On the Metaheuristics Approach to the Problem of Genetic Sequence Comparison and Its Parallel Implementation

Sergey Makarkin¹, Boris Melnikov², Alexander Panin³

¹Department of Mathematics and Information Science, Togliatti State University, Togliatti, Russia
²Togliatti Branch of Samara State University, Togliatti, Russia
³Yandex Company, Moscow, Russia

Email: S.Makarkin@gmail.com, bormel@rambler.ru, AG.Panin@gmail.com

Received August 2, 2013; revised September 2, 2013; accepted September 9, 2013

ABSTRACT

We describe parallel implementation of the metaheuristic approach to the problem of comparing strings representing DNA sequence. By this approach, one can define a whole new class of metrics on a set of strings; some of this metrics can lead to interesting results when used for string comparison. We propose several heuristics; compare results achieved when using those heuristics and compare parallel and sequential implementation of proposed approach.

Keywords: Metaheuristic Approach; Genetic Sequence Analysis; Levenshtein Distance; Strings Alignment; Parallel Programming

1. Introduction

Determining DNA likeness is a particular case of a more common task of approximate comparison of strings, although called fuzzy comparison [1]. “Fuzzy” means here, which we should be able to determine similar sequences even if there are some errors and distortions, like insertion or deletion of several symbols. The amount of such distortions can be used as a metrics on a set of strings, defined as the minimum number of edits needed to transform one string into the other. This task can be found in many areas, like comparison of genes, chromosomes and proteins, which are one of the most important problems and at the same time one of the basic tools in molecular biology and bioinformatics [1,2]. Strict comparison of chains of nucleotides is unacceptable because of errors in data and possibility of mutations. Although the fuzzy comparison is used in text processing, Levenshtein metrics is used for error correction, for improvement of text recognition quality and in database search [1].

There are several approaches used for string comparison. Algorithms based on dynamic programming (like Hunt-Shimansky, Khirshberg, Wagner-Fisher and other) can provide the exact solution ([1,3-6]). Mostly, such algorithms have quadratic worst-case complexity and are considered too slow when speed is more important than accuracy (like database search). There are many approximate algorithms designed especially for some field of science, like BLAST [7] algorithm, designed for genetic database search.

Metaheuristic approach to fuzzy string comparison was firstly described in [8]. The strings likelihood assertion made by this approach is close to Levenshtein distance, but it is not equal to, nor is an approximation of Levenshtein distance. This algorithm allows using multiple alternative metrics on space of chains of nucleotides, depending on used heuristics. This metrics can reflect the likelihood of compared strings, although they show different meanings of likelihood itself.

2. Implementing Metaheuristic Approach

Meta-heuristic approach to discrete optimization problems uses branch and bound algorithm combined with several different heuristics that are used for next step selection. Heuristics assessments are averaged using dynamical risk functions. Genetic algorithms are used to fit the averaging ratios, the same genetic algorithms with simplified self-learning are used for branch and bound start-up ([9,10]).

For this particular problem we did the following. Let x, y be the source strings, i, j be indexes of symbol in x and y, respectively, r be metrics value. By shift of a string we mean increasing appropriate index by 1. The algorithm is described by the following:

Input: Strings x and y.
3. Using Multiple Greedy Heuristics

Now let us consider heuristics used to select the element that separates the problem into right and left sub-problems for branch and bounds method. Solving a discrete optimization problem by branch and bounds method, it is desirable to choose separating algorithms depending on the solved sub-problem. Separating algorithms can be selected based on dimension of the solved problem, its bound, and by taking into account some specific characteristics of considered problem.

In classical examples of branch and bounds method for travelling salesman problem ([12] etc.), some good separating algorithms were used (by “good” we mean that they perform better than other ones). However, long before [12], various other heuristics were used for the branching, see, e.g., [13] Let us mention, for example, the following heuristics for the reduced TSP-matrix: total number of zeroes, sum of minimums for all the rows and columns, sum of some minimum values of considered row and columns multiplied by special “dam-nation constants”; all these values are computed by the TSP-matrix after reducing and selecting separating element (i.e., separating edge for branching). Probably we mentioned here less than 10% of the heuristics used before.

Thus, how can we use the fact that in different situations (i.e., in different sub-problems of the same discrete optimization problem) different heuristics relatively perform better? (This question is true for both exact and unfinished algorithms). We need decide which separating element to use for branching. We have information from various experts, i.e., of various special heuristics, so called predictors (or estimators). The predictors often give discrepant information, and we have to average it in some special way. We use an approach that is used in nondeterministic games programming: dynamic risk functions.

