Fast Virus and Bacteria Genome Sequencing by Compatible Restriction Enzyme Fingerprinting

Peter Z. Revesz and Dipty Singh

Abstract— Early identification of a dangerous strain of a virus or bacteria that may cause a pandemic requires practical methods for sequencing of their genomes. This paper describes the concept of compatible restriction enzymes, and a fast and cost-efficient genome map assembly and sequencing method. Computer experiments on plasmid and virus genomes show that the genome map assembly and sequencing can be done in an approximately linear time in the sizes of the genomes.

Keywords— Bacteria, genome map assembly, plasmid, restriction enzyme, sequencing, virus.

I. Introduction

VIRUSES undergo relatively fast genetic changes by point mutations and genetic recombination or reassortment. The genomes of pathological bacteria can also change rapidly and abruptly often by horizontal gene transfer [17]. These genetic changes can result in new viral or bacterial strains that lead to pandemics [14] that require vigilant monitoring and control. The early identification of new infectious virus and bacterial genomes hosted by the affected patients, together with quarantines, is an important preventive measure for pandemics. This implies a need for a fast and cost-efficient genome sequencing.

Unfortunately the currently available genome sequencing machines can only handle DNA (or RNA) fragments of a couple of thousand base pairs. Due to that limitation, the genome has to be broken into smaller fragments that need to be sequenced separately and then reassembled. Genome sequences are cut into smaller fragments using *restriction enzymes*.

After cutting the genome, the small fragments just float randomly in the solution, and all information about the original order of the fragments is lost. After the separated fragments are sequenced, they have to be somehow arranged and assembled to obtain the original genome sequence.

Peter Z. Revesz is a professor in the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588 USA (phone: 402-472-3488; fax: 402-472-7767; email: revesz@cse.unl.edu). Dipty Singh is a former graduate student at the University of Nebraska-Lincoln and currently works at USDA-NRCS in Fort Collins, Colorado. A preliminary version of this research was presented at the conference [13].

That process is called the *genome map assembly problem (GMAP)*, for which many solutions were proposed [1, 2, 4, 5, 6, 16, 19, 20]. In particular, Revesz [7, 9] proposed an approach based on constraint automata [10] that derive from constraint databases [3]. That method could not handle measurement errors. In this paper we describe a new measurement error-prone method for the genome map assembly problem. The new method is especially fast and effective for virus genomes as shown by computer experiments for viruses with genomes of size up to 35,937 nucleotides.

This paper is organized as follows. Section II describes the basic concepts of constraint automata. Section III describes compatible restriction enzymes. Section IV describes the new genome map assembly method using a constraint automaton. Section V describes the data sources. Section VI describes the experiments and analyzes the results. Finally, Section VII presents some conclusions and future work.

II. Review of Constraint Automata

Constraint automata are used to control the operation of a system, using state variables and conditions on them. 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 a *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.

III. Compatible Restriction Enzymes

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].

When some restriction enzyme is applied to a DNA sequence and the lengths of the resulting fragments are measured, these lengths thus serve as the "*fingerprint*" of the DNA sequence.

We say that two or more restriction enzymes are *compatible* if their cutting sites do not affect each other. That is, they can be applied in any sequence and the set of resulting small fragments is the same. In this paper, when we use several restriction enzymes in sequence, then they are always compatible restriction enzymes.

For example, Table 1 shows three restriction enzymes obtained from the New England Biolab website (https://www.neb.com/tools-and-resources/selection-charts/alphabetized-list-of-recognition-specificities). Clearly, the first and the second restriction enzymes are compatible because they can be applied in any order to result in the same set of cuts.

Table 1 Three sample restriction enzymes.

Restriction Enzyme	**Cutting Site**
BstBI	`TT/CGAA`
ClaI BspDI	`AT/CGAT`
TaqαI	`T/CGA`

If a restriction enzyme *c* cuts all the sites that another restriction enzyme *a* cuts, then *c* is strictly stronger that *a*. In Table 1 the third restriction enzyme is strictly stronger than either the first or the second restriction enzyme. Note that when *c* is strictly stronger than *a*, then *a* and *c* are compatible with each other because using them in any order will result in the same set of fragments.

