On a Novel Evolutionary Based Model for Genome Rearrangement

Dmitry Korkin

The role of understanding the genome rearrangement is becoming more and more crucial in the modern life sciences. The new, alternative model explaining the process of genome rearrangement as well as the relationships between diverged gene-order sequences is proposed. The main characteristic feature of the model is its evolutionary nature: the gene-order sequences are viewed as the result of evolutionary genome development, that is, consecutive transformation of the ancestral genome via non-local genome mutations of two types, insertion and reversal. The close relationship of the above evolutionary-based mutations in the proposed model and "traditional" genome rearrangement mutations, such as, reversal and transposition, is discussed. A new, multiple longest common subsequence-based, method for extraction the evolutionary processes directly related to genome rearrangement is presented. Its advantages, disadvantages and future enhancements are outlined.

Genome rearrangement, evolutionary model, gene transformations, longest common subsequence, dominant points

I. Introduction

In this paper we give a short introduction to a new alternative model for genome rearrangement. We first give the basic definitions that will be necessary throughout the paper (Section II). Then, we introduce a new concept, the concept of an evolutionary genome graph, and reformulate the rearrangement problem with and without the help of the concept of evolutionary distance. Next, in Section III, we introduce a basic evolutionary-based model with reversals and insertions as non-local evolutionary mutations (transformations). We present the corresponding evolutionary genome graph and discuss its properties. Then, we discuss the relationship between evolutionary transformations and "traditional" genome rearrangement mutations. Next, we introduce an evolutionary distance for the above model. Finally in this section, we talk about context sensitive genome transformations and how they could make the above model more precise. In Section IV we outline the idea of constructing the evolutionary genome graph from a set of genomes. We present the basic pseudocode of the algorithm and analyze its complexity. We briefly explained the idea of dominant-point based algorithm for computing MLCS. In the last section we conclude with a brief overview of what was done in the paper and discussions about the future directions regarding the new model.

Dmitry Korkin is with the Faculty of Computer Science, University of New Brunswick, P.O.Box 4400, Fredericton, NB, Canada, E3B 5A3 (phone: 506-451-6931, e-mail: z17b3@unb.ca).

II. Basic definitions and problem formulation

In this chapter we introduce some basic definitions as well as formulate the problem of genome rearrangement/development reconstruction.

A. Basic definitions

Definition 1. Let $A = a_1,a_2, \ldots, a_n$ be a sequence of length n over alphabet $\Sigma = \{A,C,G,T\}$. A mapping $f: A \rightarrow B$, where $B = b_1,b_2, \ldots, b_n$ is a sequence of the same length n over Σ, such that $b_i = a_{\rho(i)}$, for a fixed permutation $\rho : [1..n] \rightarrow [1..n]$, is called a **gene order transformation** in genome A, or simply **g-transformation**.

Definition 2. A **rearrangement of genome** $G \in \Sigma^*$ is a sequence $f_1, f_2, \ldots, f_N$, where $N \geq 1$ and for each i, f_i is a g-transformation.

Definition 3. Let $A_1, A_2, \ldots, A_d$ be a set of d sequences of length $n_1, n_2, \ldots, n_d$, correspondingly, over alphabet $\Sigma = \{\sigma_1,\sigma_2, \ldots, \sigma_s\}$. If $A = s_1 s_2 \ldots s_n$, then $B = s_{i1} s_{i2} \ldots s_{ik}$ is a **subsequence** of A if $\forall j \in \{1,2, \ldots ,k\}$: $i_j \in \{1,2, \ldots ,n\}$ and $\forall s,t \in \{1,2, \ldots ,k\}$, $s<t: i_s<i_t$, where k is the length of B. The **multiple longest common subsequence** (MLCS) for a set of sequences $A_1, A_2, \ldots, A_d$ is a sequence B such that B is a subsequence of each A_i and it has the largest length. In the case of $d = 2$ MLCS is simply called the longest common subsequence (LCS).

Definition 4. Given a set of d genomes, $G_1, G_2, \ldots, G_d$, a MLCS of them is called the **longest gene pattern**, or simply LG-pattern.

