Adding Numbers with DNA

Piotr Wąsiewicz, Jan J. Mulawka
The Institute of Electronic Systems

Witold R. Rudnicki, Bogdan Lesyng
The Int. Centre for Math. & Comp. Modell.

Abstract— **The new algorithm of DNA computing for adding binary integer numbers is presented. It requires the unique representation of bits placed in test tubes treated as registers. Amplification step used for the carry operation allows in theory to add numbers at the same quantity of elementary operations, regardless of a number of bits used for representation. New notation proposed in this paper allows for efficient and abstract description of the technical operations on DNA.**

Keywords— **DNA computing, molecular computing, two-step addition, binary numbers.**

I. Introduction

CURRENT development work in traditional electronic computers continues to be restricted by hardware problems [1]. Researchers and computer experts are convinced that some alternative technologies will appear. Quantum computing and molecular computing are potential candidates for such technologies, and are under extensive development. To perform computing different molecules may be used [2], [3]. However, at present it seems that DNA fragments are most suitable for this purpose.

In DNA computing [4], [5] information is stored in DNA molecules. DNA computing may be considered as a set of processing steps on DNA molecules for solving a specific problem according to a precisely defined procedure. A solution is reached by the exclusive use of genetic engineering operations on DNA such as hybridization, denaturation, ligation, PCR, etc. An executed operation supplies some DNA molecules as the result, which is identified usually by electrophoretical fluorescent or radioactive method.

DNA molecules are linear polymers built from four building blocks - nucleotides denoted by symbols A, C, G and T. Since DNA polymers are composed of four nucleotides, they represent chains of symbols over 4-letter alphabet. Therefore DNA computing is adequate for processing symbols and logical structures which has been reported in a number of publications [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. The emphasis, however, is stressed on general implementation of DNA computer [1], [17], [18], [19], [20], [21] and how to simulate the problem-solving processes, especially solving NP-complete problems [6], [22], [23].

Recently high-density oligonucleotide arrays so called DNA chips were developed as tools for sequencing by hybridization *(SBH)* [24]. It is possible to design and synthesize in situ on the support using light-directed solid phase combinatorial chemistry [25] the square inch high-density oligonucleotides arrays for monitoring the expression levels of nearly all (about 6500) yeast genes equalling about 14 *Mb* of information [26], [27]. The progress in this area allows DNA integrated circuits to be basic devices for extremely miniaturized DNA computers [28], [29], [30].

So far nobody has proposed an effective molecular algorithm for performing addition, the task which is trivial on electronic computers. There are two very good reasons for that - addition is not trivial to perform with DNA molecules. Parallel nature of DNA computing is well suited to problems requiring scanning large solution space of single, relatively complicated operation [31], [32], [33].

In this paper we describe algorithms, which would perform addition with the help of DNA molecules using standard molecular biology techniques along with the nonstandard application of the DNA chips. We think, that it is very interesting, to see how simple and effective, two-step computational algorithm can be implemented with DNA molecules and techniques of molecular engineering.

II. Fundamental Operations on DNA Molecules

DNA molecules are directional polymers, due to the details of the biochemical structure and synthesis. Their beginning is denoted as 5' and end as 3'. Due to specific stereochemical interactions between A:T and C:G nucleotides DNA molecules can form antiparallel duplexes, provided that their sequence is complementary - allowing to form A:T and C:G pairs. Therefore in double stranded DNA the information is stored in both strands, in standard and complementary sequence. Transcription between standard and complementary encoding is straightforward and is often used in the presented below algorithm.

P. Wąsiewicz and J.J. Mulawka are from the Institute of Electronic Systems, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland, pwas@ieee.org , {pwas,jml}@ise.pw.edu.pl ; W.R. Rudnicki and B. Lesyng are from Interdisciplinary Centre for Mathematical and Computational Modelling, Warsaw University, Pawinskiego 5A, 02-306 Warsaw, Poland, {W.Rudnicki,lesyng}@icm.edu.pl