IV. Genome Map Assembly by Constraint Automata

The genome map assembly problem can be abstracted as an instance of the *big-bag matching problem* [10]. 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 can be 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 [10]. We use the concept of BBM to solve the GMAP.

We modified Revesz's compatible restriction enzyme fingerprinting method [10] to collect fingerprint input data. In our modification instead of using three restriction enzymes only two enzymes, which we refer to as enzyme *a* and enzyme *b*, are used. The data collection method is the following:

1. The original DNA sequence is copied.
2. Restriction enzyme *a* is applied to one copy of the sequence, which creates several fragments of varying lengths.
3. The individual fragments are separated.
4. Restriction enzyme *b* is applied to the separated fragments from step 3, producing sub-fragments.
5. The length of the individual sub-fragments is measured using gel electrophoresis [16].
6. Steps 1-5 are repeated, but restriction enzyme *b* is applied first and restriction enzyme *a* is applied second.

The collected fingerprint data are used input to our constraint automata solution, which is shown in Fig. 1. The automaton is modification of the one proposed by Ramanathan and Revesz [7]. The modification ensures that we can take care of measurement errors.

All the elements from first copy of the DNA are stored in big-bag-A and all the elements from the second copy of the DNA are stored in big-bag-B. 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:* The set of A bags, which have not been used yet.
- *UB:* The set of B bags, which have not been used yet.
- *S:* The 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 the *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 improve the efficiency by making a deterministic, backtracking automaton and also extended it to be able to handle errors. Our new constraint automaton (see Fig. 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 *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:* The difference between the mismatched values from the A and the B bags.
- *ErrorTolerance:* The specified error tolerance value.
- *Length-A:* All elements of *S* that belong to Big-Bag-A.
- *Length-B:* All elements of *S* that belong to Big-Bag-B.
- *UA:* The set of A bags, which have not been used yet.
- *UB:* The set of B bags, which have not been used yet.
- *CurrBag:* Current bag, which is the previous *SelBag*.
- *S:* The set of elements by which either the A or the B bag is currently ahead.
- *Options:* Set of bags from which next bag is chosen.
- *SelBag:* The bag selected from *Options* as the next bag.
- *Choices:* Set of remaining bags, which were not picked from *Options* besides the *SelBag*.
- *Cflag:* 0 if *Choices* is empty; 1 otherwise.

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 fingerprint data shown in Table 2.

Table 2 Big-Bags assuming perfect measurements.

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

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 input data shown in Table 3.

Table 3 Big-Bags with some measurement errors.

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

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 as shown in Table 4.

Table 4 Tests of the approximations on some sample publications.

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

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 whereever it belongs. The elements of the bag in *SelBag* are set to either *Length-A* or *Length-B*. If *Length-A* is greater than *Length-B*, the automaton goes to the *A-ahead* state; otherwise it moves to the *B-ahead* state.

Using Fig. 2, 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 the *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 the *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 the automaton does not find any bag that meets these criteria, then it goes to the *Backtrack* state. Using Fig. 2, if S = “541, 288, 121, 373, 68, 416, 320” and *Options* = “B3, B4, B6, B5” then the leftmost bag is picked and set to *SelBag*.

If the automaton is in the *B-ahead* state, then the next bag is selected from *UA*. A similar procedure to that in step 4 is followed to select the next *SelBag*. Again, if *SelBag* is empty, then the automaton goes to the *Backtrack* state.

If the automaton is in the *Backtrack* state, then it goes back to the last node for which the *Cflag* is set to 1. The idea is to select some other bags from *Option*s beside the one picked initially, which might have led to the *Backtrack* state. Hence the automaton tries to go to a node where there are some other *Choices*. For this reason *Cflag* is used initially to help keep track of *Choices*, in case we need to come back and look for other options.

Next the automaton retrieves the values of all the 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, the automaton 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*, then the automaton goes to the *A-ahead* state; if *Length-B* is greater, then it goes to the *B-ahead* state. Sometimes, Length-A and Length-B can be equal. In that case, the automaton selects one random bag from *UA* and continues.

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

V. Data Sources

The constraint automaton was implemented using Perl, which is commonly used in bioinformatics [19]. In addition, the program was compiled in Eclipse SDK v3.5.2 (see the webpage 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.

