Focus: Toward Soft Hardware

Soft Computing 5 (2001) 106–113 © Springer-Verlag 2001

Molecular genetic programming

P. Wąsiewicz, J. J. Mulawka

Abstract The paper addresses a new implementation of genetic programming by using molecular approach. Our method is based on dataflow techniques in DNA computing. After description of fundamental operations on DNA molecules and construction of logical functions the genetic programming method is introduced. We propose a way to handle these graph encoding molecules and which can be considered a genetic programming algorithm; a short discussion about experiments in implementing parts of this procedure is added.

Keywords DNA computing, Evolutionary programming, Genetic programming, Data flow computer

1 Introduction

Genetic programming [1, 2] has been recently developed as one of evolutionary algorithms. Their earlier implementations are called: genetic algorithms [3] and evolution strategies. On the whole, genetic programming is a methodology to solve problems by genetically breeding populations of computer programs. In such an approach for a particular problem sets of functions and terminals are created. An initial population of LISP-like expressions is a collection of random tree-like or graph-like compositions of fundamental functions and terminals. Each expression called also a program represents a possible solution to the problem and is evaluated against this problem. Genetic operators of selection and crossover are applied to create new populations of programs. Evolutionary process is continued until either a solution is found or a maximum number of generations is reached. Thus, while terminate condition is not true, the following cycle is performed: increase the number of generation; select population(t) from population($t-1$); recombine population(t), using crossover and mutation operators; evaluate population(t).

With the help of genetic programming, genetic operations on graph-like structures are performed. These structures can describe e.g. logical functions. Genetic operators like selection and crossover create new populations of graphs in order to find the most appropriate solutions. However, in traditional computers such programs are not performed in parallel yet. In the von Neumann machines instructions are written in the memory one after another except for those after instructions of jumps to the other memory parts. Instruction pointer aims usually at the ready to sending (to a computational unit – processor), next instruction. Multitasking, sometimes real-time and with parallel execution systems e.g. Unix and Linux use the very quick task exchange in the computational unit. Such unit can execute only one task. More privileged tasks are performed more often. Even after adding several thousands of processors there are still great problems with fully parallel execution of tasks. And some tasks are still performed in the sequences on appropriate computational units.

On the other hand, dataflow computers have fully parallel architectures. The execution of a program does not depend on the sequence of the following one after another instructions, but on the data flow. Sending the instruction to the processor depends on the availability of all its data arguments. Every ready instruction can be at once executed. Data is held directly in the instructions. Results of computation are sent to the next instructions as special tokens with data. This asynchronous, parallel execution without the shared memory nor the counter program enables the creation of massively parallel computers [4]. However, dataflow techniques are in the research stage.

In the dataflow unit a program is described in the structure of the directed graph. In recent years much attention has been devoted to DNA computing [5–16]. It is interesting to consider representation of the directed graphs by means of DNA molecules. In the next points we introduce a concept of molecular genetic programming.

2 Fundamental operations on DNA molecules

DNA computing [5, 7, 17, 18] is a new way of information techniques implementation performed in real time during chemical reactions. In such an approach a single molecule is treated as an independent processor. A molecule anneals to another one chosen almost at once even from several million molecules placed in a very small volume. Two appropriate molecules attract each other by chemical means like a key and a lock. With this methodology it is possible to construct an associative, quick memory vastly larger than a human brain [6].

In DNA computing a DNA string or so called oligo is represented by a sequence of four basic nucleotides and is usually described by letters A, T, G, C. It may exist as a

P. Wąsiewicz (✉), J. J. Mulawka
Institute of Electronic Systems,
Warsaw University of Technology, Nowowiejska 15/19,
00-665 Warsaw, Poland
e-mail: {pwas,jml}@ise.pw.edu.pl

This work was supported by the Polish State Committee for Scientific Research, through the KBN grant number 8T11F00816. This article was processed with the LATEX2ε macros package.

separate DNA fragment or within a longer one e.g. a string w may be denoted by a sequence: $5'AGTC3'$ or may exist within a longer string $z = 5'AGAAGTCCTA3'$. Up to now several DNA computing notation standards was worked out e.g. DNA-Pascal, splicing [10, 19–21] and other [11, 22–24]. Here we introduce our symbolic representation, which is useful for molecular genetic programming. Digits $5'$, $3'$ denoting orientation of a DNA string can be replaced with symbols $|\ \rangle$. The length of the string w is denoted by: $|w|$, and its value is equal to a number of symbols forming the string w e.g. $|AGTC|$ denotes a length of four basic symbols: nucleotides. Using exemplary strings we can write:

$$w = |w\rangle = |AGTC\rangle$$
$$x = |x\rangle = |TCAGTCTAG\rangle$$
$$z = |z\rangle = |AGAAGTCCTA\rangle \ \Leftrightarrow z = |AGA * w * CTA\rangle$$
$$s = \langle s| = \langle GATGACTGA|$$
$$|w| = 4, |z| = 10$$

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 w is described by a symbol $''$ in the upper, right index of the letter w: w'', and *a left part* of the string by a symbol $'$ in the upper, right index of the letter w: w'. 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 w is divided into four parts: w', w^{ii}, w^{iii}, w^{iv}.

