Metabolomics: Open Access

Unay and Guzey, Metabolomics 2013, 3:2
DOI: 10.4172/2153-0769.1000120

Conference Proceedings Open Access

A Swarm Intelligence Heuristic Approach to Longest Common Subsequence Problem for Arbitrary Number of Sequences

Ali Teoman Unay[1] and Meral Guzey[2]

[1]Department of Intelligent Computing Systems, İzmir University of Economics, İzmir, Turkey
[2]Department of Math and Life Sciences, Main campus of University Maryland University College (UMUC), USA

Abstract

Personalized cancer care strategies involving sequencing requires accuracy. We aimed to develop a novel approach to solve the longest common subsequence problem, which is a common computer science problem in the field of bioinformatics to facilitate the next generation sequencing of cancer biomarkers. We are using particle swarm optimization heuristic technique, which uses a novel "Occurrence Listing" (OL) technique as the evaluation function. This aims to keep lists of the sequence elements and offers criteria to evaluate randomly generated population of sequences.

Introduction

In the clinical diagnosis of cancer, the corresponding biomarker methods and measurements are based on the correct algorithms, which are used for sequencing. The necessity of development new or possibly re-establishing previous biomarkers in the field of cancer research initiated us to work on new sequencing techniques.

Here, we describe "Longest Common Subsequence Problem" (LCSP) with Particle Swarm Optimization (PSO), with the proposal of novel "Occurrence Listing" (OL) technique as an evaluation function. Previous studies shows, that despite the wide differences between popular approaches, like dynamic programming or other heuristic methods, even there are some variations of dynamic programming for three or more sequences [1], they generally work on two inputs (sequences) [2,3]. The benefits of our system are as follows: First, one can work with minimum two or more sequences. Second, one has flexibility of working with arbitrary number of sequences.

Longest common subsequence problem

LCSP dwells on longest common subsequence of two or more sequences.

Although general case of a random number of input sequences, the problem is NP-Hard, Dynamic programming can manage to solve the problem on polynomial time provided that the number of is constant [4].

What is "subsequence"?

A subsequence is a sequence that is created from another sequence by excluding some elements but without changing the initial order. For example, <A,B,D> sequence derived from <A,B,C,D,E,F> by omitting element C, E and F.

Given two sequences X and Y, sequence G can be defined as a common subsequence of X and Y, if G can be derived from both X and Y individually.

For example,

if X=<A,B,C,D,E,G,C,E,D,B,G> and

Y=<B,E,G,C,F,E,U,B,K>

then a common subsequence of X and Y could be

G=<B,E,E>

This would not be the longest common subsequence. The longest common subsequence of X and Y is <B,E,G,C,E,B>.

Particle swarm optimization

In the field of computer science, PSO is a method that aims to optimize a problem by trying to improve a possible solution with iteration in limitation and manipulation of a pre-defined quality measure. PSO processes an initial population of possible solutions, dubbed particles in this case, and changing the position of these particles in the search-space according to mathematical formula which is consisting of the particles' (a) position and (b) velocity. An individual particle's movement is altered according to its "local best known position" and is also manipulated toward the "best known positions" in the search-space. The best known positions are updated as positions which are more satisfactory for quality criteria, are discovered by other particles. This mode of action is supposed to conduct the movement of the swarm toward the best solutions [5].

PSO was first intended for simulating social behavior, as a stylized representation of the movement of organisms in a bird flock or fish school [6,7]. The algorithm was simplified and it was observed to be performing optimization.

