Efficient and Robust Constraint Automaton-Based Genome Map Assembly

Peter Z. Revesz
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588
1 402 472 3488
revesz@cse.unl.edu

Dipty Singh
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588
1 402 472 7767
singh@cse.unl.edu

ABSTRACT
DNA sequences are cut into smaller fragments using restriction enzymes in order to facilitate analysis. Application of different restriction enzymes to multiple copies of a DNA sequence generates many overlapping fragments. To reconstruct the original DNA, these fragments need to be sequenced and assembled. This problem of finding the original order of the fragments is called the genome map assembly problem. We propose a constraint automaton solution to solve the genome map assembly problem for both error prone and error free data. Plasmid vectors puc57, pKLAC1-malE, pTXB1 and phage vector Adenovirus2, having a size in base pairs of 2710, 6706, 10153 and 35937 respectively, were used to prove that computational time for solving genome map assembly problem using constraint automaton solution is linear with both precise and approximate data.

Categories and Subject Descriptors
J.3 [**Computer Applications**]: Life and Medical Sciences – *biology and genetics.*

General Terms
Algorithms, Experiments, Measurement.

Keywords
Constraint automaton, genome map assembly, plasmid, virus.

1. INTRODUCTION
DNA sequences range from thousands to billions of base pairs. However, currently available sequencing machines can only handle a couple of thousand base pairs at a time. So the long DNA sequences have to be cut into smaller subsequences using restriction enzymes. After cutting the DNA, small fragments just float randomly in the solution, losing all the information about the original order of the sequence. After the subsequences are sequenced and analyzed, they have to be arranged and assembled to obtain the original sequence. This process is called genome map assembly and the problem of fully executing the genome map assembly process to recover the original sequence is called the Genome Map Assembly Problem (GMAP). The common solutions to the genome map assembly problem are based on using restriction enzyme fingerprints [1,2,4,5,6,14,17,18]. Revesz proposed an extension of this approach based on constraint automata [9,7,10] that derive from constraint databases [3]. This paper improves on the existing constraint automata solution for both precise and error-prone data. The latter is a practical problem because error may be introduced by imprecise measurements of the lengths of genome fragments as measured by number of base pairs.

C3S2E'14, August 03-05, 2014, Montreal, QC, Canada.

http://dx.doi.org/10.1145/2641483.2641533

This paper is organized as follows. Section 2 describes basic concepts of constraint automata the genome map assembly problem and constraint automata solutions to the genome map assembly problem. Section 3 presents new methods that extend the constraint automata solutions. Section 4 describes experiments and analyzes the results. Section 5 presents some conclusions and future work.

2. BASIC CONCEPTS
2.1 Constraint Automata
Based on conditions that are described using constraints on variables, constraint automata, are used to control the operation of a system. It has to find the set of reachable configurations, which is the set of states and state values that the constraint automaton can enter. This is one of the important problems in constraint automata. Each constraint automaton consists of set of states, a set of state variables, transitions between states, an initial state, and the domain and initial values of the state variable. Each transition consists of guard constraints (set of constraints) followed by assignment statements. In constraint automata, the assignment statements are shown using the symbol '=' and guards are followed by questions marks, e.g. a>=100?

If there is a transition whose guard constraints are satisfied by the current values of the state variables, a constraint automaton can move from one state to another. In addition to the state variables, the transitions of a constraint automaton may contain variables. Some of the values for these variables must be found such that the guard constrains are satisfied and the transition can be applied. For this reason, these variables are said to be existentially quantified variables. By sensing the current value of a variable, a constraint automaton can interact with its environment. This is expressed by read(x) command on a transition between states, where x is any variable. This command can appear either before or after the guard constraints, which updates the value of x.

2.2 Genome Map Assembly Problem
Currently available sequencing machines can only handle DNA fragments of a couple of thousand base pairs. The long DNA sequences have to be cut into small fragments using restriction enzymes. This cutting results in the loss of information about the

original order of the fragments. After these fragments are sequenced and analyzed, they have to be arranged and assembled to obtain the original sequence. This process is called Genome Map Assembly. The problem of reassembling these fragments from data that is incomplete, imprecise, ambiguous and often contradictory is known as the Genome Map Assembly Problem (GMAP) [10].

Genome sequencing is the process of finding the precise order of DNA nucleotides in a genome; the exact order of adenine, cytosine, guanine, and thymine that make up an organism's DNA. Genome sequencing allows scientists to sequence genes and genomes. Due to the limitation of how many bases can be sequenced in one experiment, DNA has to be broken into smaller fragments before they can be sequenced and reassembled.

Genome mapping is the process of finding the approximate position of genes in a genome without getting into the details of the actual sequence. It is a graphical representation that helps to find out where you are and how to get to where you want to go. It identifies the order of the specified subsequences in the genome. The genome map contains various landmarks identified with a series of letters and numbers, which help the researchers, find where specific pieces of DNA belong in the overall genomic jigsaw puzzle.