0-7803-6583-6/00/$10.00 © 2000 IEEE

Single or double DNA fragments are often called oligonucleotides or oligos, primers, strings and strands. In DNA computing a DNA *string* is represented by a sequence of four basic nucleotides and is usually described by letters A, T, G, C. It may exist as a separate DNA fragment or within a longer one e.g. a string a may be denoted by a sequence: $5'AGTC3'$ or may exist within a longer string $z = 5'AGAAGTCCTA3'$. A formal language may be created from DNA strings. The set of all single DNA strings over the alphabet $\Lambda = \{A, T, G, C\}$ is called the basic language of DNA computing and denoted by Λ^*. Up to now several DNA computing notation standards was worked out e.g. DNA-Pascal [34], splicing [35], [36], [37], [4] and other [38], [39], [32], [17]. Here we introduce our symbolic representation, which is useful for molecular binary operations.

Digits $5'$, $3'$ denoting *orientation* of a DNA string can be replaced with symbols $|$ $>$. *The length* of the string a is denoted by: $|a|$, and its value is equal to a number of symbols forming the string a e.g. $|AGTC|$ has a length of four basic symbols: nucleotides. Using exemplary strings we can write:

$$\begin{array}{l} a = |a >= 5'AGTC3' \\ b = |b >= |TCAGTCTAG > \\ z = |z >= |AGAAGTCCTA > \Leftrightarrow \\ \Leftrightarrow z = 5'AGA * a * CTA3' \\ s =< s| = 3'GATGACTGA5' \\ |a| = 4, |z| = 10 \end{array}$$

It should be noticed that a null string denoted by a symbol ε is a set with zero basic symbols. Thus $|\varepsilon| = 0$. In DNA computing the null string represents logical zero. *A right part* of the string a is described by a symbol $''$ in the upper, right index of the letter a: a'', and *a left part* of the string by a symbol $'$ in the upper, right index of the letter a: a'. If a number of string parts is greater than three, then in the upper, right letter index an ordinal number is placed e.g. the string a is divided into four parts: $a', a^{ii}, a^{iii}, a^{iv}$.

A string *complementary* to a is described by the same letter, but with an added symbol tilde ($\tilde{}$) this means $\tilde{a}$. Two complementary strings a and $\tilde{a}$ create after hybridization a double stranded string $\hat{a}$ made of complementary pairs $A = T, T = A, C \equiv G, G \equiv C$. Note that a string with an orientation $5' \rightarrow 3'$ is always an upper string or a single string, and a single string with an orientation $3' \rightarrow 5'$ should be underlined.

$$\hat{a} = \begin{array}{c} |a > \\ < \tilde{a}| \end{array} = \left\lceil \begin{array}{c} a \\ \tilde{a} \end{array} \right\rfloor$$

To the described below DNA chip strings $\underline{a}, \underline{\tilde{b}}, \underline{\tilde{a}}$ are attached. They are underlined to mark their 3'5' orientation. Other strings $\tilde{a}, b, a$ are annealed to them and can be extracted together with the array. This linear representation of square arrays is quite conventional.

$$\begin{pmatrix} \tilde{a} & b & a \\ \underline{a} & \underline{\tilde{b}} & \underline{\tilde{a}} \end{pmatrix}$$

Operations on DNA oligos [40], [41] may be described in the following way:

1. *Hybridization or Renaturation* means connecting of single complementary DNA strings and forming double stranded molecules. This operation is caused by cooling down the test tube reaction solution and denoted by symbols heat ↓.
2. *Denaturation* means disconnecting single complementary strings from double stranded DNA molecules and is caused by heating the test tube reaction solution. Usually this operation is connected with the operation of *mixing* the solution. It is denoted by heat ↑.
3. *Cutting* of a double DNA string into two parts is performed in DNA computing with the help of enzymes. This means that a given string d may be digested by the enzyme in the presence of a hybridized complementary to d (at least in the neighbourhood of a place to cut) string denoted by a letter c. The enzyme with an ordinal number equal to 5 cuts the string d together with the string c what is described below.