Since different heuristics return values of different measurement units, we have to normalize them for computing the final result. For that purpose one can use a special set of normalizing coefficients. The other possible solution is a modification of “voting method”: use special dynamic risk functions for the results of voting. It is important to note, that the dynamic selection of the particular risk function is similar to selecting it in nondeterministic games programming ([14,15]). Since we consider here discrete optimization problems (not nondeterministic games programming), we have to add here new heuristics, i.e. heuristics for selecting “current position estimation”, in other words, for evaluation of the situation obtained by the solving some discrete optimization problem using branch and bounds method.

Thus, let us have some various heuristics for selecting next step element of branch and bounds method (or, gen-
eraly speaking, for selecting the strategy of solving). Let each of possible strategies have various expert evaluations of availability (i.e., let us have some independent expert sub-algorithms, so called predictors). Then the concluding strategy could be chosen by maximum of average values. However, let us consider the following example; this example is connected with backgammon programming, because it uses 36 predictors.

Let expert evaluations of availability lie in segment [0,1]. Let the 1st expert evaluation of availability for the 1st strategy be equal to 1, and evaluations of 35 other experts be equal to 0.055. And for the 2nd strategy, experts have evaluation equal to 0.95, and other 34 experts have evaluation equal to 0. It is very likely that each human expert in such case will choose 2nd strategy. However, averaging-out by the simplest algorithm (i.e. simple average of expert evaluations) gives 0.081 for the 1st case and 0.053 for the 2nd one; so do we have to choose the 1st strategy?

On the other hand, we can use an approach similar to [15], i.e., use the same algorithms for dynamic risk function construction. For the 1st strategy, we obtain the following risk function:

$$-0.685 \cdot x^2 + 1.300 \cdot x + 0.386;$$

and for the 2nd strategy:

$$-0.694 \cdot x^2 + 1.374 \cdot x + 0.321.$$ 

The final values of expert evaluations averaging-out by using these risk functions are 0.111 for the 1st strategy and 0.147 for the 2nd strategy. Therefore, using such algorithms for dynamic risk function construction for expert evaluations averaging-out gives “natural” answers.

Note that repeating the averaging procedure twice (i.e., averaging-out using preliminary values of the first step of dynamic risk functions) chooses 1st strategy. However, in the limit we have “natural” answers again. This can be seen in Table 1; the column headers are equal to the number of step of averaging-out using dynamic risk function (i.e., the number of iterations ran by a dynamic risk function constructing algorithm). The column 0 is the simple average of expert evaluations, and the column $\infty$ is the limit value.

Note that in real discrete optimization problems such situations, when the difference between minimum and maximum values is more than 0.5 (i.e., more than 50% of the segment of values) are very often; for example, for accidental TSP having dimension 75 and some of predictors mentioned before, they contain, by statistics of the author, about 10%.

### 4. Approach to a Parallel Algorithm

Main problem in parallel algorithm design is finding informational dependencies and independent sub-tasks.

The proposed algorithm is based on iterative shift of strings in some position, defined by two indexes $(i,j)$ for first and second string respectively. To define next step we search in some small set of positions, closest to current one, and then we select the position with the highest score. Obviously, such shifts can be made only sequentially, and the only part of an algorithm that has independent subtasks is a usage of heuristics for position selection. There are two possible ways for parallelizing this subtask: applying each heuristics in separate thread, and using parallel algorithms for heuristics themselves.

Using separate threads for each heuristics has the following disadvantages. First, the final result of using several heuristics is not always their “linear sum”, sometimes several heuristics are combined and considered as one complex heuristics. Second, some heuristics have relatively low computational complexity, and thread management costs become too significant.

Parallelizing heuristics although has disadvantages. Some heuristics being parallelized work slower, then their serial analogues, because of high overhead costs (like mentioned above thread management costs).

Thereby selecting independent subtasks seems not to be effective enough for practical use. But this does not mean that it is impossible to create effective parallel algorithm, because one can define some new subtasks. Thereto at each step we can select multiple possible shift positions, creating several “phase trajectories” in shift position space. There are several ways to do that, e.g. use several positions with highest score or use some ad hoc heuristics. Subtasks created like this can have a fixed size (i.e. iterations of basic shift cycle). After processing subtasks we can select several best results and generate new subtasks based on them.

