

$$N_i(k) + n_j \rightarrow N_i(k) + N_i(k) \circ n_j \rightarrow N_i(k) + N_j(k+1) \qquad \text{, iff } n_i < n_j \tag{10}$$

$$N_i(k) + n_j \rightarrow N_i(k) + N_i(k) \qquad \text{, iff } n_i = n_j \tag{11}$$

$$N_i(k) + n_j \rightarrow N_i(k) + n_j \qquad \text{, else.} \tag{12}$$

where "$\circ$" indicates concatenation. As we can see, the reaction is enzymatic, since $N_i(k)$ is unchanged. Also, one of the the two reactants is usually able to replicate.

The second reaction is merging the two numbers into one. For single numbers or only one subsequence participating, this case could be treated as an elastic reaction (like (3)). But the second reactant could be a subsequence as well, in which case its first (leftmost) element would have to be compared to the last (rightmost) element of the former sequence. The merging reaction does, therefore, lead to elongated sequences even in case of equal outmost numbers.

Since the entire system is set up as a competitive system (reactions are enzymatic), it is expected that long (polymerized) sequences carrying numbers in correct order should dominate the pool after sufficiently many reactions have taken place. As it turns out, however, this is not the case! All reactions quickly turn elastic rendering the system stable.

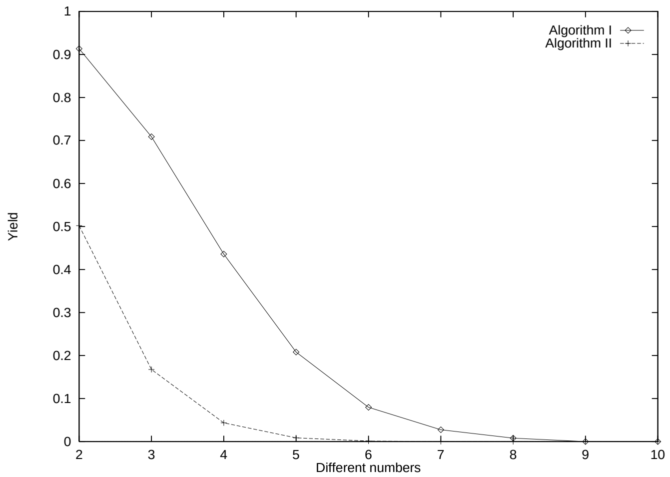

Figure 2: Yield of fully sorted sequences over length of sequences in naive approach. Yield falls very quickly to 0 for algorithm II, and slower for algorithm I. This indicates stabilization of the system by missing reaction pathways. Soup size $M = 1000$.

Figure 2 shows the yield of fully sorted sequences as a function of the number K of participating *different* numbers. It should be kept in mind, that the size of the pool is considerably larger than K.

The second curve in Figure 2 shows the behaviour of a system with more pressure towards proliferating reactions. I.e., if a sequence is able to be produced more often than another, this will help it to become more frequent in the pool. Not surprisingly, also in this case the number of elastic reactions quickly grows, ultimately leading to a stable system.

This behaviour demonstrates quite clearly the importance of devising suitable reactions for the system to reach the anticipated goal. In the naive approach taken, the interactions were obviously set up in the wrong way: The probability of reactions decreases quickly with an increasing length of strings. There are too many reactions leading into dead ends, from where no reaction paths lead any further, and the system quickly gets stuck.

Another approach, and as it turns out the correct one, is to carefully provide for reaction paths even for larger sequences. We shall do so by allowing copy reactions of a form usual in DNA double strands in addition to the reactions (1) - (3).