A string complementary to w is described by the same letter, but with an added symbol tilde (˜) this means $\tilde{w}$. Two complementary strings w and $\tilde{w}$ create after hybridization a double stranded string $\hat{w}$ 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 the upper string or a single string, and a single string with an orientation $3' \rightarrow 5'$ should be underlined.

$$\hat{w} = \begin{matrix} |w\rangle \\ \langle\tilde{w}| \end{matrix} = \left\lceil \begin{matrix} w \\ \underline{\tilde{w}} \end{matrix} \right\rfloor$$

Operations on DNA oligos 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 heat the test tube reaction solution and denoted by symbol $\uparrow$.
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 $\uparrow$.
3. Cutting of a double DNA string into two parts is executed in DNA computing with the help of enzymes. This means that a given string z may be digested by the enzyme in the presence of a hybridized complementary to z (at least in the neighbourhood of a place to cut) string denoted by a letter v. The enzyme with an ordinal number equal to 5 cuts the string z together with the string v what is described below.

$$\left\lceil \begin{matrix} - & z & - \\ & \underline{v} & \end{matrix} \right\rfloor \Rightarrow \left\lceil \begin{matrix} - & z' & z'' & - \\ & \underline{v'} & \underline{v''} & \end{matrix} \right\rfloor$$
$$\Rightarrow \left\lceil \begin{matrix} - & z' & + & - & z'' & - \\ & \underline{v'} & - & + & \underline{v''} & \end{matrix} \right\rfloor$$
$$\Rightarrow \left\lceil \begin{matrix} - & z' & \\ & \underline{v'} & + \end{matrix} \right\rfloor + \left\lceil \begin{matrix} + & z'' & - \\ & \underline{v''} & \end{matrix} \right\rfloor$$

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. 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 w and x 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 y is a concatenation of the oligo complementary to the first string right part w'' and the oligo complementary to the second string left part x'.

$$y = \widetilde{w''} * \widetilde{x'} = \langle TCAGAGTC|$$

Thus concatenation of two strings w and x in the presence of the third one y is denoted by:

$$wx = w * x \overset{*}{=} w + x \overset{y}{=} \{w, x\} \text{ or}$$
$$w + x \overset{*}{\Rightarrow} w * x$$
$$wx = |AGTCTCAGTCTAG\rangle$$
$$|wx| = 13$$

In the given above quotation the symbol $+$ means approach of two DNA strings from the set $\{w, x\}$ with the help of the complementary to them and hybridized third one y written above the sign of equality. The formed double string enables concatenation of first two DNA fragments into longer one with the help of the ligation process. The symbol $*$ means in this case the concatenation operation and the symbols $\overset{*}{\Rightarrow}$ or $\overset{*}{=}$ its execution. The letters of joined strings after concatenation can be separated by the symbol $*$, but it is not necessary. The null string is the neutral element for concatenation this means $\varepsilon * w = w * \varepsilon = w$.

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.

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

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

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

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

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. They are usually added in great quantities to the test tube before amplification. Every amplification described above is done in one cycle between cooling (heat $\downarrow$) and heating (heat $\uparrow$).
6. Mixing of DNA fragments enables their uniform distribution. It improves search for good hybridizations in the space of all possible ones.

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^*.

3
Representing graphs by DNA molecules for evolutionary programming

In the dataflow computers genetic programming may be executed on the populations of graph-structures describing logical functions [9, 15]. Each logical function consists of three logical operations and their arguments v_i: product – conjunction (AND $\wedge$), sum – disjunctive conjunction (OR $\vee$), negation (NOT) [25]. The construction of the graph starts with creating nodes, which are related to function arguments plus one node – root – named x. Arcs reflect the layout of operations in the expression. The conjunction operation e.g. $a \wedge b$ adds to a node representing the argument a only one successor node representing b as is depicted in Fig. 1a. The disjunctive conjuction operation e.g. $a \vee b$ adds to the final node of the last operation (or x) two successor nodes representing arguments a and b as is described in Fig. 1b. If the predefined node x is connected by the operation AND only with one node a, it can be bypassed and replaced with the node a. It should be noted that some logical functions have degenerated structures e.g. a function $f = a \wedge a$ has its structure simplified to a and a function $f = a \vee a \wedge b$ simplified to $f = a \wedge b$. So the easiest to code are sums of function argument products. Thus, the function

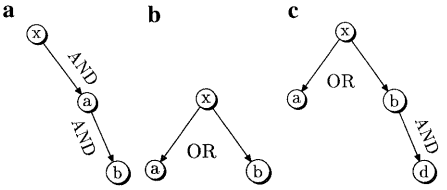

Fig. 1a–c. Exemplary graphs **a**) $f(a, b) = a \wedge b$, **b**) $f(a, b) = a \vee b$ and **c**) $f(a, b, d) = a \vee (b \wedge d)$; all with the predefined first node x

result is equal to TRUE when at least one leaf of the graph can be reached from the root through the successive nodes set only to TRUE.