Particle swarm optimization on longest common subsequence problem

This study uses PSO heuristic technique on LCSP. First, the algorithm will take n sequences and generate an alphabet among all of the distinct sequence elements without uncommon elements. Then it will generate a population of random sequences of the alphabet. Every sequence will be a particle. It will do the evaluation with the technique, occurrence evaluation, which will be described in detail in this paper. After the evaluation, known local best score will be compared with the global best score. If local best score is bigger, then it is the new global best (initial global best is 0). After this, each particle move towards to

***Corresponding author:** Meral Guzey, Department of Math and Life Sciences, Main campus of University Maryland University College (UMUC), USA, E-mail: meral.guzey@faculty.umuc.edu

Received June 25, 2013; **Accepted** August 06, 2013; **Published** August 13, 2013

Citation: Unay AT, Guzey M (2013) A Swarm Intelligence Heuristic Approach to Longest Common Subsequence Problem for Arbitrary Number of Sequences. Metabolomics 3: 120. doi:10.4172/2153-0769.1000120

the local best particle with a velocity (x - decimal representation of the sequence, y – length of the particle); it will iterate this process t times, where t is the threshold which has been set before the run.

The Algorithm and Representation

In this section, the algorithm will be explained in detail and after that, constraints which had been found during the algorithm will be mentioned.

Generate alphabet

Each subsequence has a number and variety of elements as much as input sequence it is derived from. Because of that, the algorithm first finds the distinct elements of each input sequence. Since aim is to find a subsequence, only common elements should remain in the alphabet. Thus, the algorithm checks every character if it exists in every input sequence or not. If not, then remove the element from the alphabet.

Example:

X=<A,B,C,D,E,G,C,E,D,B,G>, Y=<B,E,G,C,F,E,U,B,K>

Alphabet, A=<A,B,C,D,E,G,F,U,K> at first iteration. At second iteration, we will look for existence of every character in every sequence. Since the characters <A,D,F,U,K> don't exist in both sequences, they will be removed from the alphabet. Thus, A=<B,C,E,G>.

Generate occurrence lists

Occurrence lists are check-lists for every character which records conditions in sequence. Occurrence lists are for evaluating sub-sequence accuracy.

Maximum occurrence of characters list before last occurrence of the character: This list keeps each and every character's position in sequence in a relation to other characters. The algorithm looks for last occurrences of every character in every input sequences, and counts every character's occurrences before the last occurrence of the examined character. The algorithm is required to look for minimum number of repeats before a character in order to provide the longest valid subsequence elements, so it keeps lowest count.

Example:

X=<A,B,C,D,E,G,C,E,D,B,G>, Y=<B,E,G,C,F,E,U,B,K>

A=<B,C,E,G>

The algorithm will check all over the alphabet but let's take just 'E' for an example and look for 'C'

X=<A,B,C,D,E,G,C,**E**,D,B,G> two C's before the last occurrence of E

Y=<B,E,G,C,F,**E**,U,B,K> one C before the last occurrence of E

The algorithm take the lowest count, then now it knows that; if there are more than one C before an E, then it's not a subsequence.

Maximum occurrence of character list after first occurrence of the character: This is a list, which keeps each character's relation with other characters as well as itself. The algorithm is required to look for minimum number of repeats after a character in order to provide the longest valid subsequence elements, so it keeps lowest count.

Example:

X=<A,B,C,D,E,G,C,E,D,B,G>, Y=<B,E,G,C,F,E,U,B,K>

A=<B,C,E,G>

The algorithm will check all over the alphabet but let's take just 'E' for an example and look for 'B'

X=<A,B,C,D,**E**,G,C,E,D,B,G> one B after the first occurrence of E

Y=<B,**E**,G,C,F,E,U,B,K> one B after the first occurrence of E

Now the algorithm knows that; if there are more than one B after an E, then it is not a subsequence.

Maximum occurrence of character list after last occurrence of the character: This list keeps each character relation with other characters but not itself since it is 'the last one' by definition. The algorithm looks for last occurrence of every character in the alphabet for every input sequences and counts every characters occurrences after the last occurrence of the character. The algorithm needs to consider worst case situation, so it keeps lowest count. This list will not be used to check every occurrence of a character but the last one.