$$\left\lceil \begin{array}{ccc} - & {}_5d & - \\ \underline{\quad} & \underline{{}_5c} & \underline{\quad} \end{array} \right\rfloor = \left\lceil \begin{array}{cccc} - & d' & d'' & - \\ \underline{\quad} & \underline{c'} & \underline{c''} & \underline{\quad} \end{array} \right\rfloor \Rightarrow$$
$$\Rightarrow \left\lceil \begin{array}{cccccc} - & d' & + & - & d'' & - \\ \underline{\quad} & \underline{c'} & \underline{-} & \underline{+} & \underline{c''} & \underline{\quad} \end{array} \right\rfloor \Rightarrow$$
$$\Rightarrow \left\lceil \begin{array}{ccc} - & d' & \\ \underline{\quad} & \underline{c'} & \underline{+} \end{array} \right\rfloor + \left\lceil \begin{array}{ccc} + & d'' & - \\ \underline{\quad} & \underline{c''} & \underline{\quad} \end{array} \right\rfloor$$

A sign $+$ at the side of a DNA string describes a sticky end of it shorter than the nearest complementary oligo. A sign $-$ at the right side of the DNA string describes a sticky end of it longer than the nearest complementary string. The same signs at both ends of complementary strings mean that these strings form a double stranded oligo with blunt ends. Note that the sign $+$ may be additionally applied to mark a symbolic disjunction between two hybridized primers, and the sign $*$ to denote concatenation of strings (after hybridization and ligation), and the sign $-$ to lengthen a string (of course, only in the equations). These rules are obligatory only within brackets $\lceil$ and $\rfloor$ or (and).
4. *Concatenation* of two strings is a string formed by placing the second string after the first string without any gap. In DNA computing joining of two strings is done during hybridization and ligation. They form together a longer single string. In order to concatenate

two oligos a and b the complementary to them in the place of joint, hybridized *third one* is needed. Usually at least eight complementary pairs without a gap are necessary (four pairs for each joining string). The third string c is a concatenation of the oligo complementary to the first string right part a'' and the oligo complementary to the second string left part b'.

$$c = \widetilde{a''} * \widetilde{b'} =< TCAGAGTC|$$

Thus concatenation of two strings a and b in the presence of the complementary, hybridized to them third one c is denoted by:

$$ab = a * b \overset{*}{=} a + b \overset{c}{=} \{a, b\} \text{ or}$$
$$a + b \overset{*}{\Rightarrow} a * b$$
$$ab = |AGTCTCAGTCTAG >$$
$$|ab| = 13$$

The symbol $*$ means in this case the concatenation operation. The null string is the neutral element for concatenation this means $\varepsilon * a = a * \varepsilon = a$.

5. *Amplification (PCR)* encreases a number of double DNA strings chosen by specially designed primers two times in each cycle. The ends of these primers (square brackets) denote ends of amplified oligos. A number of PCR cycles is given in the upper, right corner of the right square bracket. If the number is unknown it is replaced by a sign \$. After tens of amplification cycles in the test tube there are millions of chosen DNA fragments copies, which are in the majority.

$$\text{heat} \downarrow; \;\; \hat{e} \approx \widehat{a[e]^{\$}b}; \;\; \text{heat} \uparrow;$$

Given above amplification of double string can be described in another way as an algorithm:

$$\left[\begin{array}{ccccccc} \tilde{a} & * & \tilde{e} & - & - & * & \tilde{b} \\ [& & & & p_2]^{\$} & & \\ & & [p_1 & & & &]^{\$} \\ \underline{a} & \underline{*} & \underline{-} & \underline{-} & \underline{e} & \underline{*} & \underline{b} \end{array}\right] \Rightarrow$$

$$\Rightarrow \left[\begin{array}{ccccc} \tilde{e} & - & - & * & \tilde{b} \\ [& & p_2]^{\$} & & \\ [p_1 & &]^{\$} & & \\ \underline{a \;\; * \;\; - \;\; - \;\; e} & & & & \end{array}\right] \Rightarrow \left[\begin{array}{c} \tilde{e} \\ \underline{e} \end{array}\right];,$$

where three amplification cycles are presented, and additional primers p_1, p_2 are short oligos complementary to small parts of the given double string.