A formalization reads like this: Suppose, two subsequences have already been formed:

$$N_3(3) = \{n_1, n_2, n_3\} \tag{13}$$

$$N_4(3) = \{n_1, n_3, n_4\} \tag{14}$$

where indices in n_1, n_2, n_3, n_4 indicate their respective order:

$$n_1 < n_2 < n_3 < n_4 \tag{15}$$

We now allow for matching reactions which generate intermediate double-stranded states:

$$\begin{pmatrix} \{n_1, -, n_3, n_4\} \\ \{n_1, n_2, n_3, -\} \end{pmatrix} \tag{16}$$

Gaps, symbolized by "$-$" are becoming detectable since matching reactions enforce connections. In a second step, these gaps are filled by copies of missing elements, upon which single strands form again:

$$\begin{pmatrix} \{n_1, n_2, n_3, n_4\} \\ \{n_1, n_2, n_3, n_4\} \end{pmatrix} \rightarrow N_4(4) + N_4(4) \tag{17}$$

As we can see immediately, this type of reaction does not suffer from the same problem as the first one. Yield remains 100 % for all sets of numbers considered. Figure 3 shows how the complete solution emerges within the pool. We can see also, how various transient states are assumed by the system. In order to arrive at ordered sequences of full length the system has to first build shorter subsequences. Since reactions do happen nearly always (elastic reactions are confined to cases where identical sequences encounter each other) this is easy to achieve.

Figure 4 shows the concentration development of the largest subsequence at any one moment. This concentration grows within bounds (constituting at times even around 1 % of the pool population), it does however, never reach a large concentration. Thus, even without knowing which number is largest in the pool, a concentration increase over a critical threshold does indicate the end of the sorting process and the macroscopic result.

We have seen the importance of interactions and how they serve as a means to destabilize the present state of a population. For the chemical computation metaphor to be useful, it is necessary to identify and realize reaction pathways to the result wanted. If for any reason reactions generate more frequently useless intermediary results than useful ones, the algorithm will either get stuck or be at least very inefficient.

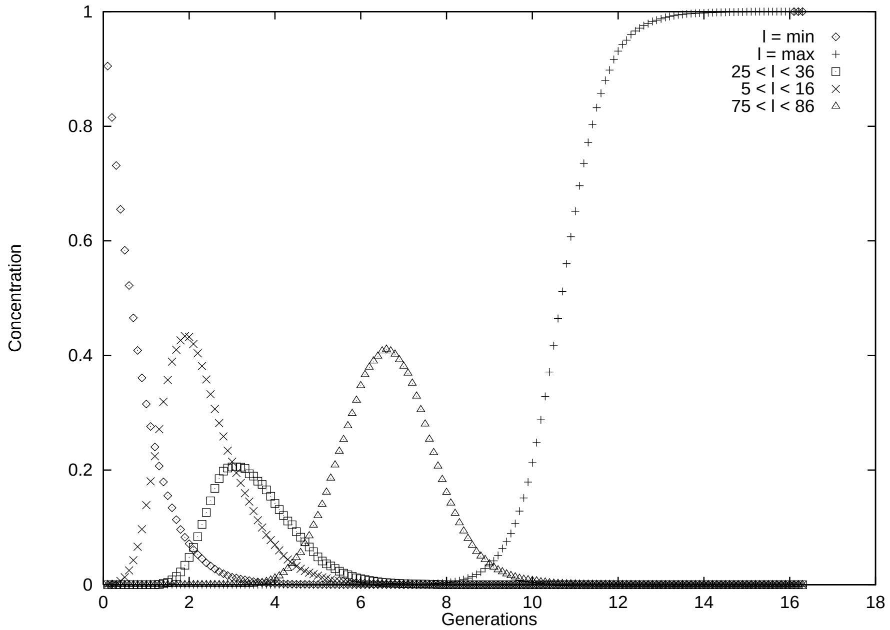

Figure 3: Concentrations of sequences in different length windows. Concentration of shortest sequences (equivalent to numbers) is decaying quickly, concentrations of intermediate length sequences is transient, growing and falling in an orderly fashion, concentration of strings of maximum length ($L = 100$) is growing monotoneously and stabilizing. Population size: 10000.