Example:

X=<A,B,C,D,E,G,C,E,D,B,G>, Y=<B,E,G,C,F,E,U,B,K>

A=<B,C,E,G>

The algorithm will check all over the alphabet but let's take just 'E' for an example and look for 'B'

X=<A,B,C,D,E,G,C,**E**,D,B,G> one B after the last occurrence of E

Y=<B,E,G,C,F,**E**,U,B,K> one B after the last occurrence of E

Now the algorithm knows that; if there are more than one B after the last occurrence of E, then it's not a subsequence.

Maximum occurrence of character list before first occurrence of the character: This list keeps each character's relation with other characters but not itself since it's the first one already. The algorithm looks for first occurrence of every character in the alphabet for every input sequences and counts every characters occurrences before the first occurrence of the character. The algorithm needs to consider worst case situation, so it keeps lowest count. This list will not be used to check every occurrence of a character but the last one.

Example:

X=<A,B,C,D,E,G,C,E,D,B,G>, Y=<B,E,G,C,F,E,U,B,K>

A=<B,C,E,G>

The algorithm will check all over the alphabet but let's take just 'E' for an example and look for 'C'

X=<A,B,C,D,**E**,G,C,E,D,B,G> one C before the first occurrence of E

Y=<B,**E**,G,C,F,E,U,B,K> no C's before the first occurrence of E

The algorithm take the lowest count, now it knows that; if there are any C's before the first occurrence of E, then it's not a subsequence.

Total occurrence of a character: This is a list of characters, which looks for the minimum of the total occurrences of each character's in every input sequence. The algorithm needs to consider worst case situation, so it keeps lowest count.

Example:

X=<A,B,C,D,E,G,C,E,D,B,G>, Y=<B,E,G,C,F,E,U,B,K>

A=<B,C,E,G>

The algorithm will check all over the alphabet but let's take just 'E' for an example

X=<A,B,C,D,**E**,G,C,**E**,D,B,G> two E's Y=<B,**E**,G,C,F,**E**,U,B,K> two E's

Now the algorithm knows that, if the count of E is bigger than two then it's not a subsequence.

Occurrence evaluation

In particle swarm optimization, we need an evaluation function to tell particles what to do for their next step. For longest common subsequence problem, we need an evaluation function to tell how close a particle to being a subsequence. Occurrence evaluation uses the occurrence lists as mentioned above to evaluate a particle and outputs a score between 0 and 1. 1 means the particle is 100% subsequence.

- Initially, output score (eval) is particle length (pl)
- For every "faulty" character, it drops by -1
- At the end, last score will be divided by pl

Generate population

The algorithm generates totally random, S sized population. Minimum size of a particle is 1 and maximum size of a particle is the lowest length between all of the input sequences, because generating a common subsequence bigger than any sequence is not possible.

Updates and particle moving

After initial procedures, (generating population and occurrence lists) we put every particles to the occurrence evaluation function. According to their score and length, the algorithm evaluates a fitness score for every particle. Biggest score will be the local best of that iteration and then it will be compared with the global best (global best is initially set to 0) If it's bigger, then it will be the new global best [8].

For the last part, all of the particles in the population will be moved to the local best (fitness and moving part will be described in detail in the representation section). The update part will be repeating itself T times and algorithm will be stop (T is the threshold value which will be set before running the algorithm) Usually, there would be also a convergence condition to stop PSO but in LCSP, max possible outcome is in the length of the minimum input sequence and this condition is highly unlikely for determining a condition. So in this study, threshold value is the only condition to stop the algorithm.

Representation

We need a solid representation in order to apply PSO on any problem. For LCSP, we need to represent particles, solutions, positioning, movement, evaluation and fitness. So, pretty much everything.

Particles

Each particle is a sequence generated by a random function. Minimum length of a particle is one and maximum length is the lowest length of the input sequences. Sequence will be generated with the alphabet, which we generated before.

Solutions

In this study, each particle is not a solution. A solution is 100% valid subsequence of the given input sequences and the best solution is the longest of the solutions. In this study, particles are "potential" solutions.

Positioning

Each particle has position information in a 2D solution space.