In the cycle of amplification single strings may be lengthened from its 3' end up to their complementary 5' end e.g.:

$$\text{heat} \downarrow; \left[\begin{array}{cc} [w &]^{\$} \\ \underline{-} & \underline{v} \end{array}\right] \Rightarrow \left[\begin{array}{c} \tilde{v} \\ \underline{u} \end{array}\right]; \text{heat} \uparrow;$$
$$\text{heat} \downarrow; \left[\begin{array}{cc} - & x \\ \underline{[} & \underline{y]^{\$}} \end{array}\right] \Rightarrow \left[\begin{array}{c} \tilde{x} \\ \underline{x} \end{array}\right]; \text{heat} \uparrow;$$

$$\text{heat} \downarrow; \left[\begin{array}{c} \tilde{v} \\ \underline{v} \end{array}\right] \Leftarrow \left[\left[\begin{array}{ccc} + & w & \\ \underline{\;} & \underline{z} & \underline{+} \end{array}\right]^{\$}\right]; \text{heat} \uparrow;$$

Every amplification described above is done in one cycle between cooling (heat ↓) and heating (heat ↑).

6. *Mixing* of DNA fragments enables their uniform distribution. It improves search for good hybridizations in the space of all possible ones.

7. *Extracting* of DNA fragments with specific sequences from other DNA strings can be performed in several ways.

Complementary strings can be separated to one tube and standard to the second tube with use of DNA arrays. The operation is described by symbols *sep*. It should be noticed that in symbolic representation strings in one tube are separated by commas, but sets of strings in different tubes are separated with semicolons. All set symbols are within set brackets.

$$\left\{\left[\begin{array}{ccc} a & + & b \\ \underline{\tilde{a}} & \underline{+} & \underline{\tilde{b}} \end{array}\right]\right\} \Rightarrow \left\{\left(\begin{array}{ccc} a & b & \\ \underline{\tilde{a}} & \underline{\tilde{b}} & \underline{\tilde{c}} \end{array}\right), \left(\begin{array}{ccc} \tilde{a} & & \tilde{b} \\ \underline{a} & \underline{c} & \underline{b} \end{array}\right)\right\} \Rightarrow$$

$$\Rightarrow \left\{\left(\begin{array}{ccc} a & b & \\ \underline{\tilde{a}} & \underline{\tilde{b}} & \underline{\tilde{c}} \end{array}\right); \left(\begin{array}{ccc} \tilde{a} & & \tilde{b} \\ \underline{a} & \underline{c} & \underline{b} \end{array}\right)\right\} \Rightarrow \{a, b; \tilde{a}, \tilde{b}\}$$

Standard strings can be extracted from a string set using DNA chips and this operation is denoted by symbols *mex* - an abbreviation of *multiextraction*. In similar way complementary strings can be extracted from a string set and that operation is denoted by symbols *mex'*. Double strings can be separated from other strings and the operation is described by symbols *mex"*.

Standard strings can be changed to their complementary, adequate strings:

$$\{a, d\} \Rightarrow \left\{a, d, \tilde{a}, \tilde{b}, \tilde{c}, \tilde{d}\right\} \overset{*}{\Rightarrow} \left\{\left[\begin{array}{ccc} a & * & d \\ \underline{\tilde{a}} & \underline{*} & \underline{\tilde{d}} \end{array}\right], \tilde{b}, \tilde{c}\right\} \overset{mex''}{\Rightarrow}$$
$$\overset{mex''}{\Rightarrow} \left\{\left[\begin{array}{ccc} a & + & d \\ \underline{\tilde{a}} & \underline{+} & \underline{\tilde{d}} \end{array}\right]\right\} \overset{mex'}{\Rightarrow} \left\{\tilde{a}, \tilde{d}\right\}$$

This operation is denoted by a symbol c. Operation described by a symbol n changes complementary strings to their standard, adequate strings.

III. Representation of Numbers

Binary representation of numbers is used. Each bit is coded by a specific sequence, which is long enough to be able to recombine specifically with the complementary sequence. Bits after connecting create a linker sequence, which is recognized by a sequence specific endonuclease. There are two possible representations of the number - standard and complementary as is seen in Tab. 1. To perform addition it is necessary to have one number represented by the standard and complementary DNA strings. In case where both sequences are represented as a mixture of standard and complementary strings, it is possible with simple procedure to filter standard and complementary sequences which are used as representations of first and second numbers, respectively.