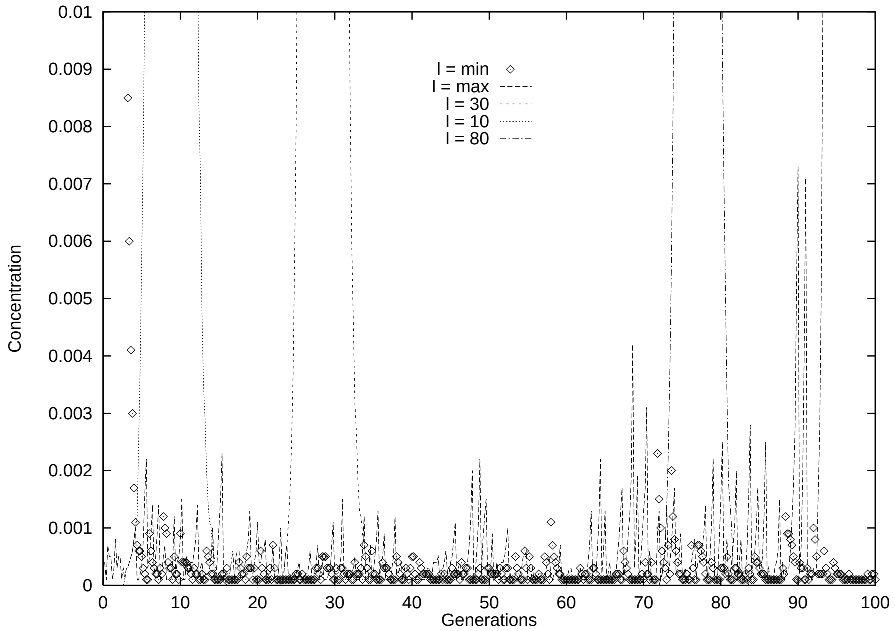

Figure 4: Concentrations of sequences of maximal length in different length windows. Concentration grows within bounds before taking off at around 95 generations indicating that the maximum has been reached. Population size: 10000.

4.4 A prime number generator

In the third example we would like to use a reaction system to produce prime numbers (see also ref [18]). Here we will demonstrate that the population size is critical for a successfull computation. For reasons that will become clear later, by increasing the population size a phase transition occurs with respect to the qualitative behavior of the system, e.g. to the production of prime numbers.

Starting with a reactor of M numbers selected randomly from the set $\{2, 3, \ldots, \text{max}\}$ with $M << max$ the following requirements must be met:

1. Prime numbers should be generated either directly or through intermediate numbers.
2. Prime numbers and intermediate numbers with a low number of prime factors should have advantages in terms of stability over numbers with many prime factors.

The first point is taken into account by using the mathematical division as the collision or reaction rule. More precisely, the product s_3 of the collision $s_1 + s_2 \Longrightarrow s_3$ is defined as

$$s_3 = s_1/s_2 \qquad \text{if } s_1 \bmod s_2 = 0 \text{ and } s_1 \neq s_2 \ . \tag{18}$$

Otherwise, i.e. if s_1 cannot be divided through s_2 without remaining, the collision is elastic. In order to stabilize ensembles of prime numbers or numbers "nearly" prime, thus meeting the second requirement, reactor algorithm II is used. Hence, a prime number is never replaced and every collision with s_1 being prime becomes elastic. A number s_1 is therefore only replaced if a reaction $s_1 + s_2 \Longrightarrow s_3$ successfully produces a new number s_3.

Figure 5 shows a simulation where the population size M is set to 100 and max = 10000. The concentration of prime numbers increases until no non-prime is left. The entire process could be termed "emergence" of the feature prime.

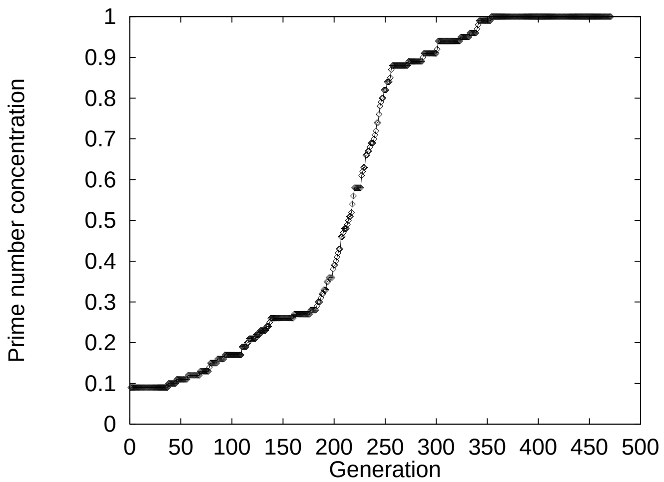

Figure 5: A single run for the prime number problem where at last every object inside the reactor is a prime number. The soup is initialized with random numbers out of $\{2, \ldots, 10000\}$. Soup size: $M = 100$.