Definition 5. Let *f* be a g-transformation of genome *A*, $A=a_1,a_2, \ldots, a_n$, and let $B = f(A)$, $B = b_1,b_2, \ldots, b_n$. The g-transformation *f* is called a **reversal** (of a subsequence $A_1=a_m, a_{m+1}, \ldots, a_{m+k}$) if the corresponding permutation ρ is such that for $k, m \geq 1$ (see Fig. 1):

(1) $b_i = a_i$, $1 \leq i \leq (k-1)$ or $(k+m+1) \leq i \leq n$;

(2) $b_{k+i} = a_{k+(m-i)}$, $0 \leq i \leq m$.

Genome A:

$\mathbf{a_1, \ldots a_{k-1}, a_k, a_{k+1}, \ldots, a_{k+m}, a_{k+m+1}, \ldots, a_n}$

Genome B:

$\mathbf{a_1, \ldots a_{k-1}, a_{k+m}, a_{k+(m-1)}, \ldots, a_k, a_{k+m+1}, \ldots, a_n}$

Fig. 1 A reversal of subsequence $A_1 = a_k, a_{k+1}, \ldots, a_{k+m}$. .

Definition 6. Let *f* be a g-transformation in genome *A*, $A=a_1, a_2, \ldots, a_n$, and let $B = f(A)$, $B = b_1, b_2, \ldots, b_n$. The g-transformation *f* is called a **transposition** (of a subsequence $A_1=a_m, a_{m+1}, \ldots, a_{m+k}$) if the corresponding permutation ρ is such that for $l, k, m \geq 1$ (see Fig. 2):

(1) $b_i = a_i$, $1 \leq i \leq l$;

(2) $b_{l+i} = a_{k+i}$, $1 \leq i \leq m$;

(3) $b_{l+m+i} = a_{l+i}$, $1 \leq i \leq (n-m-l)$,

$(l+m+i) \notin [l+1, l+m]$.

Genome A:

$\mathbf{a_1, \ldots a_{k-1}, a_k, a_{k+1}, \ldots, a_{k+m}, a_{k+m+1}, \ldots, a_n}$

Genome B:

$\mathbf{a_1, \ldots a_l, a_k, a_{k+1}, \ldots, a_{k+m}, a_{l+1}, \ldots, a_{n-l-m}}$

Fig. 2 A transposition of subsequence $A_1 = a_k, a_{k+1}, \ldots, a_{k+m}$.

Definition 7. A mapping $f: A \rightarrow B$, where $A = a_1,a_2, \ldots, a_n$, $B = b_1,b_2, \ldots, b_{n+m}$, $m>0$, is called **insertion transformation** (of sequence $c_1,c_2, \ldots, c_m$) in genome A or simply **ins-transformation** if $\exists k, 1 \leq k \leq n$, such that:

(1) $b_i = a_i$, $1 \leq i \leq k$

(2) $b_{k+i} = c_i$, $1 \leq i \leq m$

(3) $b_{i+m} = a_{i+k}$, $1 \leq i \leq n-k$.

Definition 8. A **development of genome** $G \in \Sigma^*$ is a sequence $f_1, f_2, \ldots, f_N$, where $N \geq 1$ and for each *i*, f_i is either a g-transformation or ins-transformation.

B. Evolutionary genome graph and problem formulation

Having defined the basic concepts, we next discuss the motivation for evolutionary approaches to the problem of genome rearrangement.

The evolutionary approach for genome rearrangement has the ideas similar to phylogenetic approach for the protein sequences. The phylogeny of genes has been studied for more than ten years by a number of scientists [1]-[5]. So, what might be the reasons to choose an evolutionary based approach when studying the genome rearrangement problem? First of all, if genome *B* was obtained from genome *A* as a result of a certain g-transformation *f*, this process can be naturally represented through the common ancestral, *C*, of *A* and *B*. In this case *A* and *B* can be viewed as obtained from genome *C* by applying to the latter g- or ins-transformations f_A and f_B, correspondingly. Second, one can also assume that, since all the species are the subject of *evolution*, the genomes of the close species are obtained by corresponding developments of genomes involving the *similar* g- and ins-transformations. Finally, the evolutionary approach allows one to study a genome rearrangement (and development) not of only those genomes that have same genes but also of those, whose genes might be different.