Table 1. DNA bit representation

Bit (position code)	4	3	2	1	0
Standard code	e	d	c	b	a
Complementary code	$\tilde{e}$	$\tilde{d}$	$\tilde{c}$	$\tilde{b}$	$\tilde{a}$

For example a number equal to 11 will be represented as is described in Tab. 2.

Table 2. DNA number representation

Bit (position code)	4	3	2	1	0
Standard code	-	d	-	b	a
Complementary code	-	$\tilde{d}$	-	$\tilde{b}$	$\tilde{a}$

Addition of 11 (coded as a standard sequence) and 13 (coded as a complementary sequence) is denoted in Tab. 3.

Table 3. DNA adding operands and result representation

Bit (position code)	4	3	2	1	0
Standard code	-	d	-	b	a
Complementary code	-	$\tilde{d}$	$\tilde{c}$	-	$\tilde{a}$
Result - complementary code	$\tilde{e}$	$\tilde{d}$	-	-	-

Bits which are set to zero are represented by *empty* strings, therefore in our example the number 11 is represented as *dba*, the number 13 as $\tilde{d}\tilde{c}\tilde{a}$, and the result as $\tilde{e}\tilde{d}$ in standard and complementary representation, respectively.

IV. Algorithm Description

Assuming that we have at our disposal three registers: an operation register, and a carry register and a result register, standard algorithm for binary number addition can be expressed as follows

```
Copy two numbers to the operation register
In first step for each bit position:
    If both bits are one - carry is one,
        operation bit is zero
    Else if both bits are zero - carry is zero,
        operation bit is zero
    Else - carry is zero, operation bit is one
Shift carry bits equal to one by one bit
Copy operation and carry bits equaling one
to the operation register
In second step to each operation bit group
treated as a number:
    Add appropriate carry bit eliminating
        redundant operation bits
    Not involved bits move
        to the result register
End
Move remained bits in the operation register
to the result register
Result is in the result register.
```

This algorithm can be implemented with DNA molecules and test tubes, which are laboratory representations of computer registers. Numbers that are to be added are represented as double stranded strings, with both standard and complementary strings. We present the example of application of our algorithm in Tab. 4, where the addition of 11 and 13 is presented.

V. Conclusions

In this paper the original algorithm for adding integer numbers was presented and novel notation for the operations on DNA was proposed. All carry operations are executed within the single operation, allowing addition in the same number of steps, regardless of a number of bits representing the number. As a result this algorithm is particulary well suited for adding enormously large numbers, say million bit numbers, without increasing a number of operations.

Futher design of more complicated computer devices is expected [42], [43]. There are many practical experiments to be performed. Obviously there are technical limitations to the numbers that can be represented by DNA molecules, arising from the need of uniqueness of sequences representing bits as well as from the present fabrication limits of DNA chips [44], [45], [46], [47]. We may hope, however, that limits of technically possible systems will be growing fast enough to make application of the DNA computing feasible.

Acknowledgments

This work was supported by the Polish State Committee for Scientific Research, through the KBN grant number 8T11F00816. This article was processed with the LaTeX 2_ε macros package.

References

[1] M.S. Malone, Beyond semiconductors, *The Microprocessor: A Biography*, Springer-Verlag (1995).

[2] J.R. Heath, P.J. Kuekes, G.S. Snider, R.S. Williams, A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology, *Science* **280** (1999) 1716-3721.

[3] C.P. Collier, E.W. Wong, M. Belohradsky, F.M. Raymo, J.F. Stoddart, P.J. Kuekues, R.S. Williams, J.R. Heath, Electronically Configurable Molecular-Based Logic Gates, *Science* **285** (1999) 391-393.

[4] G. Pun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag Berlin Heidelberg (1998).

[5] J.J. Mulawka, P. Wąsiewicz, Molecular Computing - new challenge of information technology (in Polish), *Informatyka Polish Journal* **7/8** (1998) 36–39.

[6] L.M Adleman, Molecular computation of solutions to combinatorial problems, *Science* **266** (1994) 1021–3024.

[7] R.J. Lipton, *DNA* Solution of Hard Computational Problems, *Science* **268** (1995) 542–545.