Figure 6 shows how critically this behavior depends on the population size. For different soup sizes M the concentration of prime numbers is measured after 700 generation. For each M 30 runs of the reactor algorithm II have been performed.

As is well known from the theory of phase transitions, this sort of behavior, i.e. the increase in fluctuations near the critical point is typical for a phase transition.

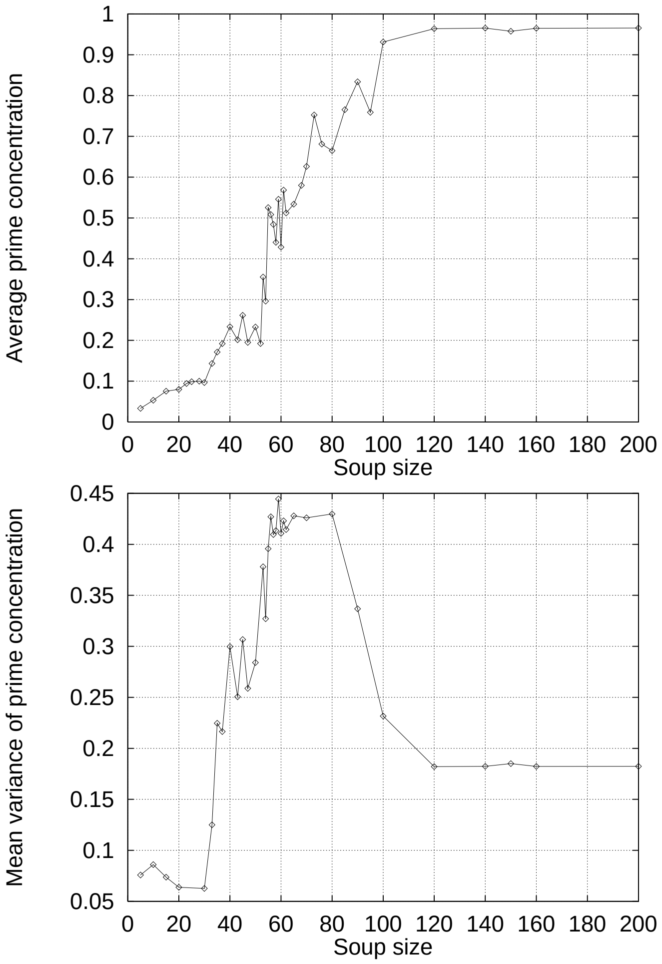

Figure 6: Dependence of prime number production on the population size. The figures summarize a series of runs of the reactor algorithm II with different soup sizes. Each dot represents 30 runs with the same soup size M. In the first graph the avarage prime conzentration at the end of each run (generation 700) is displayed. For small soup sizes ($M < 40$) the algorithm runs mostly into a dead end so that the soup still contains a lot of non-prime numbers. With a larger soup size ($M > 100$) the reactor is nearly always able to transform every number to prime number resulting in a prime number conzentration of 100%. In the vicinity of the phase transition ($40 < M < 100$) a strong fluctuation of the number of resulting primes can be observed. The second figure shows the mean variance is depicted.

Table 1: Overview of different reactor algorithms used.

Application	*Objects*	*Reaction*	*Algorithm*
Parity check	binary strings	two-bit exclusive-or	reactor-algorithm I
Sorting	number sequences	conditional concatenation match & copy	reactor-algorithm I reactor-algorithm II
Prime number computation	numbers	division	reactor-algorithm II

5 Summary and conclusions

In this contribution it has been demonstrated on an abstract level how computation can be achived by chemical systems. The results will hopefully give insight into the design of molecular reactors which can be used for information processing. Two different kinds of reactor algorithms have been employed.

The parity checking example served as an introduction into the general concepts put forward here.