An example of the **evolutionary genome graph** (or simply **EG-graph**) is shown on Fig. 3. Generally, EG-graph is defined as a directed graph *DG (V, E)* with labeled vertices, *V*, whose labels correspond to current genomes, and the labeled edges, *E*, whose labels correspond to g- or ins-transformations. It's not hard to see that, in general, it may not be a tree: we can have two or more different paths from one vertex into another one, e.g., when performing the insertion of a sequence in a genome and, then, the reversal of another subsequence in the same genome or, when performing the reversal first and, then, the insertion in the above genome.

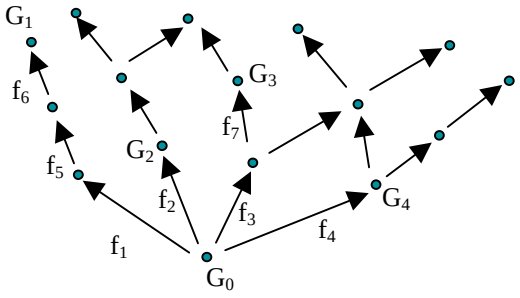

Fig. 3 An evolutionary genome graph

Thus, once defined a concept of EG-graph, we are able to formulate the basic problem of genome rearrangement/ development:

Problem 1. Given a set of genomes $\boldsymbol{G} = \{G_1, G_2, \ldots, G_n\}$, construct an EG-graph, $DG(V, E)$, such that $\forall G_i \ \exists v_j \in V$: v_j corresponds to G_i.

Next, in order to be able to compare genomes to each other, one needs to specify the similarity measure. In evolutionary approaches, one of the basic and most natural ways is to define such a measure trough the evolutionary paths, i.e. the paths from genomes to their closest common ancestor, where each of the paths is a series of transformations leading from ancestral to one of the current genomes.

Having specified a similarity measure on a set of genomes, one can come with the computationally much more complex problem:

Problem 2. Given a set of genomes $\boldsymbol{G} = \{G_1, G_2, \ldots, G_n\}$, and a similarity measure μ defined on $\boldsymbol{G}$, construct an EG-graph, $DG(V, E)$, such that $\forall G_i \ \exists v_j \in V$: v_j corresponds to G_i, and

$$\mu_A(DG) = \min_{DG'}(\rho_A(DG')),$$

where

$$\rho_A(DG) = (\sum_{(i<j)} \mu(G_i, G_j)) / n.$$

III. A NOVEL GENOME REARRANGEMENT MODEL

In this section we introduce a novel evolutionary-based model for genome rearrangement and development. First, we consider the case of two basic types of transformations, reversals and ins-transformations, and discuss their relationship. Then, we introduce a similarity measure based on the above transformations. Finally, we briefly discuss a new concept—the context of transformation—and show how it can be encapsulated into the model.

A. Basic evolutionary-based genome development and rearrangement model

Consider a set of genes $\boldsymbol{g} = \{g_1, g_2, \ldots, g_n \mid g_i \neq g_j, i \neq j\}$, that is, a set of strings over alphabet $\Sigma = \{A,C,G,T\}$. Then, a set $\boldsymbol{G}$ of genomes will be defined as

$$\boldsymbol{G} = \{g_{i1} \circ g_{i2} \circ \ldots \circ g_{ik} \mid i1, i2, \ldots ik \in \{1,2..n\}, \ k \leq n, \ i1 \neq i2 \neq \ldots \neq ik\}.$$

Next, we need to specify the type of the corresponding EG-graph. The EG-graph $DG(V, E)$ is specified as follows:

- There is one-to-one correspondence between the elements of V and the elements of $\boldsymbol{G}$;
- Each of the edges from E corresponds either to reversal or to insertion transformations;
- A reversal transformation cannot be applied twice in the same genome.

Note, that we use an additional restriction (the last item in the above specification) on the EG-graph: it allows us to avoid the case, where a reversal transformation can be applied infinitely many times in a genome. Moreover, the following Lemma is valid:

Lemma 1. With the condition defined above, $DG(V,E)$ is a directed acyclic graph.

The above Lemma gives us two related and very important properties: first, the genome development cannot ever return to one of the points it has already passed and, what follows from this, if one allows to copy (that is, insert) each of the genes only for a finite number of times (in the case of the above model – only once), then the process of constructing all possible genomes is finite.