Implementation of the simple, evolutionary algorithm on the graphs looks like this: for given arguments with values TRUE or FALSE one mixes the graphs performing the crossover operation are chosen. The best ones means such ones with their value after computing equal to a value of TRUE. Exchanging parts of cut arcs so called crossover is the main operation in the cycle of genetic programming. The finishing operation is checking whether the logical function has an appropriate structure, which after computing has a result value equal to TRUE.

In order to describe the mentioned method, the exemplary logical function is denoted by:

$$f : \{v_1, v_2, \ldots, v_i\} \Rightarrow x \ ,$$

where $v_1, v_2, \ldots, v_i$ for $i \in \{1, \ldots, M\}$ (M is a number of function arguments) and x can have one of two values: TRUE or FALSE.

The created logical function graph is the directed graph $\Phi = (V, E)$, where V is a set of function argument nodes. This means $V = \{v_1, v_2, \ldots, v_i\} = \{\langle x_i * z_i * y_i|\}$ for $i \in \{1, \ldots, N\}$ and $N = M$ (a repeating of an argument in the function does not increase a number of nodes in the graph), where N is a number of graph nodes, and M is a number of function arguments, and E is a set of ordered node pairs called *arcs*. The arc from a to b is denoted by $a \rightarrow b$. One can write $E = \{v_i \rightarrow v_j\} = \{\langle v_i'' * s_h * v_j'|\}$ for $i, j \in \{1, \ldots, N\}$ and $h \in \{1, \ldots, G\}$, where s_h is a restriction place for cutting enzymes and G is a number of such places. The path from v_1 to v_k in the graph is a sequence of nodes v_1, v_2, …, v_k, $k \geq 1$, such that every $v_i \rightarrow v_{i+1}$ is an arc for each $i \in \{1, \ldots, k\}$. If $a \rightarrow b$ is an arc, then a is a predecessor of b, and b is a successor of a. The mentioned path is denoted by $v_1 \cdot v_k$. First nodes has no predecessors, and final nodes has usually no successors, but in our examples some of them have successors.

As is seen, single strings representing graph nodes consist of three sectors from the following groups: $X = \{x_1, x_2, x_3, \ldots, x_M\}$, $Z = \{z_1, z_2, z_3, \ldots, z_M\}$, $Y = \{y_1, y_2, y_3, \ldots, y_M\}$. Single strings representing graph arcs are also made of three sectors from the following groups: $X = \{x_1, x_2, x_3, \ldots, x_M\}$, $S = \{s_1, s_2, s_3, \ldots, s_G\}$, $Y = \{y_1, y_2, y_3, \ldots, y_M\}$. If a node v_i is denoted by $\langle x_i * z_i * y_i|$, and his successor v_{i+1} is described by $\langle x_{i+1} * z_{i+1} * y_{i+1}|$, then the string of the arc between them $v_i \rightarrow v_{i+1}$ is denoted by $\langle y_i * s_h * x_{i+1}|$ or $\langle v_i'' * s_h * v_{i+1}'|$.

The above model of the logical function graph does not use the negation operation (NOT) by now. The $-x$ and x are treated as separated arguments and negative operations have to be transformed to normal operations

with negative arguments. A node representing a negative argument (a positive argument is called v_i) is denoted by $v_k = \overline{v_i}$ and is represented by a string $\langle x_k * \tilde{z}_i * y_k|$ for $k \neq i$. Only the sector $\tilde{z}_i$ is complementary to the sector z_i of the positive argument string v_i. The given method cannot calculate all Boolean functions and requires further research. It is true that every string in DNA computing is represented by a hard to count group of the same strings. Thus, it is impossible in practice to hybridize equal numbers of v_i and $\overline{v_i}$, but it is possible to destroy one sort of DNA strings, if one string group is smaller than another one and two strings after hybridization can be destroyed by enzymes as is depicted in Fig. 2.