The sorting example shows that the collision mechanism has to be carefully designed. Slight modifications of the reaction rule lead to a massive increase in efficiency. For larger systems, this is not only a matter of efficiency but a prerequisit for reaching the desired goal.

The prime number example demonstrated that reactor size is a critical quantity. As always, it is decisive to have reaction pathways at hand which do not lead into dead ends but keep enough reaction paths open toward the state aimed at (high prime number concentration). Here the numbers can be viewed as simple machines that manipulate other numbers by dividing them [1] A small soup size would not provide enough useful "machines" (e.g. numbers that are able to divide other numbers) at the beginning to initiate the avalanche-like process depicted in Figure 5.

The presented implementations of algorithmic reactors are specialized on the efficient computation of defined problems and could be a compromise between universal molecular computers and what is realizable with today's molecular technology. We anticipate, however, that general purpose molecular computing will become feasible in the future (see also [27]).

Considering the chemical metaphor it can be assumed that DNA and other biomolecules are a suitable material for in vitro implementation. Under the assumptions of the chemical metaphor as it was put forward here three factors limit the power of molecular computation: the number of molecules which are available per computation, the speed of the chemical reactions in a reactor and the amount of information that can be stored per molecule. In this context the usage of DNA would allow for two considerable advantages in molecular computation: Firstly, the amount of information that can be stored per DNA molecule is very high, secondly, DNA does not need to be generated in large quantities. Macroscopic effects can be detected even if caused by only.a few molecules which can be amplified using standard techniques of molecular biology. [7].

[1] The term "enzymatic numbers" has been suggested to us for reflecting this feature.

Despite the fact that the reactors presented in this paper still make use of hypothetical interactions they show a strategy to implement parallel algorithms for molecular computation and give insight in the reactor's design and system dynamics that will determine the behavior of in vitro reactors.

Acknowledgement

This work has been funded in part by Deutsche Forschungsgemeinschaft under contract Ba 1042/2-1.

References

[1] H. Haken, *Pattern formation and pattern recognition - an attempt at a synthesis*, in: H.Haken (ed.), Pattern formation by dynamical systems and pattern recognition, Springer, Heidelberg, 1979

[2] C.H. Bennett, R. Landauer, Scientific American, **254** (1985)

[3] M. Conrad, *Integrated precursor architecture as a framework for molecular computer design*, Microelectronics J., **24** (1993) 263

[4] see, for instance, *International Workshop on Information Processing in Cells and Tissues (IPCAT95)*, Liverpool, September 1995, R. Paton (ed.), The University of Liverpool, UK.

[5] A. Arkin and J. Ross, *Computational Functions in Biochemical Reaction Networks*, Biophys. J., **67** (1994) 560

[6] A. Hjelmfelt, E.D. Weinberger and J. Ross, *Chemical implementation of neural networks and Turing machines*, Proc. Natl. Acad. Sci. USA, **88** (1991) 10983

[7] L.M. Adleman, *Molecular Computation of Solutions to Combinatorial Problems*, Science, **266** (1994) 1021

[8] M. Okamoto, K. Tanaka, Y. Maki, S. Yoshida, *Information Processing of Neural Network System Composed of "Biochemical Neuron": Recognition of Pattern Similarity in Time-variant External Analog Signals*, Information Processing in Cells and Tissues (IPCAT 95 Proceedings), R. Paton (ed.), Department of Computer Science, The University of Liverpool, 1995

[9] R. J. Lipton, *Speeding up Computation via Molecular Biology*, unpublished manuscript, 1995

[10] B. Mayoh, *Biological Computation is Universal*, unpublished manuscript, 1995

[11] D. Beaver, *A Universal Molecular Computer*, unpublished manuscript, 1995

[12] P.W.K. Rothemund, *A DNA and restriction enzyme implementation of Turing Machines*, unpublished manuscript, 1995

[13] J. Sambrook, E. F. Fritsch, T. Maniatis, *Molecular Cloning*, Cold Spring Harbor, New York, 2nd ed, 1989

[14] R. Hecht-Nielsen, *Neurocomputing*, Addison Wesley, Reading, MA, 1989

[15] J.J. Holland, *Adaption in natural and artificial systems* University of Michigan Press, Ann Arbor, 1994 (2nd ed.)