Restriction endonucleases, commonly called *restriction enzymes,* are nucleases that are made by bacteria to protect themselves from a virus by cutting their genomes at sites that have a specific pattern [3]. *Restriction site analysis* and *hybridization* are the two most popular method of getting fingerprints. In restriction site analysis, one or more restriction enzymes are applied to the DNA sequence and the lengths of the resulting fragments are measured, these lengths thus serve as the "fingerprint" of each subsequence. In hybridization fingerprinting, certain small sequences are checked to see if it binds to fragments, the subset of such small sequences that binds to the fragment serves as its fingerprint.

2.3 The Constraint Automata for GMAP

The algorithm of the constraint automata for GMAP can be described easily, if we define GMAP as a Big-Bag Matching Problem [10]. The Constraint Automata Solution proposed by Ramanathan and Revesz [7] has been modified to deal with both error prone and error free sub fragments of DNA.

A bag is a multiset, a generalization of a set in which each element can occur multiple times. A big-bag is a multiset whose elements are bags that can occur multiple times. [10].

Each permutation of the bags and permutation of the elements of each bag within a big-bag is called a presentation [9,10] There are several different presentations of a single big-bag. The big-bag matching decision problem (BBMD) is the problem of deciding whether two big-bags match. The big-bag matching problem (BBM) is the problem of finding matching presentations for two given big-bags if they match [24][25]. We use the concept of BBM to solve the GMAP. A modified version of Revesz's Fingerprint [10] data has been used to collect fingerprints of input data. Instead of using three restriction enzymes only two enzymes have been used. The fingerprints collected will later be used as an input for Constraint Automata Solution for GMAP. The methodology was as follows:

1. An original DNA sequence was copied.
2. Restriction enzyme "a" was applied to the copied sequence which created several fragments of varying lengths depending on the restriction site.
3. Individual fragments were separated.
4. Restriction enzyme "b" was applied to the separated fragments from step 3, producing sub-fragments.
5. 5.The length of individual sub-fragments was measured.
6. Steps 1-5 were repeated, but restriction enzyme "b" was applied first and restriction enzyme "a" was applied consequently.

All the elements from first copy of DNA are stored in Big-bag-A and elements from the second copy of DNA are stored in Big-bag-B. For the GMAP, we use a constraint automaton (see Figure 1) that is a modified version of the ones proposed by Revesz and Ramanathan [9,7,10]. The automaton has the following states:

- INIT – this is where the automaton begins
- A-ahead – if A bag is ahead
- B-ahead – if B bag is ahead
- HALT – if the solution is found

The automaton has the following state variables:

- UA: set of A bags which has not been used yet
- UB: set of B bags which has not been used yet
- S: set of elements by which either A or B bag is currently ahead

The automaton starts in the INIT state and tries to reach the HALT state. The automaton moves from left to right by adding either an A bag or a B bag. If A bag is greater than B bag, it goes to A-ahead state. Else, it goes to B-ahead state. If they are equal, it goes to INIT state and starts the automaton with remaining UA and UB. When UA, UB, and S are empty and all the bags are used, automaton goes into HALT state and stops.

This algorithm does not use backtracking. If it does not find the elements in S in both A-bag and B-bag it goes back to INIT state and starts the automaton all over again. This makes it very inefficient. We further improved the efficiency by making a deterministic, backtracking automaton and also extended it to be able to handle errors. Our new constraint automaton (see Figure 2) adds new state and state variables to the existing constraint automaton. Following are the states:

- Error-Check – check to see if input data has any errors
- Replace-Error- – if there is an error, replace it with the mean of two mismatched data elements from both A and B bags
- INIT – this is where the automaton begins
- A-ahead – if A bag is ahead
- B-ahead – if B bag is ahead
- Backtrack – if solution is not yet found but the S is empty

- HALT – if the solution is found or if the error is greater than error tolerance value

The automaton has the following state variables:

- Error: difference between mismatched values from A and B bags
- ErrorTolerance: the specified error tolerance
- Length-A : all the elements of S that belongs to Big-Bag-A
- Length-B : all of the elements of S that belong to Big-Bag-B
- UA: set of A bags which has not been used yet
- UB: set of B bags which has not been used yet
- CurrBag : current bag, which is the SelBag from previous state
- S: set of elements by which either the A or the B bag is currently ahead
- Options: the set of bags from which the next bag can be chosen
- SelBag: the bag that as selected from the Option as the next bag
- Choices: set of remaining bags that were not picked from Options besides the SelBag.
- Cflag: 0 if Choices is empty and 1 if Choices is not empty.

The automaton goes through the following steps to solve the GMAP. The input to the constraint automaton is the fingerprints of DNA fragments. Bags of big-bag A and big-bag B are fed to the constraint automaton. The automaton starts with Error-Check state where input data are checked for any errors. An error may occur when the length of sub-fragments of DNA is measured, which results in two different sets of elements in big-bags A and B. To check for any errors, each element of bags within big-bag-A is compared with the elements of bags within big-bag-B. If the elements do not match, then the input data has some error. Otherwise the input data is error free.

If there is no error in the data it goes to INIT state. If the error is more than specified error tolerance value, then it goes to HALT state.

If the error is less than or equal to the specified error tolerance value, it goes to Replace-Error state.

If the data has some error, it goes to the Replace-Error state. The idea is to replace the wrong data with the mean of two mismatched data, so that it will have the same set of data in both big-bag A and big-bag B.