The model discussed above has also the strong relation with the basic 'traditional' genome-level mutations. Namely, it is not hard to obtain the following results:

Lemma 2. Let A and B be two genomes, $A,B \in \boldsymbol{G}$, and let genome C be their closest ancestor. Then:

1) If $B=f(A)$, where f is a transposition, then there exist two insertions f_A, f_B, such that $A = f_A(C)$, $B = f_B(C)$;

2) If $B=f(A)$, where f is a reversal, then $A = C$.

In other words, genomes related to each other by either reversal or transposition, are also related via the closest ancestor with the help of corresponding g- or ins-transformations. This is partially explains the need for a evolutionary similarity measure of genomes to be based on transformations leading to these genomes.

Next we introduce a similarity measure for the model. In order to specify the similarity measure formally, we need to assign a weight to each of the reversal and insertion transformations used in the corresponding EG-graph. There are many ways to assign a weight to a genome-level mutation. On of the basic ways is to consider the size of the inserted, reversed, or transposed substring. In our model we will use this weight determination schema:

Definition 9. For any reversal f of a string a, the **weight** of this g-transformation is defined as $w(f) = |a|$, where $|a|$ is the length of the string a. For any insertion g of a string b, the weight of this ins-transformation is defined as $w(f) = |b|$, where $|b|$ is the length of the string b.

Having assigned the weights of transformations, now we can define one of the possible similarity measures.

Definition 10. Let A and B be two genomes, $A,B \in \boldsymbol{G}$, and let genome C be their closest ancestor. Let $A = f_n \circ f_{n-1} \circ \ldots \circ f_1(C)$, $B = g_m \circ g_{m-1} \circ \ldots \circ g_1(C)$ (Fig. 4). Then the **similarity measure** is defined as :

$$\mu(A,B) = \sum_{i=1}^{n} w(f_i) + \sum_{i=1}^{m} w(g_i),$$

where $w(f_i)$ and $w(g_i)$ are the corresponding weights of the transformations.

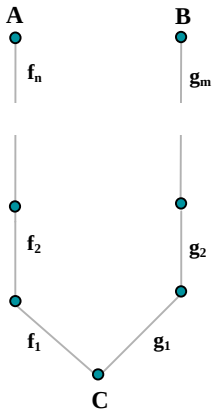

Fig. 4 The closest common ancestor of two genomes, *A* and *B*

Once specified the type of EG-graph, weighting schema, and the similarity measure, it becomes possible to define the relationships between genomes. Namely, given a set of genomes ***G***, we need, first, to compute a particular EG-graph, thus specifying the particular mutation pathways that would lead to each of the genomes in ***G***. Then, any genome in ***G*** can be compared with any other genome in ***G*** and their similarity measure, in terms of similar transformations occurring in the ancestor of the above genomes, can be calculated.

B. The case of context sensitive gene transformations

In this section we introduce the idea of using the context-sensitive genome-level mutations. It is natural to assume that the process of genome rearrangement depends on what constitutes the current genome being rearranged. In other words, the mutation of a genome can depend on a particular region of this genome (the case of a *local context*) or it can depend on the regions that are not close to each other and, even, on the entire structure of genome (the case of a *global context*). Since the issue of a global context-sensitive mutation is very complex, in this introductory paper we will discuss the local context-sensitive mutations only. We next consider the basic types of the local context for both, reversals and ins-transformations.

Definition 11. Given a set of genomes, ***G***, and a transformation f_X, where f is either reversal or ins-transformation, X is a subsequence, which is to be inversed/inserted, the **local context** of f_X is a pair of sequences, (C_1,C_2), $C_1,C_2 \in \Sigma^*$, such that:

1) If f_X is a reversal, and X' is the reversed image of X, then $\forall\, A,B \in \boldsymbol{G} \mid B = f_X(A)$:

$A = A'^{\circ}C_1^{\circ}\, X\, ^{\circ}C_2\, ^{\circ}A''$ and $B = B'^{\circ}C_1^{\circ}\, X'\, ^{\circ}C_2\, ^{\circ}B''$,

where $A', A'', B', B'' \in \Sigma^*$.

2) If f_X is an ins-transformation, then

$\forall\, A,B \in \boldsymbol{G} \mid B = f_X(A)$:

$A = A'^{\circ}C_1^{\circ}C_2\, ^{\circ}A''$ and $B = B'^{\circ}C_1^{\circ}\, X\, ^{\circ}C_2\, ^{\circ}B''$,

where $A', A'', B', B'' \in \Sigma^*$.