[8] *The Bibliography of Molecular Computation and Splicing Systems*, at http://liinwww.ira.uka.de/bibliography/Misc-/dna.html

[9] M. Amos and P.E. Dunne, *DNA Simulation of Boolean Circuits*, Technical Report CTAG-37009, Department of Computer Science, University of Liverpool UK (1997).

[10] T. Leete, J. Klein, H. Rubin, Bit Operations Using a DNA Template, *Proc. of the Third DIMACS Workshop on DNA-based Computers*, Philadephia, USA (1997) 159–366.

[11] J.J. Mulawka, P. Wąsiewicz, A. Płucienniczak, Another Logical Molecular *NAND* Gate System, *Proc. of the 7th Int. Conf. on Microelectronics for Neural, Fuzzy, and Bio-Inspired Systems (MicroNeuro'99)*, Granada, Spain (1999) 340–345.

[12] J.J. Mulawka, A. Malinowski, Calculation of Logic Functions by DNA Computing, *Proc. Nat. Conf. On Evol. Alg. and Global Optim.*, Potok Zloty (1999).

[13] P. Wąsiewicz, P. Borsuk, J.J. Mulawka, P. Węgleński, Implementation of Data Flow Logical Operations via Self-Assembly of DNA, *Lect. Not. Comp. Sci.* **1586** (1999) 174–382.

[14] P. Wąsiewicz, T. Janczak, J.J. Mulawka, A. Płucienniczak, The Inference Via DNA Computing, *Proc. Congress on Evolutionary Computation (CEC'99)*, **2**, Washington, USA (1999) 988–393.

[15] P. Wąsiewicz, A. Malinowski, R. Nowak, J.J Mulawka, P. Borsuk, P. Węgleński, A. Płucienniczak, DNA Computing: Implementation of Data Flow Logical Operations, accepted to *Future Generation Computer Systems Journal*, Elsevier.

[16] P. Wąsiewicz, T. Janczak, J.J. Mulawka, A. Płucienniczak, The Inference Via Molecular Computing, *Cybernetics and Systems: An International Journal*, Taylor & Francis, vol. **31/3** (2000) 283-315.

[17] S. Roweis, E. Winfree, R. Burgoyne, N. Chelyapov, M. Goodman, P. Rothemund, L. Adleman, A Sticker-Based Model for DNA Computation, *Journal of Computational Biology*, **5(4)** (1998) 615-629.

[18] M. Ogihara and A. Ray, *Simulating Boolean Circuits On a DNA Computer*, TR 631, University of Rochester, Computer Science Department (1996).

[19] M. Ogihara and A. Ray, *The Minimum DNA Computation Model and Its Computational Power*, TR 672, University of Rochester, Singapore (1998).

[20] K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama, M. Hagiya, State Transitions by Molecules, *Proc. of the Fourth DIMACS Workshop on DNA-based Computers*, Pennsylvania, USA (1998) 87–300.

[21] J.J. Mulawka, P. Borsuk, P. Węgleński, Implementation of the Inference Engine Based on Molecular Computing Technique, *Proc. IEEE Int. Conf. Evolutionary Computation (ICEC'98)*, Anchorage (1998) 493–498.

[22] L.M. Adleman, P.W.K. Rothemund, S. Roweis, E. Winfree, On applying molecular computation to the data encryption standard, *Journal of Computational Biology*, **6(1)** (1999) 53-63.

[23] M. Amos, *DNA Computation*, Ph.D. Thesis, Department of Computer Science, University of Warwick UK (1997).

[24] D. Shalon, S.J. Smith, P.O. Brown, A DNA Microarray System for Analyzing Complex DNA Samples Using Two-color Fluorescent Probe Hybridization, *Genome Research* **6** (1996) 639-645.

[25] S.P.A. Fodor, et al., Light-Directed, Spatially Addressable Parallel Chemical Synthesis, *Science* **251** (1991) 767–773.

[26] M. Schena, et al., Quantitative Monitoring of Gene Expression Patterns with a Complementary *DNA* Microarray, *Science* **270** (1995) 467–470.

[27] L. Wodicka, et al., Genome-wide expression monitoring in saccharomyces cerevisiae, *Nature Biotechnology* **15** (1997) 1359–3367.