After creating graphs one should synthesize DNA strings representing all function argument nodes (positive and negative) and the special node x:

$$x = \langle x| \ ,$$

$$v_i = \langle v_i| \text{ for } i \in \{1, \ldots, N_p\} \ ,$$

$$\overline{v_i} = \langle \overline{v_i}| \text{ for } i \in \{1, \ldots, N_n\} \ ,$$

where N_p is a number of positive function argument nodes, and N_n is a number of negative function argument nodes, and $N_p + N_n = N$; and all graph arcs (and all $|\widetilde{s_h}\rangle$):

$$x \to v_i = \langle x'' * s_h * v_i'| \ ,$$

$$v_i \to v_j = \langle v_i'' * s_h * v_j'|$$

$$\text{for } i, j \in \{1, \ldots, N\} \text{ and } h \in \{1, \ldots, G\} \ ,$$

$$\widetilde{s_h} = |\widetilde{s_h}\rangle \text{ for } h \in \{1, \ldots, G\} \ ,$$

where N is a number of graph nodes, and an additional DNA fragment $p = |\tilde{x}'\rangle$ is used in the amplification of paths in the graph.

As a result only the DNA string p is complementary to the string x. The remained strings are not complementary to each other and do not hybridize with each other. Of course, this fact depends on appropriate coding of DNA strings sequences [8, 15]. Exemplary experiments on synthesized DNA strings were implemented and described in [15].

4
Proposed method of genetic algorithm

The genetic programming algorithm cycle proceeds as follows:

1. Initiation
   - Ready for use arc strings together with complementary to them strings $\tilde{s}_h$ are added to test tube T_1.
2. Operation of cutting arc strings by enzymes is performed:

   for $k \in \{1, \ldots, L\}$ and $h \in \{1, \ldots, G\}$ do in parallel begin

   $$\text{heat} \downarrow; \left[\begin{array}{ccc} & {}_1\tilde{s}_h & \\ x_k & {}_{\tilde{1}}s_h & y_k \end{array}\right]$$

   $$\Rightarrow \left[\begin{array}{ccc} & \tilde{s}_h' & \\ x_k & s_h' & + \end{array}\right] + \left[\begin{array}{ccc} + & \tilde{s}_h'' & \\ & s_h'' & y_k \end{array}\right]; \text{ heat} \uparrow;$$

   end

   where L is a number of arc strings and G is a number of restriction places.
3. Concatenation of arc parts. New arcs are created and in the consequence new function structures:

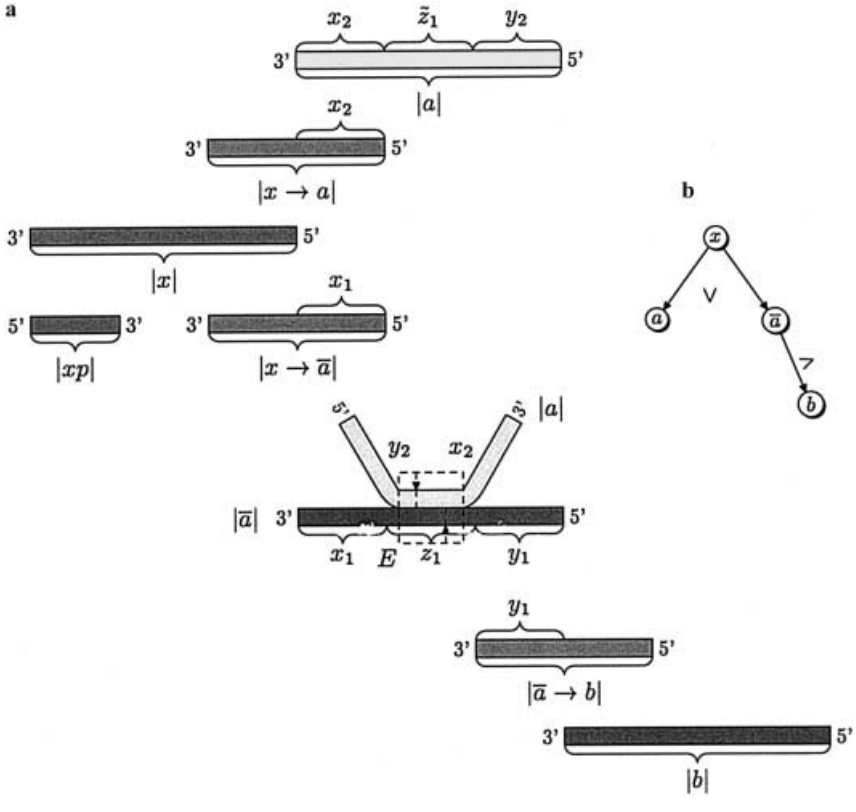

Fig. 2a, b. Implementation of the negation (NOT): **a**) two oligonucleotide sets (arc strings are here without sectors z_k), **b**) a logical function graph with $\overline{a}$; E – an enzyme restriction site

for $i \in \{1,\ldots,L\}$ and $j \in \{1,\ldots,L\}$
and $h \in \{1,\ldots,G\}$ do in parallel
begin

$$\text{heat}\downarrow;\ \left[\begin{array}{ccc} & \tilde{s}'_h & \\ x_i & s'_h & + \end{array}\right] + \left[\begin{array}{ccc} + & \tilde{s}''_h & \\ & s''_h & y_j \end{array}\right] \overset{*}{\Rightarrow} \left[\begin{array}{ccc} & {}_1\tilde{s}'_h * \tilde{s}''_h & \\ x_i & {}_{\bar{1}}s'_h * s''_h & y_j \end{array}\right];\ \text{heat}\uparrow;$$

end

where L is a number of arc strings and G is a number of restriction places.