Note, that C_1 and/or C_2 can be empty. In a case when both, C_1 and C_2, are empty the corresponding transformation f_X is said to be **context-free**.

How would the concept of context be reflected on a similarity measure? There are many ways to redefine the similarity measure, such that the presence of a context would be taken into account. It goes without saying that the context-sensitive transformation when applying to genome would be more specific to the above genome, than the same transformation but without any context. Therefore, after applying a context-free operation f_X, the new genome $B = f_X(A)$ can be considered as a *farther* genome than the one obtained by applying the same transformation f_X but with non-empty context. In other words, in a view of the additive nature of the similarity measure, the weight of a context-free transformation should be larger than that one of the same transformation but with non-empty context. Below we give one of the possible ways to define such the weighting schema for context-sensitive transformations.

Definition 12. Let be a transformation and *(A,B)* be its context. Then, the **context-sensitive** weight of transformation f_X is defined as

$$w_{CS}(f_X) = w_{CF}\frac{|X|}{|X|+|A|+|B|},$$

where $w_{CS}(f_X)$ is a "standard" weighting schema of a (context-free) transformation defined in Def. 9.

The weighting schema in Def. 12 has two important features. First of all, for any context-free transformation f_X, its context-sensitive weight $w_{CS}(f_X)$ is equal to the "standard" weight of f_X, and the bigger the context of f_X, the smaller its context-sensitive weight. The latter, as was already mentioned above, can be explained by the fact that the bigger is the context of a transformation, the more specific is this transformation for a genome, in which it is applied, and thus, the resulting new genome should be closer to its ancestor than the one obtained by applying a context-free "version" of the same transformation f_X. Second, the smaller weight of context-sensitive transformation necessitates the search for the latter when solving the Problem 2 defined above, since the similarity measure between two genomes obtained by using context-sensitive transformations will be also smaller in comparison with using of the same transformation but without context.

IV. Implementation

In this section we discuss how to reconstruct an EG-graph, given a set of genomes. We consider a basic type of rearrangement that uses only transpositions. The EG-graph reconstruction algorithm described in this section use the idea of multiple longest common subsequence (MLCS) as well as its partial case – longest common subsequence (LCS). One of the most recent and promising improvements for the methods solving the MLCS problem—the idea of

dominant points—that makes our model very implementable, is briefly discussed at the end of the section.

A. Genome rearrangement using translocations only

1) Some basic ideas and assumptions

The basic and easiest in terms of computational complexity is the case when any genome from $\boldsymbol{G}$ can be obtained from any other by applying a finite set of transpositions of one or several genes from a set of genes $\boldsymbol{g}$. In terms of evolutionary models this means that we assume the same set of the genes (but in different order) to be inserted by applying the corresponding ins-transformations in the ancestral genome. This assumption results the following lemmas:

Lemma 3. Each of genomes consists of the same number of genes from $\boldsymbol{g}$.

*Lemma 4.*Let *DG(V,E)* be any EG-graph whose set of leaves corresponds to the set $\boldsymbol{G}$ (see section II *B*). Then, for any set $V_0 \subseteq V$, the MLCS of V_0 will contain at least one gene from $\boldsymbol{g}$.

The last lemma follows from the previous one and allows us to use successfully the idea of MLCS in the algorithm whose pseudocode is shown next.

2) Algorithm

The algorithm outputs one of the possible EG-graphs, *DG(E, V)*, given a set of genomes $\boldsymbol{G}$ consist of genes from a set $\boldsymbol{g}$. The pseudocode of the algorithm consists of two parts. In the first part the common ancestor of all genomes in $\boldsymbol{G}$ is built. In the second (main) part the transformations leading to each of genome are consecutively extracted. The pseudocode of the algorithm is presented below (note that we use notation //---// for comments).

Algorithm EG-graph

Input: $\boldsymbol{G}$, $|\boldsymbol{G}| = N$

Output: *DG(V,E)*, where $V = \{v_1, v_2, \dots, v_K\}$ is the set of labeled vertices, $E = \{e_1, e_2, \dots, e_M\}$ – the set of labeled edges.