Let's consider the sequence of plasmid puc57. If we use the procedure discussed earlier to collect fingerprints of the sequence, we will get the following fingerprint data:

Big-Bag A	Big-Bag B
355	355, 19
19, 196	196, 288, 121, 13, 541
288	416
121	373
13	320
541, 416, 373, 320, 68	68

While measuring the length of each sub-fragment, if we get "17" instead of "13" in Big-Bag-A and "418" instead of "416" in Big-Bag-B. Then we will have the following as the input data.

Big-Bag A	Big-Bag B
355	355, 19
19, 196	196, 288, 121, 13, 541
288	418
121	373
17	320
541, 416, 373, 320, 68	68

The automaton compares each element of both bags and finds that 17 and 13, and 416 and 418 do not match. It takes the mean of each pair of mismatched elements and replaces them with their mean in both Big-Bag-A and Big-Bag-B.

Big-Bag A	Big-Bag B
355	355, 19
19, 196	196, 288, 121, 15, 541
288	417
121	373
15	320
541, 417, 373, 320, 68	68

After replacing the wrong data, both bags contain the same set of elements and they move to INIT state.

The automaton comes to INIT state, if either the data has no error or if all the errors have been replaced by the means of two mismatched data. All the A bags are stored in UA and all B bags are stored in UB. Since this is the first state where it actually starts processing data for solving GMAP, it can pick any bag from either the big bag A or B. So Options is set to all the bags from A and B. The leftmost bag is selected from Options and set to SelBag and remaining bag is set to Choices. The elements of SelBag are set to S. Since Choices is not empty, Cflag is set to 1. The bag that is just selected is removed from either UA or UB,

from where ever it belongs The elements of the bag in SelBag are set to either Length-A or Length-B. If the Length-A is greater than Length-B, the automaton goes to A-ahead state, if not then it moves to the B-ahead state.

Using figure 2.5, if A1 is picked as the SelBag, then, UA=A2, A3, A4, A5; S = 355 and Length-A = 355. The automaton goes to the A-ahead state.

If the automaton is in A-ahead state, the next bag to be picked is selected from UB. The automaton tries to match the elements of S and SelBag as far as possible.

If S is a subset of SelBag, then the automaton moves to B-ahead state and S is set to the difference between S and the elements of SelBag.

If SelBag is a subset of S, then the automaton moves to A-ahead state and S is set to the difference between S and the elements of SelBag

The elements of S are now compared with the elements of all the bags of UB. All the bags that are either a subset or a superset of S are set to Options. If it doesn't find any bags that meet these criteria, the automaton goes to Backtrack state. Using Figure 2, if S = "541, 288, 121, 373, 68, 416, 320" and Options = "B3, B4, B6, B5". Again, the leftmost bag is picked and set to SelBag.

If the automaton is in B-ahead state, the next bag is selected from UA. Similar procedure as in step 4 is followed to select the next SelBag. Again, if SelBag is empty, it goes to backtrack.

If the automaton is in the backtrack state, it goes back to the last node of which Cflag is set to 1. The idea is to select some other bags from Options besides the one picked initially, which might have led to the backtrack state. So it tries to go to a node where there are some other Choices. For this reason Cflag was used initially to help keep track of Choices, if we need to come back and look for other options.

It retrieves the values of all state variables from that particular node and resets the current values of state variables with the one from that node. It either goes to A-ahead or B-ahead state, depending on the current value of Length-A or Length-B. No matter which state it goes to, it picks the next bag from Options, discarding the ones that it has already tried and has failed.

If Length-A is greater than Length-B, automaton goes to A-ahead state; if Length-B is greater it goes to B-ahead state. But sometimes, both Length-A and Length-B can be equal. In this case, the automaton selects one random bag from UA and continues the automaton.

If all the bags in UA and UB are used and if S is empty, the solution has been found and it goes to the HALT state.

3. METHODOLOGY

The constraint automaton was implemented using Perl (Practical Extraction and Report Language) v5.12.2 downloaded from <http://www.activestate.com/activeperl/downloads>. Perl is commonly used in bioinformatics [17].

In addition, the program was compiled in Eclipse SDK v3.5.2 (http://www.eclipse.org/downloads/) using EPIC (Eclipse Perl Integration). Eclipse is a multi-language software development environment. EPIC is an open source Perl Integrated Development Environment is based on the Eclipse platform.

3.1 Data Sources and Data Collection

To implement and test any algorithm we need to have data sets. The data for the implemented algorithm are the sequences of DNA of plasmids and phage. The sequence of DNA is often stored in a flat text file called FASTA file. It is a text-based format for representing nucleotide sequence or peptide sequence. "A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater than (">") symbol in the first column." The FASTA files for the plasmids and phage were downloaded from New England BioLabs. (http://www.neb.com).