[16] H.-P. Schwefel, *Evolution and Optimum Seeking*, Wiley, New York, 1995 (2nd ed.)

[17] J.-P. Banatre, A. Coutant, D. Le Metayer, *A parallel Machine for Multiset Transformation and its Programming Style*, Future Generation Comp. Sys. **4** (1988) 133

[18] J.-P. Banatre, D. Le Metayer, *The Gamma Model and its Discipline of Programming*, Sci. Comp. Prog., **15** (1990) 55

[19] G. Berry, G. Boudol, *The chemical abstract machine*, Theor. Comp. Sci., **96** (1992) 217

[20] G. Boudol, *Towards a λ-calculus for concurrent and communicating systems*, in: TAPSOFT 1989, Lecture Notes in Computer Science, Vol. 351, Springer, Berlin, 1989

[21] R. Milner, J. Parrow, D. Walker, *A caluculus of mobile processes*, LFCS, Edinburgh University, ECS-LFCS-89-85, 1989

[22] R. Milner, *Functions as Processes*, ICALP '90, Lecture Notes in Computer Science, Vol. 443, Springer, Berlin, 1990

[23] W. Fontana, *Algorithmic Chemistry*, in: Artificial Life II: Proceedings of the 2nd ALife Workshop, C.G. Langton et al. (eds.), Addison Wesley, Reading, MA, 1991, 159

[24] W. Fontana, L. Buss, *"The arrival of the Fittest": Toward a theory of biological organization*, Bull. Math. Biol., **56** (1994) 1

[25] W. Banzhaf,*Self-replicating sequences of binary numbers – Foundations I and II: General and Strings of Length $N = 4$*, Biological Cybernetics, **69** (1993) 269 and 276

[26] W. Banzhaf, *Self-organisation in a system of binary strings*, in: R. Brooks and P. Maes, Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Cambridge MA, MIT Press, 1994

[27] W. Banzhaf,*Self-Organizing Algorithms Derived from RNA Interaction* in: M. Banzhaf and F. H. Eeckman, Evolution and Biocomputation: Computational Models of Evolution, Berlin, Springer 1995

[28] R. P. Feynman, *There's Plenty Room at the bottom*, Eng. and Sci, **23** (1960) 22

[29] R. P. Feynman, *Quantum mechanical computers*, Foundations of Physics, **16** (1986) 507

[30] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. Royal Society London, **1818** (1985) 97

[31] R. Landauer, *Irreversibility and Heat Generation in the Computing Process*, IBM J. Res. Dev., **3** (1961) 183

[32] C, H. Benett, *The Thermodynamics of Computation - a Review*, Int. J. Theor. Phys., **21** (1982) 905

[33] K. E. Drexler, *Molecular engineering: An approach to the development of general capabilities for molecular manipulation*, New York, Anchor Press, 1986

[34] K. E. Drexler, *Nanosystems*, New York, John Wiley, 1992

[35] S. Forrest, *Emergent Computation*, Cambridge (MA), The MIT Press, 1991

[36] S. Kauffman, *Origins of Order*, Chapter 7, Oxford University Press, Oxford, 1993

[37] M. Thürk, *Ein Modell zur Selbstorganisation von Automatenalgorithmen zum Studium molekularer Evolution*, PhD. dissertation, Universität Jena, 1995

[38] T.S. Ray, *An Approach to the Synthesis of Life*, in: Artificial Life II: Proceedings of the Second Artificial Life Workshop, C.G. Langton, C. Taylor, J. Doyne Farmer S. Rasmussen (eds.), Morgan Kaufmann, San Francisco, 1991, 371

[39] M. Eigen, P. Schuster, *The Hypercycle: a Principle of Natural Self-Organisation, Part A*, Naturwissenschaften, **64** (1977) 541

[40] M. Eigen, P. Schuster, *The Hypercycle: a Principle of Natural Self-Organisation, Part B*, Naturwissenschaften, **65** (1978) 7

[41] M. Eigen, P. Schuster, *The Hypercycle: a Principle of Natural Self-Organisation, Part C*, Naturwissenschaften, **65** (1978) 341