1. Construct ancestor

Ancestor = v_0 = MLCS($\boldsymbol{G}$)

2. Construct transformations:

cur_level = V = $\{v_0\}$;

level = 0;

cur_$G_v(v_0)$={1, 2, … ,N};

// contains indices of all genomes//

While cur_level is not empty do

{

For all vertices *v* in cur_level

{

New_level = ∅;

// cur_G_v: indices of all current genomes corresponding to a current vertix from Cur_level //

cur_S = MLCS(cur_G_v);

cur_Tr = Transform(S);

// Transforms the MLCS cur_S to the sequence of ins-transformations, cur_Tr //

For each transformation *f* in cur_Tr

{

(V_1,E_1, cur_G_v) = Children_v (v, f, cur_ G_v, V, E);

// Returns new vertices (V_1) together with edges (E_1), connecting vertex v and new vertices, based on the transformation f and the set of genomes corresponding to vertex v, cur_G_v plus the set of indices Cur_G_v(w) corresponding to genomes that are children of the genome corresponding to a vertex w∈V_1//

V = V ∪ V_1;

E = E ∪ E_1;

} *// End For each transformation //*

New_level = New_level ∪ V_1;

// Adds vertices, corresponding to the last transformation in the sequence cur_Tr *to the new level //*

} *//End For all vertices v in cur_level //*

cur_level = new_level;

level = level+1;

} *//End While//*

END *//Algorithm//*

The algorithm's pseudocode presented above depends on two basic subroutines. While the first subroutine, MLCS() will be discussed in the section 5) below, the second one, Children_v(), can be described as follows. Given a vertex v_A corresponding to a genome A, the set of indices *cur_*G_v (*v*) of those genomes in $\boldsymbol{G}$, who have A as a common

ancestor, and an ins-transformation *f*, the subroutine Children_v() builds:

1) the set of vertices $V_1 = \{v_{A1}, v_{A2}, \ldots, v_{Ad}\}$ corresponding to the set $\{A_1, A_2, \ldots, A_d\}$ of immediate descendants of A obtained by applying *f* in A;
2) the set of edges E_1 , each of which is labeled by *f*, connecting each vertex in V_1 with v_A ;
3) for each of A_i — the set of indices $cur\_G_v(Ai)$ corresponding to those genomes in ***G***, who have A_i as a common ancestor.

3) Example

This example is rather illustrative than taken from the "real life". However, it reflects all the necessary aspects of the algorithm as well as allows comparing the results of "traditional" genome rearrangement with the evolutionary genome rearrangement represented by the corresponding EG-graph. Table 1 and Fig. 3 present the initial data and the resulting EG-graph, correspondingly.

TABLE 1
THE INITIAL DATA ***G*** CONSISTS OF 6 GENOMES OVER THE SET OF GENES
$G = \{A, B, C, D, E, F, G, H\}$

G_1 = D A B E C F G H;
G_2 = A D G B C F H E;
G_3 = A G B D C F E H;
G_4 = G A B D C F E H;
G_5 = E G C A F D B H;
G_6 = C E A F B G H D.

4) Complexity

In order to estimate the computational time complexity of the algorithm we use the following result.

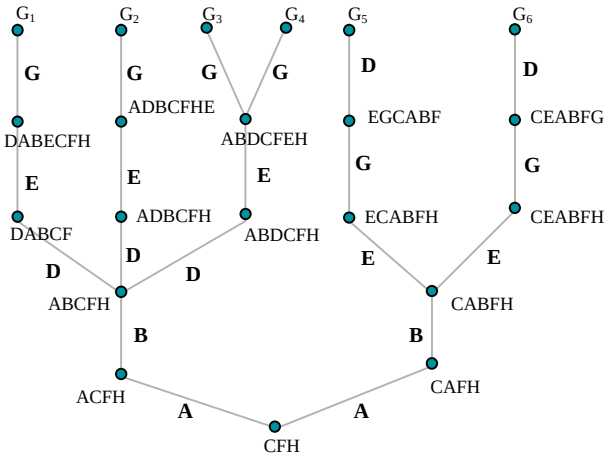

Fig. 5 An EG-graph for the set of genomes $\boldsymbol{G} = \{G_1, G_2, \ldots, G_6\}$ defined in the Table 1.