X axis: In this study, X axis is the decimal value of the sequence. For calculating this decimal value, we need;

- "base" value which is the length of the alphabet plus 1, b+1. That plus one is necessary, because 0123 and 123 is the same value for decimal values. abcd and bcd is not the same for sequences. So we need to increase position values by 1. If we increase the position values, we have to increase the base value also.
- "power" value which is the position value in the particle for every element in a particle, p.
- Every element in the alphabet has a unique position number, u.
- Thus, decimal value of a particle p, X is; (l : particle length)

$$X=\sum_{p=o}^{b} u*(b+1)^{p}$$

Example:

Let, P=<BCCEBG>

Alphabet, A=<B,C,E,G>

Unique Position Values

B=1, C=2, E=3, G=4

Base

Alphabet Length+1=4

Power

From 0 to length of the particle.

X Value For BCCEBG

$X=(1*4^{0})+(2*4^{1})+(2*4^{2})+(3*4^{3})+(1*4^{4})+(4*4^{5})$

1.1.1 Y axis: Y axis is basically length of the particle.

Example:

Let, P=<BCCEBG>

Y Value for P is 6.

Movement: Movement is the most important part of the PSO algorithm. Particles move to the best position with a velocity (in this study, velocity values are given before the run).

Move to X: Pick a random element in a sequence, take its position value, add velocity x value, divide it to the alphabet length, replace the element with the position value with remaining of the division (modulus).

Example:

Let, P=<BCCEBGE>, let's move it by 1

Pick a random element: <BCCEBGE>

Move it by 1 <BCCEBBE>.

Move to Y: Increase or decrease the particles length by given velocity with random elements within the alphabet.

Evaluation and Fitness: As mentioned in the second section (the algorithm), the algorithm evaluates particles with the occurrence lists and outputs a value between 0 and 1. Since our desire is not finding a

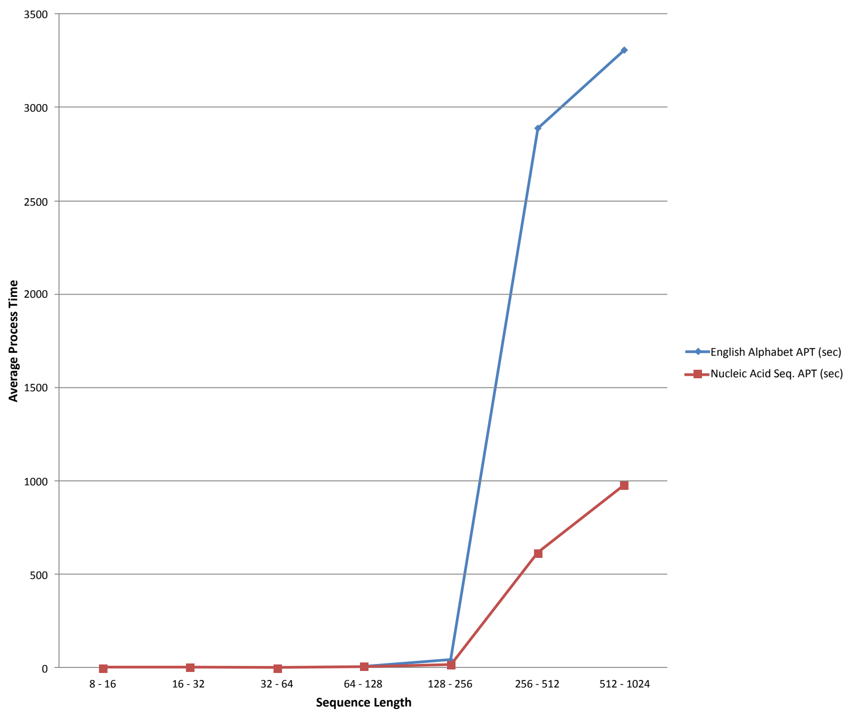

Chart 1: Average process time results. According to the results, main differences between two particular tests come up at interval six (256–512) and further. It proves that, the alphabet length is a crucial factor for the speed performance of the algorithm, since English Alphabet has 26 unique characters and a Nucleic Acid Sequence has only 4 unique characters.