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” [16]. 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 15 kbp [16]. 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 proposed 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 [20]. Each DNA sequence is first cut by MvaI, then each individual fragment is again cut by MaeII to obtain sub-fragments. This resulting data for Big-Bag-A is shown in the next table.

Bag	Fragment	Sub-Fragments
A1	355	355
A2	215	19, 196
A3	288	288
A4	121	121
A5	13	13
A6	1709	541, 416, 373, 320, 68

Next, each DNA sequence is cut by MaeII, and then by MvaI to obtain the data for Big-Bag-B as shown in the next table.

Bag	Fragment	Sub-Fragments
B1	374	355, 19
B2	416	416
B3	373	373
B4	320	320
B5	68	68
B6	1159	196, 13, 288, 121, 541

VI. 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 5 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	A5	A3	A4	A6
355	19, 196	13	288	121	541, 416, 373, 320, 68
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 6. 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-2	35937	221	76

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

With erroneous 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 (s)
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 Hinfl(G^ANTC) were applied to the DNA sequence of pB85766. In this case, the

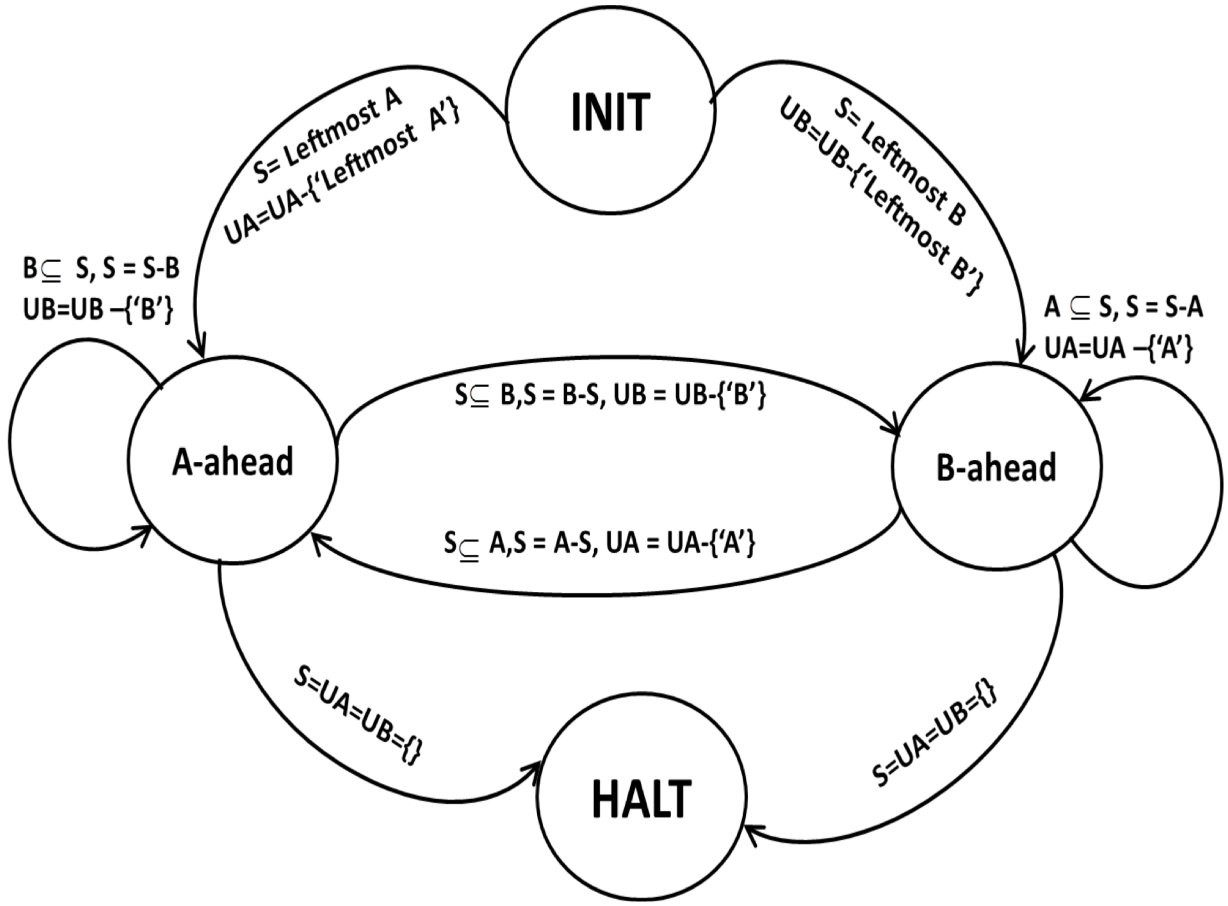

Fig. 1. A non-deterministic constraint automaton for the genome map assembly problem.

number of A and B bags changed and the execution time improved as follows.