One way of making copies of a DNA is to insert a DNA piece into the genome of an organism, a host or vector and let the organism multiply itself. The inserted piece (the insert) gets multiplied along with the original DNA of the host upon host multiplication. "A plasmid is a piece of circular DNA that exists in bacteria." [13]. It replicates itself when the cell divides and each copy of a daughter cell keeps one copy of the plasmid. Plasmids make good vectors but can only handle inserts up to 15kbp [13]. Bacteriophages or just phages are viruses that infect bacteria. They are often used as vectors. Inserts in phage DNA get replicated when the virus infects a host bacterium. To observe the variation in computational complexity with respect to different length sequences, plasmid and phage ranging from 2710 to 35937 bp were chosen. The following plasmids and phage are used to test the purposed algorithm.

- pUC57: 2710 base pairs
- pTXB1: 6706 base pair
- pKLAC-malE – 10153 base pairs
- pB85766 – 14875 base pairs
- Adenovirus-2 – 35937 base pairs

To collect fingerprints of sequences, restriction enzymes MvaI and MaeII were used. MvaI is an isolate from Micrococcus varians RFL19 and has restriction site at CC^WGG. "W" can be either A or T. MaeII is isolated from Methanococcus aeolicus and has restriction site at A^TCG.

DNA sequences were cleaved into fragments and sub-fragments by using Webcutter 2.0 [35]. Each DNA sequence is first cleaved by MvaI and each individual fragement was again cleaved by MaeII to obtain sub-fragments. This is the data for Big-Bag-A.

No	Fragments	Sub-Fragments
1	355	355
2	215	19, 196
3	288	288
4	121	121
5	13	13
6	1709	541, 416, 373, 320, 68

Again, each DNA sequence is cleaved by MaeII and then by MvaI to obtain Big-Bag-B.

No	Fragments	Sub-Fragments
1	374	355, 19
2	1159	196, 288, 121, 13, 541
3	416	416
4	373	373
5	320	320
6	68	68

4. EXPERIMENTAL RESULTS AND ANALYSIS

Input data of all five DNA sequences were fed to the implemented application and the results were analyzed separately and also compared with one another. The results were compared by the time it takes to assemble each subsequence for both erroneous and error free data. To better analyze the results we considered input data both without measurement errors and with error. For space limitations, we describe only the latter case.

We set the error tolerance to 5, that is, we allowed a percent of the original data values to deviate by at most 5 units from the precise measurements. Table 1 illustrates the process of adding random errors within the threshold to mimic measurement errors. Below is a summary of the same input data.

A1	A2	A3	A4	A5	A6
355	19, 196	288	121	541, 416, 373, 320, 68	13
355, 19	196, 13, 288, 121, 541	416	373	320	68
B1	B6	B2	B3	B4	B5

The execution steps of the constraint automaton are shown in Table 1. The average execution time for the erroneous input data for each DNA sequences is as follows:

Sequence	Base Pairs	Bags	Time (s)
pUC57	2710	10	0.32
pTXB1	6706	44	0.88
pKLAC-malE	10153	46	1.04
pB85766	14875	84	19
Adenovirus	35937	221	76

The execution time data for error-free and erroneous data are visualized in Figures 3 and 4.

With errorenous input data, the R-square value for linear equation is 0.96 and P-value is 0.003. This concludes that the linear function better fits (R-square value approximately equal to 1 and P-value <0.01) the execution time and the length of DNA sequence than the cubic, quadratic and exponential functions.

Functions	P-Value	R^2- Value
Linear	0.96	0.003
Quadratic	0.987	0.013
Cubic	0.997	0.071
Exponential	0.845	0.027

The execution times for different sets of input were analyzed to predict the pattern of output and use the results to help reduce the execution time to find the solution for GMAP.

Sequence	A Bags	B Bags	Difference	Time
pUC57	5	5	0	0.31
pTXB1	20	24	4	0.84
pKLAC-malE	21	25	4	1.01
pB85766	51	33	18	18.1
Adenovirus	137	84	53	74.6

From the above table, the difference between number of bags in Big-Bag-A and Big-Bag-B for pB85766 is 18. The higher the difference between numbers of bags, there is a high chance that there will be more bags with just one element in which ever big bag has higher number of bags.

If the numbers of bags with only a single element is high, there is less likelihood of finding the overlapping fragments in the opposite Big-Bag without backtracking multiple times. The output file in Appendix BA shows the stepwise fragment assembly for pB85766. The program backtracks several times and ultimately finds the solution at an average execution time of 18.1 seconds.

To investigate whether or not difference in numbers of bags with two big bags effect the execution time, another pair of restriction enzymes; MaeI(C^TAG) and HinfI(G^ANTC) were applied to the DNA sequence of pB85766. In this case, the number of A and B bags changed and the execution time improved as follows.

Sequence	A Bags	B Bags	Difference	Time
pB85766	44	40	4	3.5

Even though the total number of bags are almost the same, the difference between numbers of bags is significantly reduced. This resulted in many overlapping fragments in the opposite big-bags and led to finding the solution within only 3.5 seconds by eliminating many backtrackings.

5. CONCLUSIONS AND FUTURE WORK

The Genome Map Assembly Problem was solved using an abstraction of the GMAP that allows error tolerance unlike the original proposal by Revesz [10]. During fragment assembly the algorithm does not distinguish between these two sub-fragments and may result in wrong assembly of original DNA In the future, this application can be modified to create unique fingerprint for each sub-fragment, so that the fragment assembly is more precise even in the presence of duplicate length sub-fragments.