Lemma 5. Suppose $|\boldsymbol{G}| = d$, $|\boldsymbol{g}| = s$. Then:

1. In the algorithm one walks through the EG-graph only once;
2. For each level the MLCS algorithm can be performed no more than *d/2* times;
3. There are no more than s levels in EG-graph;
4. Transform() takes no more than *O(d*L)*, where $L = max\{|G_1|, |G_2|, \ldots, |G_d|\}$;
5. During the running of the main algorithm, Children() takes no more than *O(ds)*

Based on the above Lemma, it is not hard to find the time complexity of the algorithm given in 2). Namely, let O(T) be a time complexity for MLCS() subroutine. Then the following result can be obtained.

Theorem 1. The time complexity of the algorithm EG-graph is $O(s*((d/2)*T + d*L) + (d*s))$.

5) The extraction of genes from genome

So far we considered a problem where all genomes were represented as the sets of *given* genes. What if each of the genomes is given simply as a sequence over alphabet $\boldsymbol{\Sigma} = \{A,C,G,T\}$? Suppose all genomes in ***G*** satisfy the following two conditions:

1. $\forall\ G_i \in \boldsymbol{G}$: $|\ G_i\ | > L_{MIN}$, where L_{MIN} is a minimal length of a gene;
2. $\forall\ G_i, G_j \in \boldsymbol{G}$: $LCS(G_i, G_j) < L_{MIN}$, where LCS is a longest common subsequence of two sequences.

Then, using the slightly modified version of the algorithm described above, one can reconstruct genes from the given genomes sequences. The simplified pseudocode is represented below.

Algorithm EG-graph

Input: ***G*, | *G* |** = *N*

Output: ***g*** = $\{g_1, g_2, \ldots g_s\}$

---

cur_***G*** = ***G*; *g*** = ∅**;**
While cur_***G*** is not empty do
{ cur_S = MLCS(cur_G);
cur_ ***g*** = Transform(S);
*// Transforms the MLCS cur_S to the sequence of genes, cur_**g** //*
*Update(cur_**G**, cur_**g**)*
*//Removes the genes of cur_**g** from eachelement of cur_**G** //*
g = ***g*** ∪ *cur_**g***
} // End While //

As one can see, both of the algorithms described above use, as one of the key ideas, the idea of extracting the common patterns of genomes using multiple longest common subsequence method. Thus, the effectiveness of those algorithms will depend on how good is the method for finding MLCS, given a set of strings. The next section describes the most recent and promising method — the method of dominant points.

B. Dominant point-based method for multiple longest common subsequence

The method is based on several ideas. In order to describe these ideas, let us introduce some definitions and notations. Given a set of sequences, $A_1, A_2, \dots, A_d$, over alphabet $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_s\}$, the position p in the corresponding score matrix L is denoted as $p[p_1, p_2, \dots, p_d]$, where each p_i is a coordinate of p for the corresponding string, A_i. For a sequence A we denote a symbol corresponding to the k-th position in A as $A[k]$.

Definition 13. Position p in L is called *a match* iff
$$A_1[p_1] = A_2[p_2] = \dots = A_d[p_d].$$
A match p, corresponding to a symbol σ is denoted as $p(\sigma)$.

Definition 14. We say that point p *dominates* point q if $p_i \le q_i$ for all $i = 1,2, \dots, d$. We denote this fact as $p \le q$. The relation $p < q$ can be defined similarly.

Definition 15. A match p is called a *k-dominant* iff
$$L[p] = L[p_1, p_2, \dots, p_d] = k.$$
The set of all k-dominants for a point p is denoted as $D^k(p)$. The set of all k-dominants is denoted as $D(p)$.

Definition 16. A match $p(\sigma)$ is called a σ-*parent* of a point q iff $q < p$ and there is no match $r(\sigma)$ such that $q < r < p$. The set of all σ-parents of q is denoted as $Par(q, \sigma)$.

Definition 17. A point p in a set of points S *is* a *minimal* element of S, if $\forall q \notin S: q \le p$.

There are several main ideas leading to the above method. First, it is not hard to see that one should search among only matches since each position in a MLCS should at least be a match. Second, it can be shown that the 'special' points in the lowest-cost path corresponding to positions in the MLCS, which were discussed in the previous section, are k-dominants, $k = 1, \dots, |MLCS|$. Finally, for each k, $k = 1, \dots, |LCS|$, only minimal points of D^k can be the candidates for positions in the MLCS (see Table 4). Based on these ideas and some more advanced properties of dominant points properties Hakata and Imai developed an algorithm for computing a MLCS of a set of d sequences.