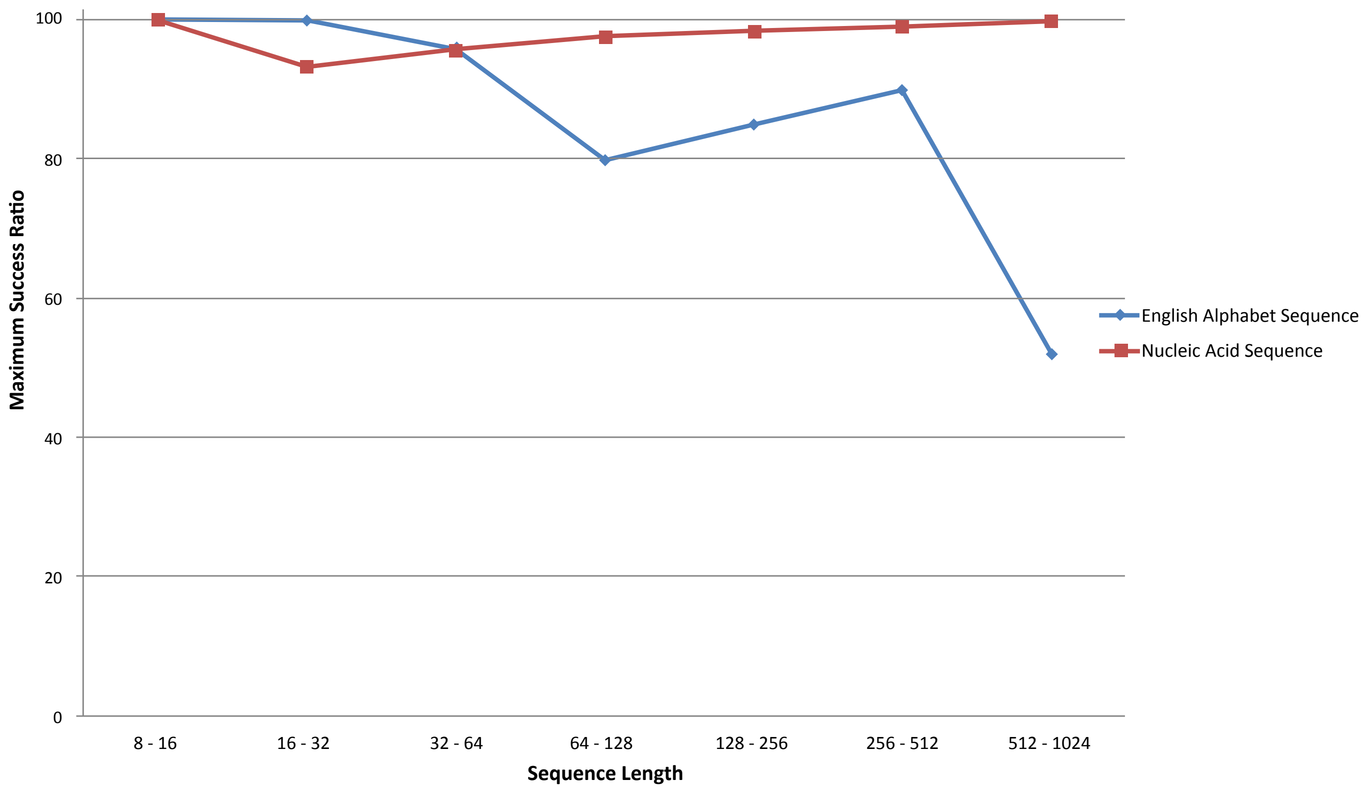

Chart 2: Average Success Ratio Results.