Future research may also take advantage of more general methods of combining information that may be slightly contradictory, like in the case of measurement errors for the number of base pairs. For example, contradictory infromation can be combined using arbitration operators [8] and classification integration techniques [12]. Trying to reconstruct an phylogenetic tree of evolutionarily related species also needs the combination of the divergent genomes of existing species into a hypothetical common ancestor genome. Although in this case the differences are not due to measurement errors but to ecolutionary drift [15], the combination needs a similar approach. A recent proposal that emphasises commonalities insteaad of differences while reconstructing a hypothetical evolutionary tree is [11].

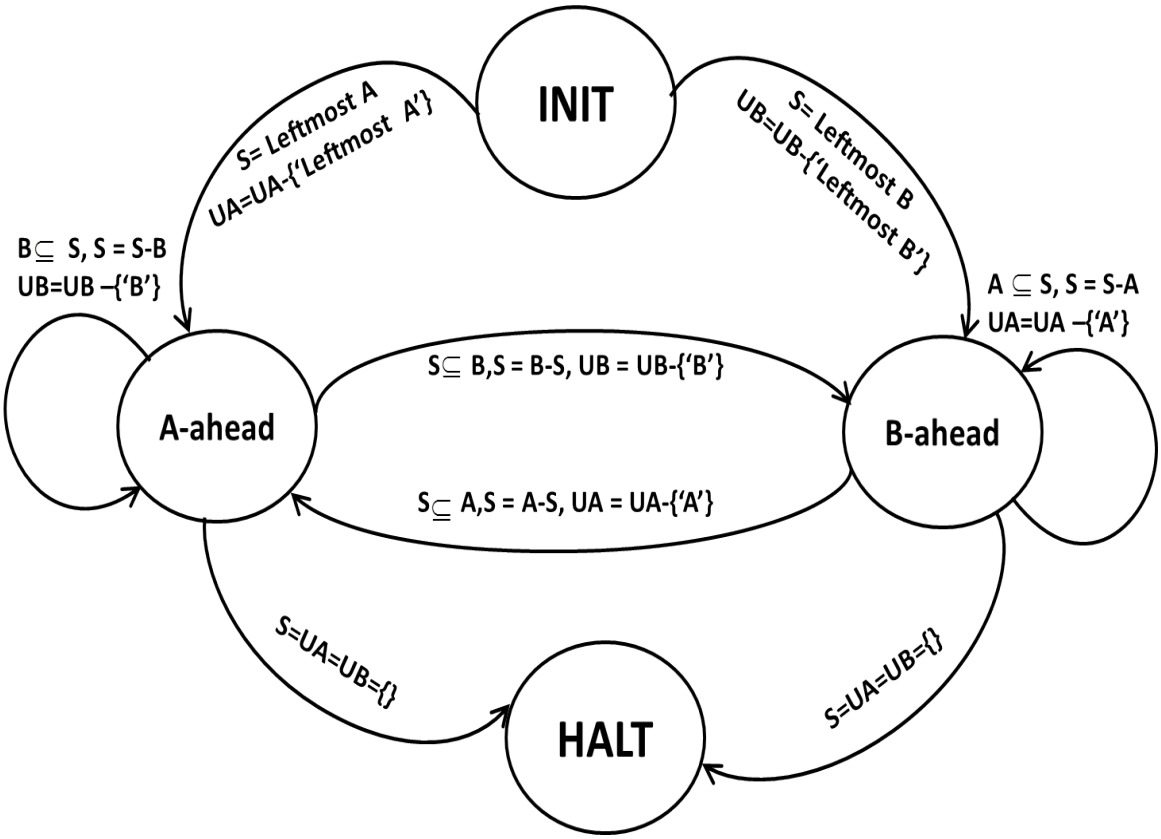

Figure 1. The original non-deterministic constraint automaton for the genome map assembly problem.