4. Finding values of function results and estimation of function graph complexity. Thus, a certain quantity of DNA from the test tube T_1 is transfered to the test tube T_2 and the following operations are executed in T_2:
   - The string p and negative argument strings are added to the test tube T_2. System waits for input signal strings.
   - Input signals are strings representing arguments with values equal to TRUE (logical one), which are added to the test tube in this step point ($T_2 = T_2 \cup v_i$). The quantity of v_i must be larger than the quantity of $\overline{v_i}$.
   - Operation of cutting by an enzyme number 2 strings consisted of hybridized v_i with $\overline{v_i}$:

     for $i \in \{1,\ldots,N_p\}, k \in \{1,\ldots,N_n\}$ and $v_k = \overline{v_i}$ do in parallel
     begin

$$\text{heat}\downarrow;\ \left[\begin{array}{ccc} y_k & {}_1\tilde{z}_i & x_k \\ x_i & {}_{\bar{1}}z_i & y_i \end{array}\right] \Rightarrow \left[\begin{array}{ccc} y_k & \tilde{z}'_i & \\ x_i & z'_i & + \end{array}\right] + \left[\begin{array}{ccc} + & \tilde{z}''_i & x_k \\ & z''_i & y_i \end{array}\right];\ \text{heat}\uparrow;$$

     end

     where N_p is a number of arguments v_i with values equal to TRUE, N_n is a number of negative arguments $v_k = \overline{v_i}$ for $i \in \{1,\ldots,N_p\}$ and $k \in \{1,\ldots,N_n\}$, $N_p + N_n = N$.

   - for $w = 1$ to W do in parallel
     begin

$$\text{heat}\downarrow;\ \left[\begin{array}{cc} [p &]^{\$} \\ - & x_w \end{array}\right] \Rightarrow \left[\begin{array}{c} \tilde{x_w} \\ x_w \end{array}\right];\ \text{heat}\uparrow;$$

     for $i = 1$ to I do in parallel begin

$$\text{heat}\downarrow;\ \left[\begin{array}{ccc} - & \tilde{x_w} & \\ [& v_i & +]^{\$} \end{array}\right] \Rightarrow \left[\begin{array}{cc} \tilde{x_w} & + \\ - & x_w \cdot v_i \end{array}\right];\ \text{heat}\uparrow;$$

     end
     end

     where W is a number of logical functions in the genetic programming population, and I – a number of first node successors.
   - Cycles of amplification:

     for $i,j \in \{1,\ldots,N\}$ and $w \in \{1,\ldots,W\}$ do begin

$$\text{heat}\downarrow;\ \left[\begin{array}{cc} [p &]^{\$} \\ - & x_w \cdot v_i \end{array}\right] \Rightarrow \left[\begin{array}{c} \widetilde{x_w \cdot v_i} \\ x_w \cdot v_i \end{array}\right];\ \text{heat}\uparrow;$$

$$\text{heat}\downarrow;\ \left[\begin{array}{ccc} - & \widetilde{x_w \cdot v_i} & \\ [& v_i \to v_j & +] \end{array}\right] \Rightarrow \left[\begin{array}{ccc} \widetilde{x_w \cdot v_i} & & + \\ - & x_w \cdot v_i \to v_j & \end{array}\right];\ \text{heat}\uparrow;$$

$$\text{heat}\downarrow;\ \left[\begin{array}{cc} [p &]^{\$} \\ - & x_w \cdot v_i \to v_j \end{array}\right] \Rightarrow \left[\begin{array}{c} \widetilde{x_w \cdot v_i \to v_j} \\ x_w \cdot v_i \to v_j \end{array}\right];\ \text{heat}\uparrow;$$

$$\text{heat}\downarrow;\ \left[\begin{array}{ccc} - & \widetilde{x_w \cdot v_i \to v_j} & \\ [& v_j & +]^{\$} \end{array}\right] \Rightarrow \left[\begin{array}{cc} \widetilde{x_w \cdot v_i \to v_j} & + \\ - & x_w \cdot v_j \end{array}\right];\ \text{heat}\uparrow;$$

     end,

     where N is a number of graph nodes, and w is an ordinal number of logical functions: $w \in \{1,\ldots,W\}$, and v_j is a successor node of v_i and at the end of amplification there exists paths $x_w \cdot v_i$ and $x_w \cdot v_j$ from first nodes to final nodes in the graph.
   - Detection of the final results as the evaluation process. In the final DNA solution detection of the correct paths from first nodes to final nodes is performed. It is enough for a function to have only one such correct path in order to have a result value equal to TRUE. In addition new function graph structures are checked. Found values decide whether DNA computing algorithm should be terminated.
5. If termination is not executed, the next algorithm cycle begins from the step point number 2.