Parallel implementation of meta-heuristic approach described above is a new algorithm, and introduced method of subtasks creation is basically, a new heuristics. This new parallel algorithm has better results than the sequential one, it is more precise. While the initial algorithm uses greedy heuristics to select next shift position, the introduced algorithm make deeper analysis of consequences of next step selection.

### 5. Results of DNA Comparison

To test our algorithm, we used mitochondrial DNA of different organisms. Those DNA molecules containing in cell’s mitochondria are not recombined, and they are inherited from mother’s organism by most of multicellular

Table 1. Risk function values for different strategies.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>$\ldots$</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st strategy</td>
<td>0.081</td>
<td>0.111</td>
<td>0.104</td>
<td>0.106</td>
<td>0.105</td>
<td>0.105</td>
<td>$\ldots$</td>
<td>0.105</td>
</tr>
<tr>
<td>2nd strategy</td>
<td>0.053</td>
<td>0.147</td>
<td>0.094</td>
<td>0.118</td>
<td>0.106</td>
<td>0.112</td>
<td>$\ldots$</td>
<td>0.110</td>
</tr>
</tbody>
</table>

Copyright © 2013 SciRes.
organisms. Therefore they can be changed only due to mutations. By analyzing mitochondrial DNA and its mutations, one can determine not only the degree of kinship of two species, but although a time, needed to accumulate some mutations in population [16]. Thereby one can estimate a moment of time when there were no mutations and population was genetically homogeneous.

We used genetic data for next species: Homo sapiens (human), Pan troglodytes (chimpanzee), Bison bison (bison), Bos taurus (wild bull), Sus scrofa taiwanensis (pig), Canis lupus (wolf), Felis catus (domestic cat), Gallus gallus (chicken), Mus musculus (mouse), Rattus norvegicus (rat), Orcinus orca (killer whale), Orcaella brevirostris (Irrawaddy dolphin), Peponocephala electra (melon-headed whale), Gadus morhua (Atlantic cod), Drosophila simulans (see [17]). An example of DNA comparison resulted in a parallel implementation of the seventh heuristics (that is a combination of third and sixth one) showed results pretty similar to Needlemah-Wulsch algorithm. For big values (more than 0.9) seventh heuristics results differing from Needleman-Wulsch were not more than 1%.

Results shown by parallel version of algorithm showed the same results, but shifted to a high values range. Parallel implementation of fifth heuristics performed better than sequential one, and was closer to results showed by the longest common subsequence search algorithm. On the contrary, results of parallel implementation of the seventh heuristic performed poor, and showed obviously wrong results for some DNA pairs. In [10] we had already covered an approach to parallelizing branch-and-bounds algorithms, and here we used the same approach with heuristics. The results of that research contain lots of data and can’t be presented here, so it will be described in a separate paper.

For most of heuristics the results of DNA comparison correspond to biological taxonomy [18]. At the same time the proposed algorithms are pretty fast, because the run time is in linear dependence with input length, but depends on heuristics parameters. This makes the proposed algorithms applicable for DNA data analysis.

### Table 2. Results of comparison Peponocephala electra with other species.

<table>
<thead>
<tr>
<th>Heuristics #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td><strong>Bison bison</strong></td>
</tr>
<tr>
<td><strong>Bos taurus</strong></td>
</tr>
<tr>
<td><strong>Canis lupus</strong></td>
</tr>
<tr>
<td><strong>Drosophila simulans</strong></td>
</tr>
<tr>
<td><strong>Felis catus</strong></td>
</tr>
<tr>
<td><strong>Gadus morhua</strong></td>
</tr>
<tr>
<td><strong>Gallus gallus</strong></td>
</tr>
<tr>
<td><strong>Homo sapiens</strong></td>
</tr>
<tr>
<td><strong>Mus musculus</strong></td>
</tr>
<tr>
<td><strong>Orcinus orca</strong></td>
</tr>
<tr>
<td><strong>Orcaella brevirostris</strong></td>
</tr>
<tr>
<td><strong>Pan troglodytes</strong></td>
</tr>
<tr>
<td><strong>Peponocephala electra</strong></td>
</tr>
<tr>
<td><strong>Rattus norvegicus</strong></td>
</tr>
<tr>
<td><strong>Sus scrofa taiwanensis</strong></td>
</tr>
</tbody>
</table>

### REFERENCES