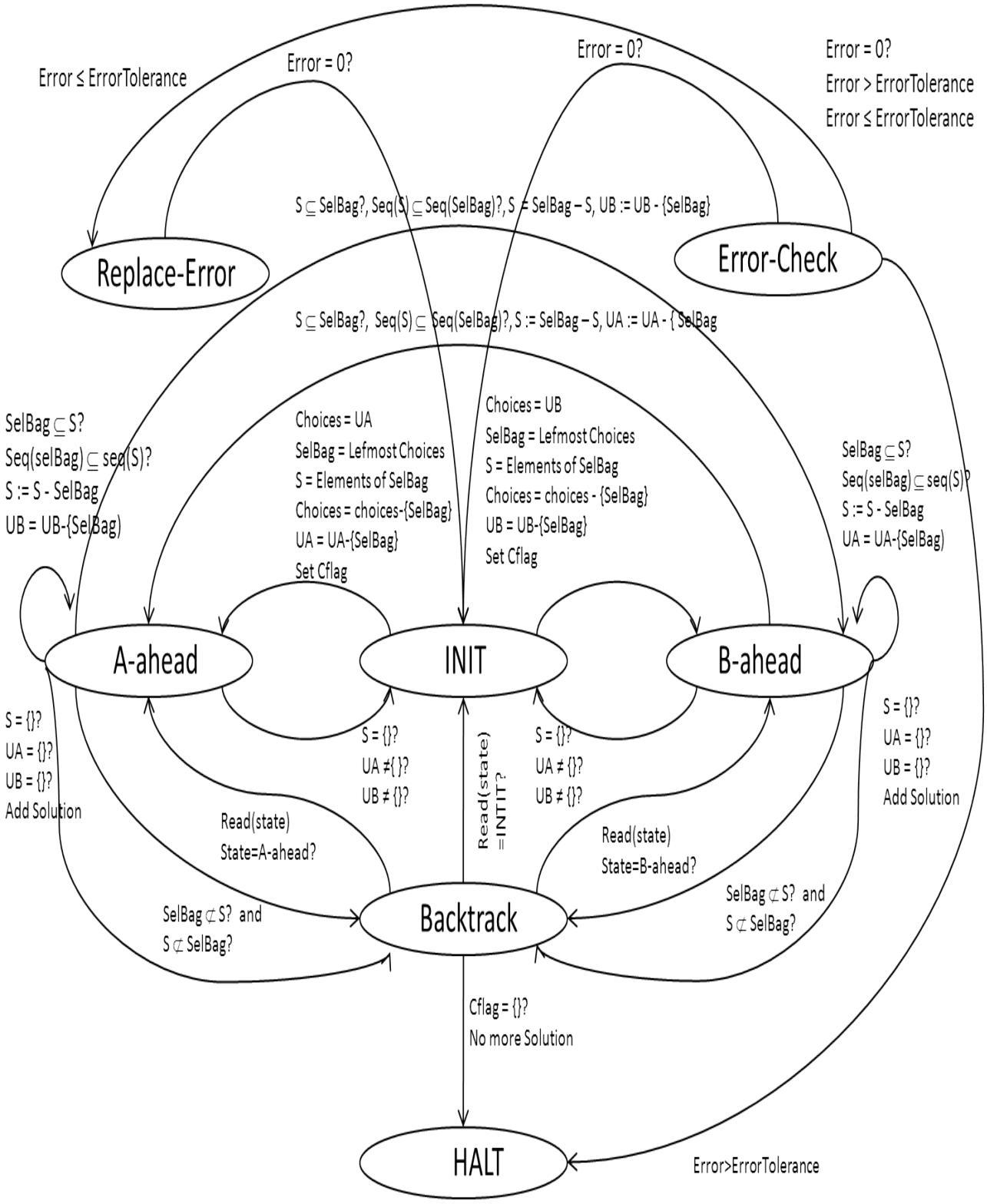

Figure 2. The new deterministic constraint automaton for the genome map assembly problem.

Table 1. This table shows the process of taking the original input data and randomly applying changes that mimic measurement errors. The individual bags are also randomly reordered. All the introduced errors are less than or equal to the threshold value.

Original Input Data	Randomized Input Data with Error	Bags
<BAG>	<BAG>	Big-Bag-A
355	**355**	A1
19 196	19 196	A2
288	**288**	A3
121	121	A4
13	541 416 373 **320** 68	A5
541 416 373 320 68	13	A6
<BAG>	<BAG>	Big-Bag-B
355 19	**359** 19	B1
196 288 121 13 541	416	B2
416	373	B3
373	**325**	B4
320	68	B5
68	196 **285** 121 13 541	B6

Table 2. Stepwise fragment assembly of pUC57 with error.

Node No	CurrBag	S	UA	UB	Options	SelBag	Choices	*Cflag*
1	A1	{357}	A2 A3 A4 A5 A6	B1 B2 B3 B4 B5 B6	{B1}	{B1}	{}	0
2	B1	{19}	A2 A3 A4 A5 A6	B2 B3 B4 B5 B6	{A2}	{A2}	{}	0
3	A2	{196}	A3 A4 A5 A6	B2 B3 B4 B5 B6	{B6}	{B6}	{}	0
4	B6	{286 13 541 121}	A3 A4 A5 A6	B2 B3 B4 B5	{A3 A6 A5 A4}	{A3}	{A6 A5 A4}	1
5	A3	{13 541 121}	A4 A5 A6	B2 B3 B4 B5	{A6 A5 A4}	{A6}	{A5 A4}	1
6	A6	{541 121}	A4 A5	B2 B3 B4 B5	{A5 A4}	{A5}	{A4}	1
7	A5	{373 68 322 121 416}	A4	B2 B3 B4 B5	{B3 B5 B4 B2}	{B3}	{B5 B4 B2}	1
8	B3	{68 322 121 416}	A4	B2 B4 B5	{B5 B4 B2}	{B5}	{B4 B2}	1
9	B5	{322 121 416}	A4	B2 B4	{B4 B2}	{B4}	{B2}	1
10	B4	{121 416}	A4	B2	{B4 B2}	{A4}	{B2}	1
11	A4	{416}		B2	{B2}	{B2}	{}	0