In every algorithm cycle (in T_2) there are four cycles of lengthening 3′ ends of single DNA strings. Every function graph has only one first node x_w. In the above mentioned algorithm there are involved W logical functions. System is so parallel that new function adding does not change the mentioned algorithm structure and its execution time. In parallel the same appropriate algorithms (each for one function) will be executed at the same time.

In Figs. 3–5 the process of the arc crossover in the graph $f = a \vee (b \wedge (c \vee d))$ is illustrated. Respective arc oligonucleotides are prepared for cutting by an enzyme as is depicted in Fig. 3a. Strings $\tilde{s}_i$ create double molecules with them enabling their cutting. After cutting by the

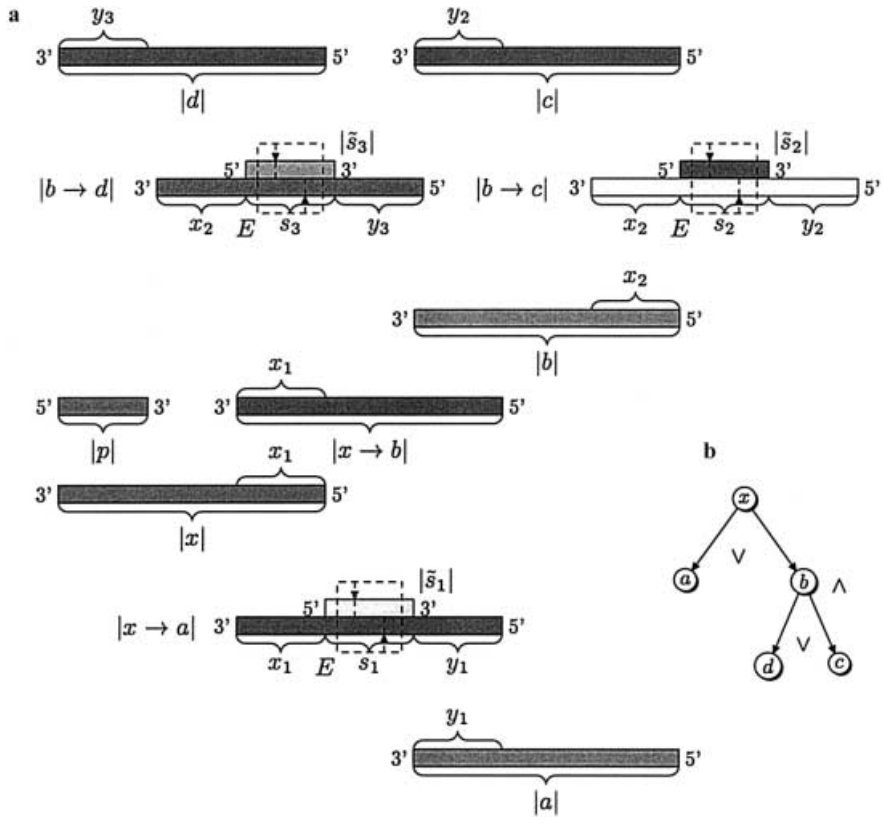

Fig. 3a, b. Implementation of the arc string part crossover – situation before cutting by an enzyme, **a**) oligonucleotides, **b**) a logical function graph $f = a \vee (b \wedge (c \vee d))$