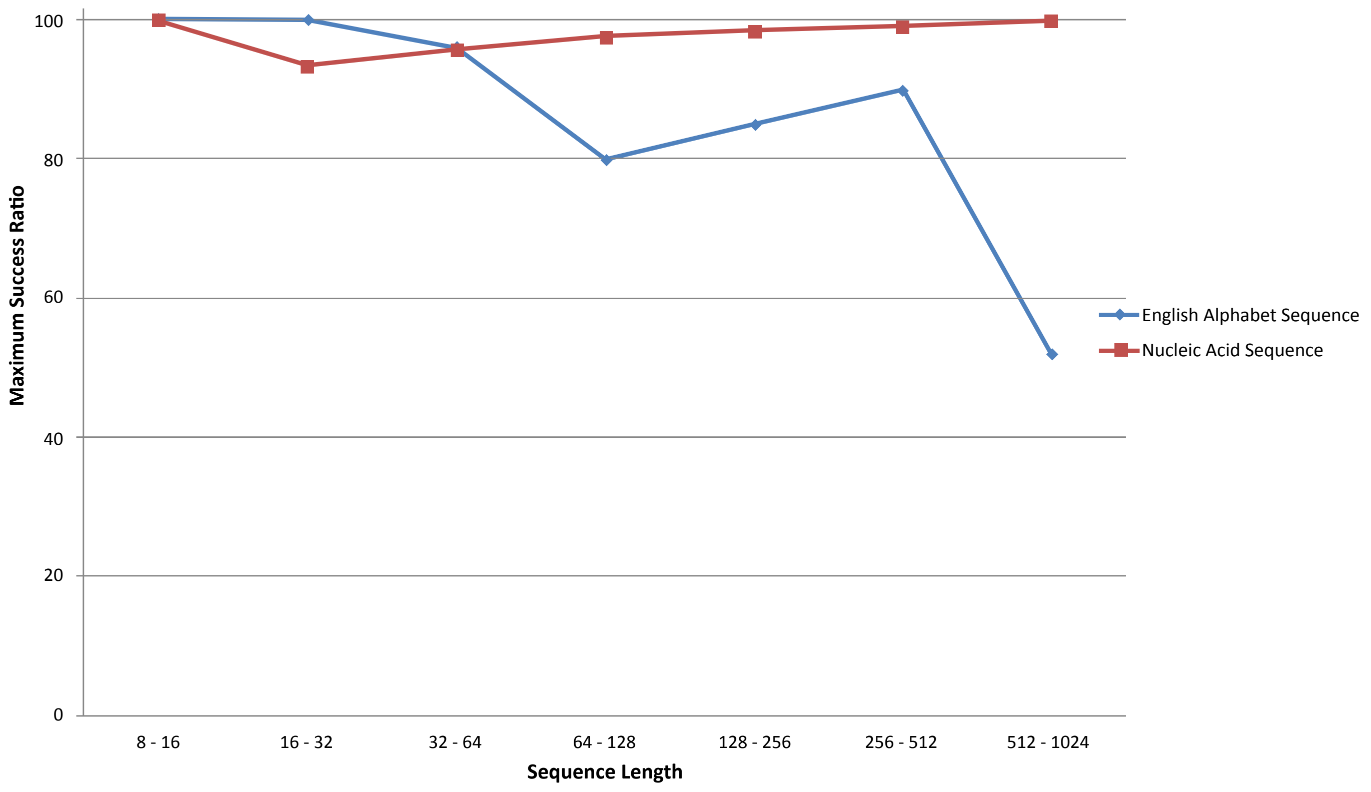

Chart 3: Maximum success ratio results. Success ratio stands for how close the output sequence to being a common sub-sequence. Since it is a heuristic approach, it may not give a 100% common subsequence but thanks to the OL technique, the output can be evaluated for it.