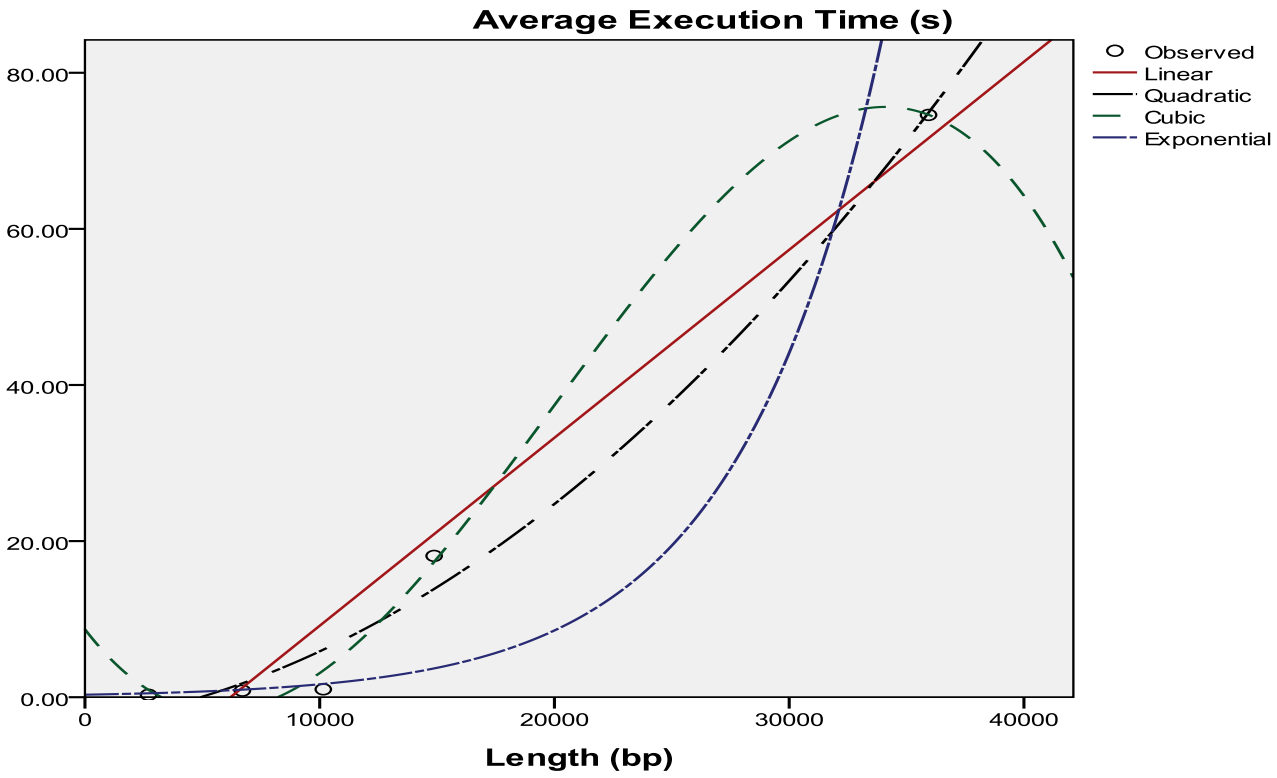

Figure 3. Regression model for error free data, using execution time as dependent variable and length of sequence as predictor