[28] S. Meier-Ewert, E. Maier, A. Ahmadi, J. Curtis, H. Lehrach, An automated approach to generating expressed sequence catalogues, *Nature* **361** (1993) 375-376.

[29] M. Chee, et al., Accessing Genetic Information with High-Density DNA Arrays, *Science* **274** (1996) 610-614.

[30] Q. Liu, A. Frutos, L. Wang, A.J. Thiel, S.D. Gillmor, T. Strother, A.E. Condon, R.M. Corn, M.G. Lagally, L.M. Smith, Progress Toward Demonstration of a Surface Based DNA Computation: a One Word Approach to Solve a Model Satisfiability Problem, *Proc. of the Fourth DIMACS Workshop on DNA-based Computers*, Pennsylvania, USA (1998) 15–25.

[31] E.B. Baum, Building an associative memory vastly larger than the brain, *Science* **268** (1995) 583–585.

[32] V. Mihalache, Prolog approach to dna computing, *Proc. of the IEEE International Conference on Evolutionary Computation (ICEC97), Special Session on DNA-based Computation* (1997).

[33] P. Wąsiewicz, J.J. Mulawka, Molecular Genetic Programming, sent to *Journal of Soft Computing*, Springer.

[34] D. Rooß, K.W. Wagner, *On the power of DNA-computers*, Technical reports, University of Wurzburg (1995, 1996).

[35] T. Head, Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors, *Bulletin of Mathematical Biology* **49** (1987) 737–759.

[36] E. Csuhaj-Varjú, L. Kari, G. Pun, Test tube distributed systems based on splicing, *Computers and AI* **15(2–3)** (1996) 211–232.

[37] T. Head, G. Pun, D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, chapter 7 in vol. 2 of G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages* 3 volumes, Springer-Verlag, Heidelberg (1997).

[38] Z. Li, Algebraic properties of DNA operations, *Proc. of the Fourth DIMACS Workshop on DNA-based Computers*, Pennsylvania, USA (1998) 57–70.

[39] V. Gupta, S. Parthasarathy and M. Zaki, Arithmetic and Logic Operations with DNA, *Proc. of the Third DIMACS Workshop on DNA-based computers*, Philadelphia (1997) 212–220.

[40] F. Ausubel et al., K. Struhl, *Short Protocols in Molecular Biology. A Compendium of Methods from Current Protocols in Molecular Biology*, Third Edition, Wiley & Sons (1995).

[41] J. Sambrook, E.F. Fritsch, T. Maniatis, *Molecular Cloning. A Laboratory Manual.*, Second Edition, Cold Spring Harbor Laboratory Press (1989).

[42] N.N. Biswas, *Logic Design Theory*, Prentice-Hall International Editions, USA (1993).

[43] F.J. Hill and G.P. Peterson, *Switching Theory and Logical Design*, Wiley, New York (1974).

[44] R. Lipton, D. Boneh, C. Dunworth, *Making DNA Computers Error Resistant*, Princeton University (1996).

[45] K. Langohr, *Sources of Error in DNA Computation*, University of Western Ontario (1997).

[46] S. Roweis, E. Winfree, On the reduction of errors in DNA computation, *Journal of Computational Biology*, **6(1)** (1999) 65-75.

[47] J.J. Mulawka, P. Wąsiewicz, K. Piętak, Virus-enhanced genetic algorithms inspired by DNA computing, *Lect. Not. Art. Int. - Subseries LNCS* **1609** (1999) 527–537.

Table 4. Application of the algorithm - adding 11 and 13; each column describes a separate test tube. Increase or decrease of a column number means separation into different test tube or merging tube contents. Operations: *sep, mex, mex', mex", c, n* are described earlier.