TABLE 2
THE SET OF DOMINANTS AND MATCHES IN THE SCORE MATRIX FOR TWO SEQUENCES, A = 'AABCAABCAB ' AND B = 'BACBAB'. THE DOMINANT POSITIONS ARE CIRCLED WHILE THE REMAINING MATCHES THAT ARE NOT DOMINANT ARE SQUARED

		a	a	b	c	a	a	b	c	a	b
	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	①	1	1	1	1	1	1	1
a	0	①	1	1	1	②	2	2	2	2	2
c	0	1	1	1	②	2	2	2	③	3	3
b	0	1	1	②	2	2	2	③	3	3	④
a	0	1	②	2	2	③	3	3	3	④	4
b	0	1	2	③	3	3	3	④	4	4	⑤

The theoretical time complexity for the above MLCS algorithm is given by the following theorem.

Theorem 2 [10]. The MLCS problem for $d \ge 3$ strings of length n can be solved in time
$$O(nsd + |D|sd\,(\log^{d-3} n + \log^{d-2} s)),$$
where $|D|$ is the size of the set of all dominant positions.

It is not hard to see that the size $|D|$ of the set of all dominants is much less than the set of all positions. Although a nontrivial (rather than by n^d) estimation of $|D|$ is still an open question the results obtained by the implementation of dominant point-based approach show the great advantage of this method in contrast to classical dynamic programming approaches.

V. DISCUSSIONS AND FUTURE RESEARCH

In this paper we introduced an alternative, evolutionary-based, way to study genome rearrangement. The development of genomes can be represented via special directed graph, called EG-graph, with labeled edges and vertices. One of the main advantages of the evolutionary method is that it allows reconstructing not only binary relations between genomes: e.g., EG-graph can represent three genomes having one common ancestor. In order to represent such relationships, the idea of a multiple longest common subsequence was used. Although "classical" dynamic programming methods for computing MLCS, practically, can be used only for the case of two, maximum three strings, the dominant points-based method allows one to get a MLCS for a much larger set of genome sequences.

There are many directions for the future research regarding this model. First of all, the next step can be the reconstruction of EG-graph when the models of genome rearrangement are more complex, e.g. models based on the following genome-level mutations:

1) *transpositions and reversals of genes;*
2) *transpositions and insertions of genes.*

Each of the above two cases is not trivial and needs careful studying.

Next, there is a computationally more complex problem of reconstructing an optimal EG-graph (see Problem 2 in section II.B.).

Another problem is the reconstruction of context-sensitive transformations. One of the possible solutions, if the optimal EG-graph is not necessary, is to search for the context of transformations after EG-graph with context-free transformations is built.

Finally, one can consider a model of genome rearrangement with the presence of a noise, that is, point mutations occurring when performing genome-level mutations. The weighting schema should take this fact into consideration, in such a way that the presence of some point mutations would affect the change of similarity measure between two genome sequences.

VI. REFERENCE

[1] S. Hannenhalli and C. Chappey and E. Koonin and P. Pevzner: Genome Sequence Comparison and Scenarios for Gene Rearrangements: A Test Case. *Genomics*, 30: 299-311, 1995.

[2] Sankoff, D., G. Sundaram and J. Kececioglu. ‘‘Steiner points in the space of genome rearrangements.” *International Journal of Foundations of Computer Science* 7:1, 1-9, 1996.

[3] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B. F. Land, and R. Cedergren. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences*, USA, 89:6575-6579, July 1992.

[4] David Sankoff, Mathieu Blanchette: Multiple Genome Rearrangement and Breakpoint Phylogeny. *Journal of Computational Biology* 5(3): 555-570, 1998.

[5] David Sankoff, Mathieu Blanchette: Probability models for genome rearrangement and linear invariants for phylogenetic inference. *RECOMB* : 302-309, 1999.

[6] Koji Hakata and Hiroshi Imai: Algorithms for the Longest Common Subsequence Problem for Multiple Strings Based on Geometric Maxima. *Optimization Methods and Software*, Vol.10, pp.233-260, 1998.

[7] K. Hakata and H. Imai, “The Longest Common Subsequence Problem for Small Alphabet Size between Many Strings,” *Proceedings of the 3rd International Symposium on Algorithms and Computation*, Nagoya, Lecture Notes in Computer Science Vol.650, pp.469-478, December 1992.