Sequence	A Bags	B Bags	Difference	Time (s)
pB85766	44	40	4	3.5

Even though the total number of the bags is almost the same, the difference between the 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.

VII. Conclusions and Future Work

This paper presented a solution to the genome map assembly and sequencing problem using a constraint automaton that allows error tolerance unlike an earlier proposal by Revesz [10]. Our genome map assembly is particularly suitable for viral genomes. In the future it may be possible to automate the process of viral genome map assembly and sequencing by building a machine that implements our algorithm. Such a machine needs to include automated gel electrophoresis and sequencing of small size genomes.

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 information can be combined using arbitration operators [8] and classification integration techniques [12].

Our approach may be also tried for larger genomes such as those of bacteria. Bacterial genomes also suffer considerable evolutionary genetic drift [11, 17]. Horizontal gene transfer plays a significant role in bacterial evolution and is a mechanism by which bacteria can develop resistance to antibiotics. The growing antibiotic resistance by pathogenic bacteria is an important emerging medical problem in hospitals. While the experiments focused on plasmid and viruses, our fast genome sequencing method can likely also help monitor emerging bacterial strains that may be highly antibiotic resistant. The easier monitoring of emerging strains of viruses and bacteria could help control pandemics and save human lives.

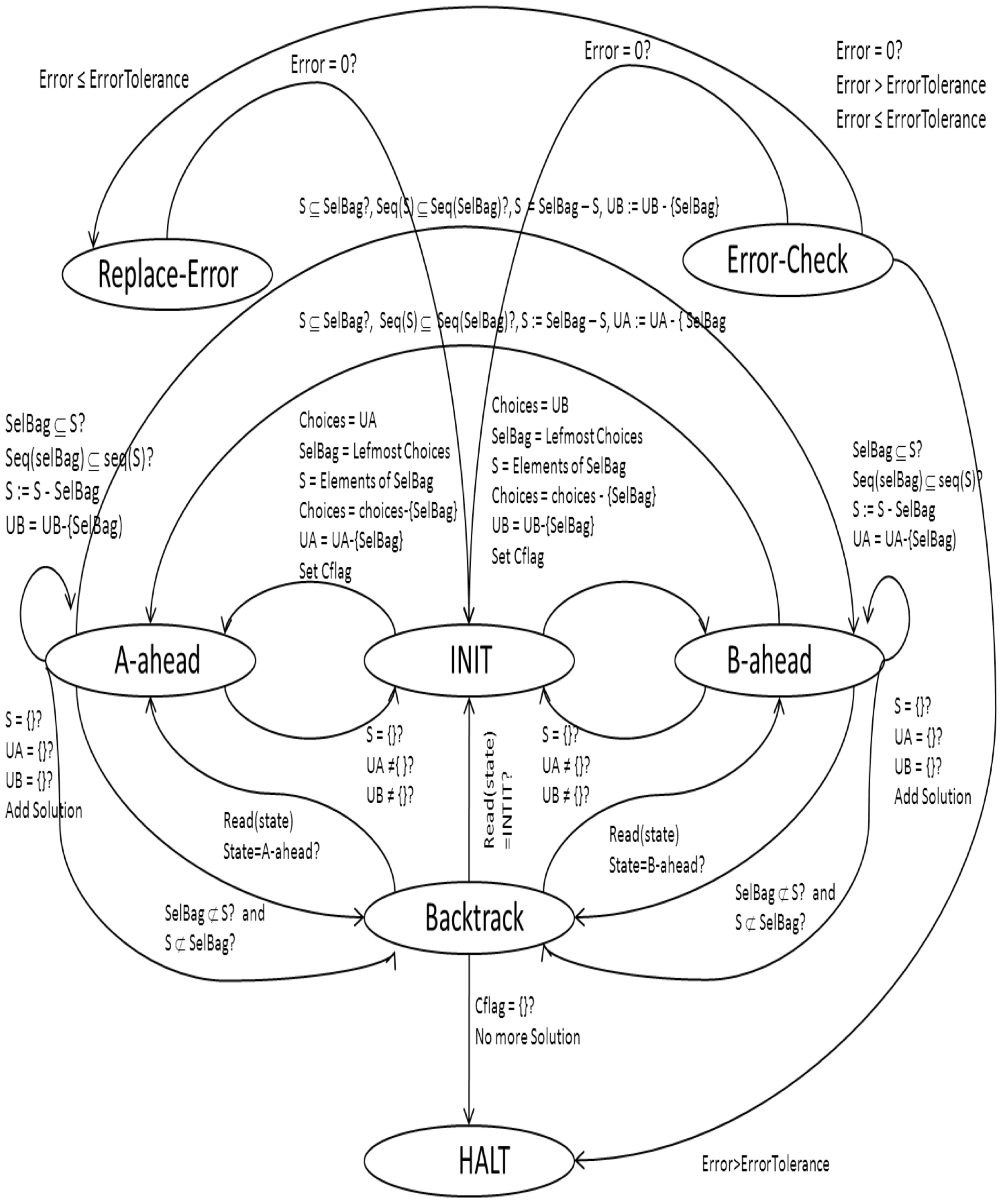

Fig. 2 A deterministic constraint automaton for the genome map assembly problem.

Table 5. 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 6. Stepwise fragment assembly of pUC57 with error.

Node	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

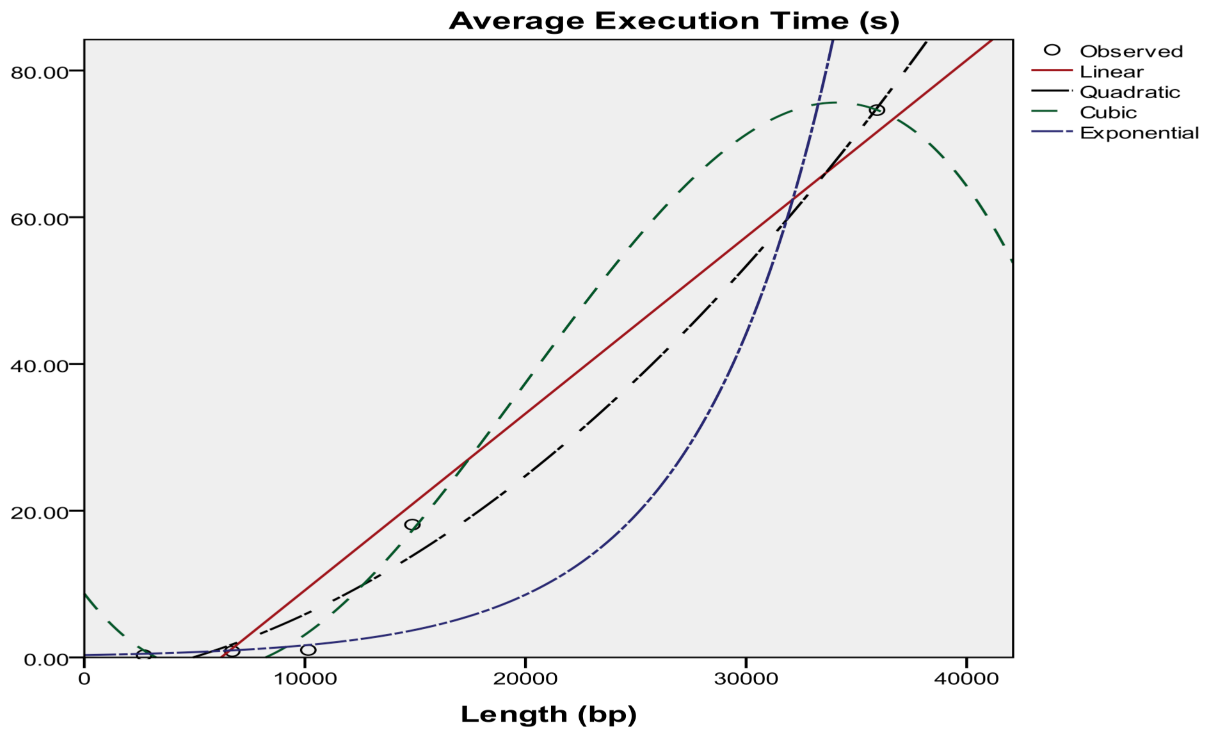

Fig. 3. Regression model for error free data, using execution time as dependent variable and length of sequence as predictor.