1	$\left\lceil\begin{array}{cccccc} + & d & * & b & * & a \\ \tilde{d} & * & \tilde{b} & * & \tilde{a} & + \\ \hline \end{array}\right\rfloor$	$\left\lceil\begin{array}{cccccc} + & d & * & c & * & a \\ \tilde{d} & * & \tilde{c} & * & \tilde{a} & + \\ \hline \end{array}\right\rfloor$	
2	$\left\lceil\begin{array}{cccccc} + & d & + & b & + & a \\ \tilde{d} & + & \tilde{b} & + & \tilde{a} & + \\ \hline \end{array}\right\rfloor \overset{mex}{\Rightarrow} \{d, b, a\}$	$\left\lceil\begin{array}{cccccc} + & d & + & c & + & a \\ \tilde{d} & + & \tilde{c} & + & \tilde{a} & + \\ \hline \end{array}\right\rfloor \overset{mex'}{\Rightarrow} \{\tilde{d}, \tilde{c}, \tilde{a}\}$	
3	$\left\{\left\lceil\begin{array}{c} d \\ \tilde{d} \end{array}\right\rfloor, \tilde{c}, b, \left\lceil\begin{array}{c} a \\ \tilde{a} \end{array}\right\rfloor\right\} \overset{mex'',sep}{\Longrightarrow} \left\{\left\lceil\begin{array}{c} d \\ \tilde{d} \end{array}\right\rfloor, \left\lceil\begin{array}{c} a \\ \tilde{a} \end{array}\right\rfloor; b; \tilde{c}\right\}$		
4	b	$\tilde{c} \overset{n}{\Rightarrow} c$	$\left\{\left\lceil\begin{array}{c} d \\ \tilde{d} \end{array}\right\rfloor, \left\lceil\begin{array}{c} a \\ \tilde{a} \end{array}\right\rfloor\right\} \overset{mex}{\Rightarrow} \{d, a\}$
5	$\left\lceil\begin{array}{ccccccccc} & & & & c & + & b & & \\ \tilde{e} & * & \tilde{d} & * & \tilde{c} & * & \tilde{b} & * & \tilde{a} \\ \hline \end{array}\right\rfloor$	$\left(\begin{array}{cc} [d \quad]^{\$} & [a \quad]^{\$} \\ \underline{\tilde{d} * e} & \underline{\tilde{a} * b} \end{array}\right)$	
6	$\left\lceil\begin{array}{ccccccccc} & & & & c & * & b & & \\ \tilde{e} & * & \tilde{d} & * & \tilde{c} & * & \tilde{b} & * & \tilde{a} \\ \hline \end{array}\right\rfloor$	$\left(\begin{array}{cc} d * \tilde{e} & a * \tilde{b} \\ \underline{\tilde{d} * e} & \underline{\tilde{a} * b} \end{array}\right)$	
7	$c * b$	$\left(\begin{array}{cc} d + \tilde{e} & a + \tilde{b} \\ \underline{\tilde{d} * e} & \underline{\tilde{a} * b} \end{array}\right) \overset{mex'}{\Rightarrow} \{\tilde{e}, \tilde{b}\}$	
8	$\left\{\left\lceil\begin{array}{ccc} c & * & b \\ [\; & \tilde{b} & +]^{\$} \\ \hline \end{array}\right\rfloor, \tilde{e}\right\} \Rightarrow \left\{\left\lceil\begin{array}{cccc} - & c & * & b \\ \tilde{c}' & * & \tilde{b} & + \\ \hline \end{array}\right\rfloor, \tilde{e}\right\} \overset{sep,mex'}{\Longrightarrow} \left\{\tilde{b} * \tilde{c}'; \tilde{e}\right\}$		
9	$\tilde{e}$	$\left(\begin{array}{ccc} [\tilde{b} * \tilde{c}' \quad]^{\$} & & \\ & \underline{c * d} & \underline{b * \tilde{c}} \end{array}\right)$	
10	$\tilde{e}$	$\left(\begin{array}{ccc} \tilde{b} * \tilde{c} * d & & \\ & \underline{c * \tilde{d}} & \underline{b * \tilde{c}} \end{array}\right)$	
11	$\tilde{e}$	$\left(\begin{array}{ccc} \tilde{b} * \tilde{c} + d & & \\ & \underline{c * \tilde{d}} & \underline{b * \tilde{c}} \end{array}\right) \overset{mex}{\Rightarrow} d$	
12	$\tilde{e}$	$d \overset{c}{\Rightarrow} \tilde{d}$	
13	$\{\tilde{e}, \tilde{d}\}$		