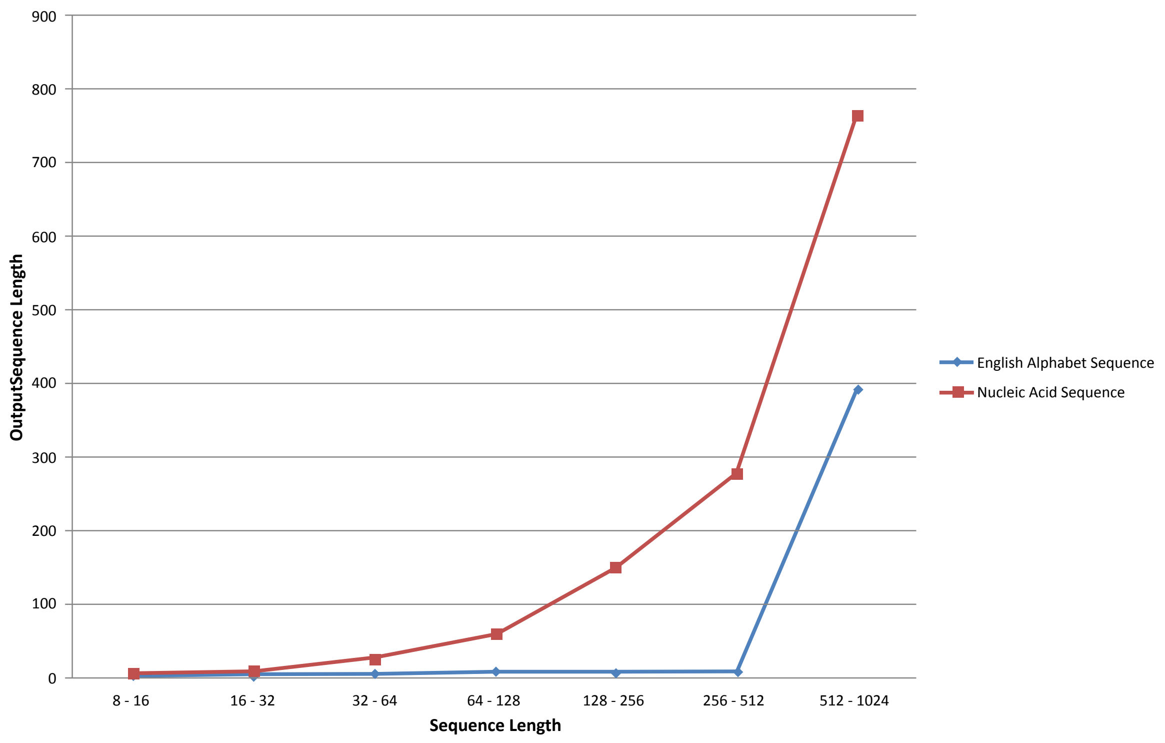

Chart 4: Average output length results.

subsequence but finding the longest subsequence, we multiply it with the length of the particle, and acquire fitness score.

Results

Two different types of tests have been applied. One has took two sequences generated with English Alphabet (26 characters) and the other one has two DNA strings (4 characters, A – G – C - T) as input. Every instance has been made with random sequences which has random number of length between specific boundaries (8-16, 16-32... etc.). PSO algorithm generates 1000 population and each particle moves one unit to X axis and one unit to Y axis, through the global best on each iteration. Whole run of the algorithm does ten iteration. The algorithm has run ten times with the same couple of sequences for each interval.

The algorithm were implemented in Java and executed on 2 GB of memory and AMD Turion X2 Ultra Dual-Core Mobile Processor ZM-82 (2.2 GHz, 2 MB L2 cache) under Ubuntu 10 operating system.

Analyzing the test results

The results have been analyzed on three criteria; process time, success ratio, and output length (Tables 1 and 2).

Process time result analysis

Process time result analysis is described in Chart 1.

Average success ratio result analysis

Test results show that, the algorithm is having trouble for English Alphabet sequences as input lengths increase. Yet, this situation is not valid for nucleic acid sequences, since the output is 100% near at almost every interval. Thus, the algorithm is very dependant to the alphabet unique character count (as known as alphabet length) for precision (Charts 2 and 3).