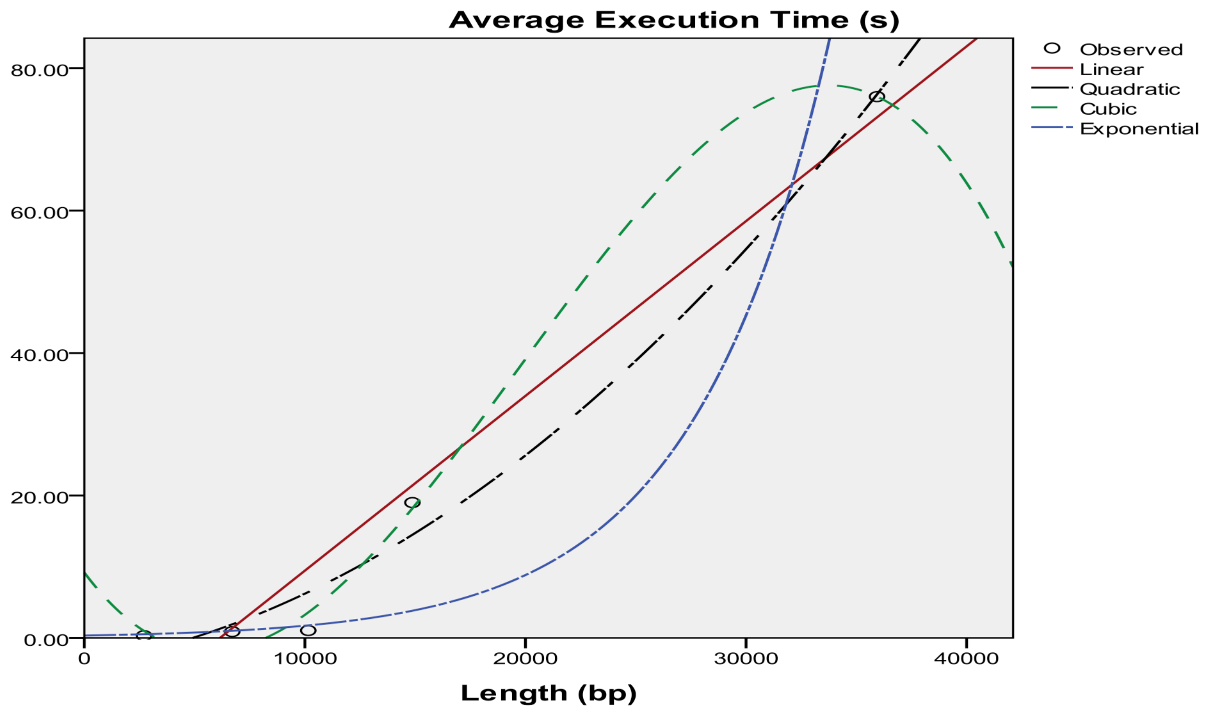

Fig. 4. Regression model for erroneous data, using execution time as dependent variable and length of sequence as predictor.

References

[1] W. Gillett, L. Hanks, G. K. Wong, J. Yu, R. Lim, and M. V. Olson, "Assembly of high-resolution restriction maps based on multiple complete digests of a redundant set of overlapping clones," *Genomics*, vol. 33, no. 3, pp. 389-408, 1996.

[2] E. D. Green, and P. Green, "Sequence-tagged site (STS) content mapping of human chromosomes: Theoretical considerations and early experiences," *PCR Methods and Applications*, vol. 1, no. 2, pp. 77-90, 1991.

[3] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz, "Constraint query languages," *Journal of Computer and System Sciences*, vol. 51, no. 1, pp. 26-52, 1995. Available: http://dx.doi.org/10.1006/jcss.1995.1051

[4] E. S. Lander, and M. L Waterman, "Genomic mapping by fingerprinting random clones: A mathematical analysis," *Genomics*, vol. 2, no. 3, pp. 231-239, 1988.