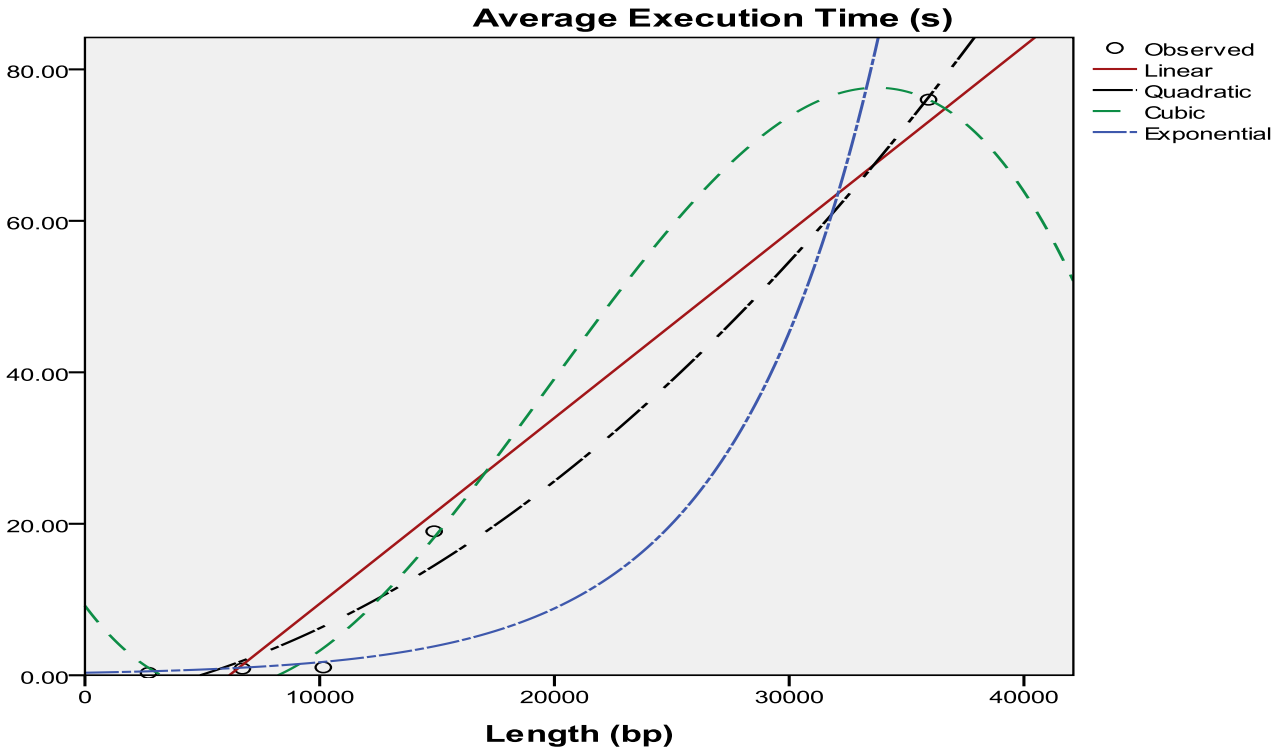

Figure 4. Regression model for erroneous data, using execution time as dependent variable and length of sequence as predictor.

6. REFERENCES

[1] Gillett, W., Hanks, L., Wong, G.K., Yu, J., Lim, R., and Olson, M.V. 1996. Assembly of high-resolution restriction maps based on multiple complete digests of a redundant set of overlapping clones. *Genomics*, 33, 3 (May 1996), pp. 389-408.

[2] Green, E. D. and Green, P. 1991. Sequence-tagged site (STS) content mapping of human chromosomes: Theoretical considerations and early experiences. *PCR Methods Appl*., 1, 2 (November 1991), pp. 77-90.

[3] Kanellakis, P. C., Kuper, G. M. and Revesz, P. Z. 1995. Constraint query languages. *Journal of Computer and System Sciences*, 51, 1 (August 1995) 26-52. DOI= http://dx.doi.org/10.1006/jcss.1995.1051

[4] Lander, E S and M L Waterman., M. L. 1988. Genomic mapping by fingerprinting random clones: A mathematical analysis. Genomics, 2, 3 (April 1988), pp. 231-239.

[5] Olson, M. V., Dutchik, J. E., Graham, M. Y., Brodeur, G. M., Helms, C., Frank, M., MacCollin, M., Scheinman, R., Frank, T. 1986. Random-clone strategy for genomic restriction mapping in yeast. Proceedings of the National Academy of Sciences of the USA, 83, 20 (October 1986), pp. 7826-7830.

[6] Pevzner, P A. 2000. Computational Molecular Biology: An Algorithmic Approach. Bradford Book.

[7] Ramanathan, V and Revesz, P. Z. 2004. Constraint database solutions to the genome map assembly problem. in *Proceedings of the 1st International Symposium on Constraint Databases*, Springer LNCS 3074, 2004, pp. 88-111.

[8] Revesz, P. Z. 1997. On the semantics of arbitration. *International Journal of Algebra and Computation*, 7, 2 (April 1997), 133-160. DOI= http://dx.doi.org/10.1142/S0218196797000095

[9] Revesz, P Z. 1997. Refining Restriction Enzymes Genome Maps. *Constraints*, 2, 1997, pp. 361-375.

[10] Revesz, P. Z. 2010. *Introduction to Databases: From Biological to Spatio-Temporal*, Springer, New York, NY.

[11] Revesz, P. Z. 2013. An algorithm for constructing hypothetical evolutionary trees using common mutations similarity matrices. in *Proceedings of the 4th ACM International Conference on Bioinformatics and Computational Biology,* ACM Press, pp. 731-734.

[12] Revesz, P. Z. and Triplet, T. 2010. Classification integration and reclassification using constraint databases. *Artificial Intelligence in Medicine*, 49, 2 (June 2010), pp. 79-91. DOI= http://dx.doi.org/10.1016/j.artmed.2010.02.003

[13] Triplet T., Shortridge, M. Griep, M., Stark J., Powers R., and Revesz, P.Z. 2010. PROFESS: A protein function, evolution, structure and sequence database. *Database -- The Journal of Biological Databases and Curation,* 2010, DOI= 10.1093/baq011

[14] Setubal, C and J Meidanis. 1997. *Introduction to Computational Molecular Biology*. Brooks/Cole Publishing Company, Pacific Groce, CA.

[15] Shortridge, M.,Triplet, T., Revesz, P. Z., Griep, M., Powers, R. 2011. Bacterial protein structures reveal phylum dependent divergence, Computational Biology and Chemistry, 35, 1 (2011), pp. 24-33.

[16] Siegel, A. F., Roach, J. C., Magness, C., Thayer, E., and van den Engh, G. 1988. Optimization of restriction fragment DNA mapping. *Journal of Computational Biology*, 5, 1 (1998), pp.113-126.

[17] Tisdall, J. *Beginning Perl for Bioinformatics*. O'Reilly Media, Sebastopol, CA, 2001.

[18] Wong, G K, et al. "Multiple-complete-digest restriction fragment mapping:Generating sequence-ready maps for large-scale DNA sequencing." (1997): 5225-5230.