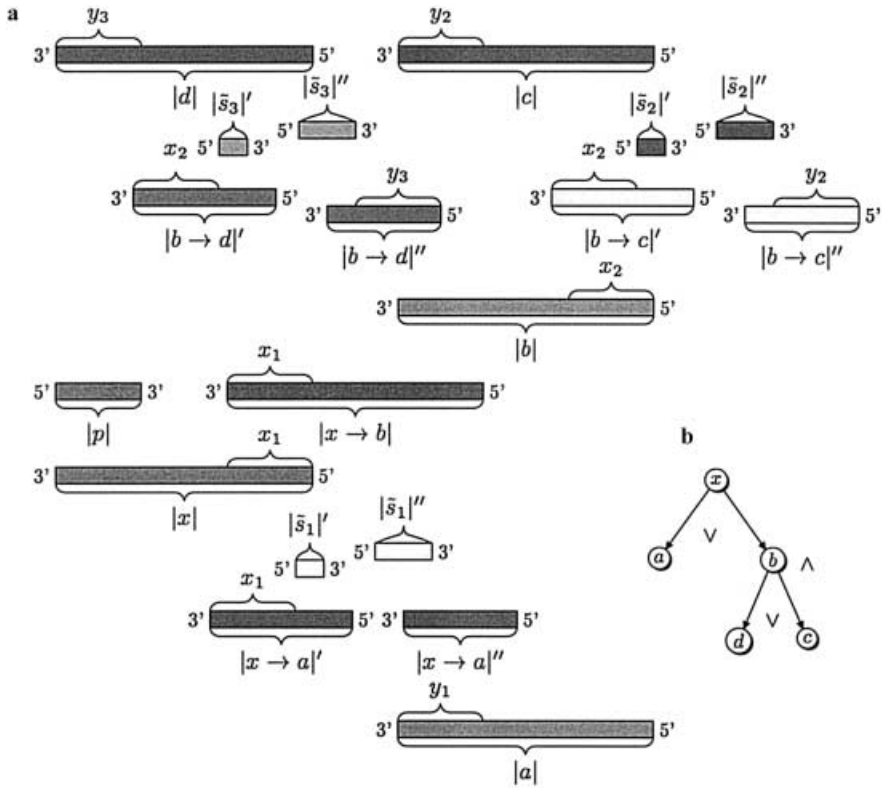

Fig. 4a, b. Implementation of the arc string part crossover – situation after cutting by an enzyme, **a**) oligonucleotides, **b**) a logical function graph $f = a \vee (b \wedge (c \vee d))$

enzyme the solution is heated. During this operation enzymes are destroyed and DNA molecules are denatured as is seen in Fig. 4a. All created strings are mixed in the test tube and during hybridization again anneal (Fig. 5a). In the given case after cutting, heating, mixing and hybridization 18 different, logical functions can be obtained. One algorithm cycle (4 amplification cycles) is enough to have all these functions in the test tube.

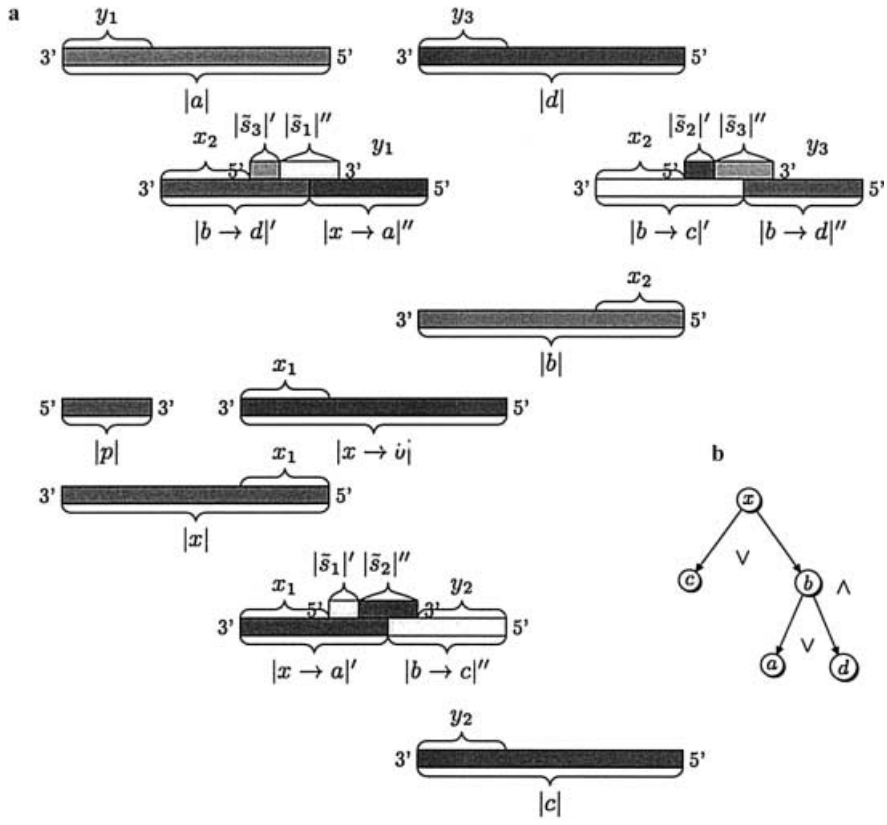

Fig. 5a, b. Implementation of the arc string part crossover – situation before concatenation **a**) oligonucleotides, **b**) a logical function graph $f = c \vee (b \wedge (a \vee d))$

5 Conclusion

In this paper a new method of genetic programming by means of DNA computing has been provided. The whole algorithm with the crossover and negation operations is considered and illustrated. Logical function arguments, which are treated as data, are input signals of the dataflow function graph consisted of special and arc strings. This dataflow machine structure depends on results of genetic programming operations and changes after each cycle. During execution of the algorithm the operation of amplification is performed only in one test tube. Identification of its products is performed by electrophoresis.

The approach provided may be useful in development of new computers [15, 16, 26]. These massively parallel operations of interacting molecules may be used in powerful parallel computers. In future dataflow computers special handshaking or token matching operations may be better implemented in molecular computers. One molecule – an operand – finds among million ones an appropriate molecule – an instruction – and both molecules form the larger one. Connected molecules generate – a new molecule – an output token, which is an operand of the next instruction in the dataflow graph program.