[5] M. V. Olson, J. E. Dutchik, M. Y. Graham, G. M. Brodeur, C. Helms, M. Frank, M. MacCollin, R. Scheinman, T. Frank, "Random-clone strategy for genomic restriction mapping in yeast," *Proceedings of the National Academy of Sciences of the USA*, vol. 83, no. 20, pp. 7826-7830, 1986.

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

[7] V. Ramanathan, and P. Z. Revesz, "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] P. Z. Revesz, "On the semantics of arbitration," *International Journal of Algebra and Computation*, vol. 7, no. 2, pp. 133-160, 1997. Available: http://dx.doi.org/10.1142/S0218196797000095

[9] P. Z. Revesz, "Refining Restriction Enzyme Genome Maps," *Constraints*, vol. 2, no. 3-4, pp. 361-375, 1997.

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

[11] P. Z. Revesz, "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, 2013.

[12] P. Z. Revesz and T. Triplet, "Classification integration and reclassification using constraint databases," *Artificial Intelligence in Medicine,* vol. 49, no. 2, pp. 79-91, 2010. Available: http://dx.doi.org/10.1016/j.artmed.2010.02.003

[13] P. Z. Revesz, D. Singh, "Efficient and robust constraint automaton-based genome map assembly," in *Proc. 4th International C* Conference on Computer Science and Software Engineering*, ACM Press, no. 9, pp. 1-9, Montreal, Canada, August 2014.

[14] P. Z. Revesz and S. Wu, "Spatiotemporal reasoning about epidemiological data," *Art. Int. in Medicine*, vol. 38, no. 2, pp. 157-170, 2006. Available: http://dx.doi.org/10.1016/j.artmed.2006.05.001

[15] T. Triplet, M. Shortridge, M. Griep, J. Stark, R. Powers, and P. Z. Revesz, "PROFESS: A protein function, evolution, structure and sequence database," *Database - The Journal of Biological Databases and Curation, DOI*=10.1093/baq011, 2010. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2911846/#lpo=2.6158

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

[17] M. Shortridge, T. Triplet, P. Z. Revesz, M. Griep, and R. Powers, "Bacterial protein structures reveal phylum dependent divergence," Computational Biology and *Chemistry*, vol. 35, no. 1, pp. 24-33, 2011.

[18] A. F. Siegel, J. C. Roach, C. Magness, E. Thayer, and G. van den Engh, "Optimization of restriction fragment DNA mapping," *Journal of Computational Biology*, vol. 5, no. 1, pp.113-126, 1998.

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

[20] G. K. Wong, J. Yu, E. C. Thayer, M. V. Olson, "Multiple-complete-digest restriction fragment mapping: Generating sequence-ready maps for large-scale DNA sequencing," *Proceedings of the National Academy of Sciences of the USA*, vol. 94, no. 10, pp. 5225-5230, 1997.

Peter Z. Revesz (Ph.D.'91) holds a Ph.D. degree in Computer Science from Brown University and was a postdoctoral fellow at the University of Toronto.

He is an expert in databases, data mining, big data analytics and bioinformatics. He is the author of *Introduction to Databases: From Biological to Spatio-Temporal* (Springer, 2010) and *Introduction to Constraint Databases* (Springer, 2002). He is currently a professor in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln, Lincoln, NE 6815, USA.

Dr. Revesz also held visiting appointments at the Aquincum Institute of Technology, the IBM T. J. Watson Research Center, INRIA, the Max Planck Institute for Computer Science, the University of Athens, the University of Hasselt, the U.S. Air Force Office of Scientific Research and the U.S. Department of State. He is a recipient of an AAAS Science & Technology Policy Fellowship, a J. William Fulbright Scholarship, an Alexander von Humboldt Research Fellowship, a Jefferson Science Fellowship, a National Science Foundation CAREER award, and a "Faculty International Scholar of the Year" award by *Phi Beta Delta*, the Honor Society for International Scholars.

Dipty Singh (M.S.'11) earned a M.S. degree in Computer Science at the University of Nebraska-Lincoln. She is currently working as a build & release engineer at the USDA-NRCS in Fort Collins, Colorado.