Output length analysis

Output length is the most important part of the LCSP. Maximum common subsequence length is equal to the length of the shortest input sequence. Although that result is unrealistic for different sequences, which may only occur, if one sequence contains the other completely (Chart 4).

According to the test results, the algorithm can find really good results for nucleic acid sequences since outputs are close to the maximum length. It is not true for English Alphabet results, though. But we cannot be sure about those are "bad" results, since they're completely random sequences.

Sequence Length	Avg Process Time (sec)	Avg Subsequence Ratio	Max Success Ratio	Avg Output Length
8-16	0.48	90.6	100	4
16-32	0.81	87.2	100	5
32-64	0.96	81	96	6
64-128	11	63	80	9
128-256	44.7	74.3	85	8
256-512	2895.5	63	90	10
512-1024	3310.2	45	52	392

Table 1: Test Results (English Alphabet).

	Avg Process Time (sec)	Avg Subsequence Ratio	Max Success Ratio	Avg Output Length
8-16	0.12	86.5	100	7
16-32	0.67	85.5	93.3	10
32-64	1.9	77.1	95.6	27
64-128	8.9	95.3	97.6	60
128-256	15.6	97.6	98.4	150
256-512	618.2	98.5	99	278
512-1024	978.5	99.5	99.8	365

Table 2: Test Results (DNA Strings A-G-C-T).

Future Work

Currently, our study aims to create a new heuristic approach to the LCSP. Our future plan is to study on real prostate cancer data to test the applicability of this sequencing technique. Further, we planned to test dynamic programming and some of the best known heuristic techniques in the contemporary literature using the same datasets in order to compare them to PSO with OL more accurately. Additionally, the tests in the future work will be running through more than two sequences, which will help us to understand the competency of algorithms in a task structure with increasing complexity.

Conclusion

The purpose of this study is to apply particle swarm optimization technique to the longest common subsequence problem. We propose OL technique as an evaluation function to PSO for applying PSO to LCSP. Since there are severe differences between popular approaches, like dynamic programming or other heuristic methods, it's not very easy to fully compare this algorithm with them, but there are obvious benefits in PSO with OL approach: it is not bounded with prerequisites of "fixed number sequences" such as known length and equal length of input sequences [7,9]. With correct population settings and parameter tuning, it can give good solutions, and accurate evaluation score, yet it may not give the "best" (longest) solution since it's a heuristic approach.

Acknowledgement

Dr. Meral Guzey acknowledges her sincere thanks to the Department of General Medicine, NIH, Bethesda, Maryland, USA (Research Supplements to Promote Re-Entry into Biomedical and Behavioral Research Careers (PA-08-191).

References

1. Irving RW, Fraser C (1992) Two algorithms for the longest common subsequence of three (or more) strings. Combinatorial Pattern Matching 644: 214-229.
2. Hirschberg DS (1977) Algorithms for the longest common subsequence problem. Journal ACM, 24: 664-675.
3. Tseng TC, Yang CB, Ann HY (2013) Efficient algorithms for the longest common subsequence problem with sequential sub-string constraints. Journal of Complexity 29: 44-52.
4. Maier D (1978) The complexity of some problems on subsequences and supersequences. JACM 25: 322-336.
5. Kennedy J, Eberhart R (1995) Particle Swarm Optimization. Neural Networks 4: 1942-1948.
6. Shi Y, Eberhart RC (1998) A modified particle swarm optimizer. Proceedings of IEEE International Conference on Evolutionary Computation 69-73.
7. Kennedy J, Eberhart RC, Shi Y (2001) Swarm Intelligence. Morgan Kaufmann -Academic Press.
8. Poli R (2008) Analysis of the Publications on the Applications of Particle Swarm Optimization. Journal of Artificial Evolution and Applications.
9. Hinkemeyer B, Julstrom BA (2006) A genetic algorithm for the longest common subsequence problem. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, Washington, USA 609-610.