Several experiments were performed [15], but there are much more ones to design and proceed [8, 12]. It is also important to pay attention to new crossover operations e.g. between node parts after cutting by an enzyme. Further research in these fields is required.

References

1. **Koza JR** (1992) Genetic Programming, MIT Press, MA
2. **Wąsiewicz P, Mulawka JJ, Verma B** (1997) Global Optimization and Genetic Methods, Proc. of the 1st Int. Conf. on Computational Intelligence and Multimedia Applications (ICCIMA-97), Gold Coast, Australia, 10–12 II 30–36
3. **Goldberg DE** (1989) Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA
4. **Malone MS** (1995) Beyond semiconductors, The Microprocessor: A Biography, Springer-Verlag
5. **Adleman LM** (1994) Molecular computation of solutions to combinatorial problems, Science **266**: 1021–1024
6. **Baum EB** (1995) Building an associative memory vastly larger than the brain, Science **268**: 583–585
7. **Mulawka JJ, Borsuk P, Węgleński P** (1998) Implementation of the Inference Engine Based on Molecular Computing Technique, Proc. IEEE Int. Conf. Evolutionary Computation (ICEC'98), Anchorage 493–498
8. **Mulawka JJ, Wąsiewicz P, Piętak K** (1999) Virus-enhanced genetic algorithms inspired by DNA computing, Lect. Not. Art. Int. – Subseries LNCS **1609**: 527–537
9. **Mulawka JJ, Malinowski A** (1999) Calculation of Logic Functions by DNA Computing, Proc. Nat. Conf. On Evol. Alg. and Global Optim., Potok Zloty
10. **Păun G, Rozenberg G, Salomaa A** (1998) DNA Computing – New Computing Paradigms, Springer-Verlag Berlin Heidelberg
11. **Roweis S, Winfree E, Burgoyne R, Chelyapov N, Goodman M, Rothemund P, Adleman LM** (1998) A sticker-based model for DNA computation, J. Comput. Biol., **5(4)**: 615–629
12. **Roweis S, Winfree E** (1999) On the reduction of errors in DNA computation, J. Comput. Biol., **6(1)**: 65–75
13. **Wąsiewicz P, Borsuk P, Mulawka JJ, Węgleński P** (1999) Implementation of data flow logical operations via self-assembly of DNA, Lect. Not. Comp. Sci. **1586**: 174–182

14. **Wąsiewicz P, Janczak T, Mulawka JJ, Plucienniczak A** (1999) The Inference Via DNA Computing, Proc. Congress on Evolutionary Computation (CEC'99), **2**, VII, Washington, USA 988–993
15. **Wąsiewicz P, Malinowski A, Nowak R, Mulawka JJ, Borsuk P, Węgleński P, Plucienniczak A** 2001. DNA Computing: Implementation of Data Flow Logical Operations, accepted for Future Generation Computer Systems Journal, Elsevier vol. 17, Number 4
16. **Wąsiewicz P, Janczak T, Mulawka JJ, Plucienniczak A** 2000 The Inference Via Molecular Computing, accepted for Journal of Cybernetics and Systems, Taylor & Francis, **31:** 283–315
17. **Lipton RJ** (1995) DNA solution of hard computational problems, Science **268**: 542–545
18. The Bibliography of Molecular Computation and Splicing Systems, at http://liinwww.ira.uka.de/bibliography/Misc/dna.html
19. **Head T** (1987) Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors, Bulletin of Mathematical Biology **49**: 737–759
20. **Csuhaj-Varj E, Kari L, Păun G** (1996) Test tube distributed systems based on splicing, Computers and AI **15(2–3)**: 211–232
21. **Head T, Păun G, Pixton D** (1997) Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, chapter 7 in vol. 2 of Rozenberg G, Salomaa A (Eds), Handbook of Formal Languages 3 volumes, Springer–Verlag, Heidelberg
22. **Li Z** (1998) Algebraic properties of DNA operations, Proc. of the Fourth DIMACS Workshop on DNA-based Computers, Pennsylvania, USA, **V:** 57–70
23. **Gupta V, Parthasarathy S, Zaki M** (1997) Arithmetic and logic operations with DNA, Proc. of the Third DIMACS Workshop on DNA-based computers, Philadelphia, **VI:** 212–220
24. **Mihalache V** (1997) Prolog Approach to DNA Computing, Proc. of the IEEE International Conference on Evolutionary Computation (ICEC97), Special Session on DNA-based Computation
25. **Hill FJ, Peterson GP** (1974) Switching Theory and Logical Design, Wiley, New York
26. **Gehani A, Reif J** (1998) Micro Flow Bio-Molecular Computation, Proc. of the Fourth DIMACS Workshop on DNA-based Computers, Pennsylvania, USA, **V:** 253–266