12-2003

# EXPERIMENTAL EVALUATION OF CONSTRAINT AUTOMATA SOLUTIONS TO THE GENOME MAP ASSEMBLY PROBLEM

Viswanathan Ramanathan
*University of Nebraska - Lincoln*, ramanath@cse.unl.edu

# EXPERIMENTAL EVALUATION OF CONSTRAINT AUTOMATA SOLUTIONS TO THE GENOME MAP ASSEMBLY PROBLEM

by

Viswanathan Ramanathan

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Peter Z. Revesz

Lincoln, Nebraska

December, 2003

# EXPERIMENTAL EVALUATION OF CONSTRAINT AUTOMATA SOLUTIONS TO THE GENOME MAP ASSEMBLY PROBLEM

Viswanathan Ramanathan, M.S.

University of Nebraska, 2003

Advisor: Peter Z. Revesz

DNA sequences are really huge having a length of around 3-4 million base pairs. Hence, each DNA sequence has to be cut down into small fragments using restriction enzymes. Once analyzed, these fragments have to be arranged or assembled into a single set of sequences called a genome map, to obtain the original DNA sequence. This problem is called the Genome Map Assembly Problem.

A Constraint-Automata Solution was proposed for this purpose [11]. This thesis improves and implements the Constraint-Automata Solution to find all the possible solutions. The Modified Constraint-Automata Solution was implemented in Perl and executed on parts of various chromosomes of the human DNA. We show that the constraint automaton has a linear execution time.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The thesis addresses the Genome Map Assembly Problem [11] using a Constraint-Automata Solution [12]. The following chapter outlines the definition of a constraint automaton with the help of an illustration. Key concepts relevant to understanding the Genome Map Assembly Problem are then explained.

## 1.1 Constraint Automata

*Constraint automata* are used to control the operation of systems based on conditions that are described using constraints on variables [12]. An important problem for constraint automata is to find the set of *reachable configurations*, which is the set of states and state values that a constraint automaton can enter.

A constraint automaton consists of a set of *states*, a set of state *variables*, *transitions* between states, an *initial state*, and the domain and initial values of the state variables [12]. Each transition consists of a set of constraints, called the *guard* constraints, followed by a set of assignment statements. In constraint automata the guards

are followed by question marks (?), and the assignment statements are shown using the symbol ": =".

A constraint automaton can move from one state to another if there is a transition whose guard constraints are satisfied by the current values of the state variables. The transitions of a constraint automaton may contain variables in addition to state variables. These variables are said to be *existentially quantified* variables, which means that some values for these variables must be found such that the guard constraints are satisfied and the transition can be applied.

A constraint automaton can interact with its environment by sensing the current value of a variable. This is expressed by a read(x) command on a transition between states, where *x* is any variable. This command updates the value of *x* to a new value.

### 1.1.1 Illustration of a Constraint Automaton

We illustrate a constraint automaton by an example taken from Revesz [12]. A hotel elevator can move from a lobby up to a restaurant or down to an underground parking garage with no other intermediate stops. This elevator can sense the number of people waiting in front of the elevator on each floor, the up and down signals pushed in the lobby, and the floor request signals pushed within the elevator. (There are no up and down signals in the restaurant or the parking garage.)

The elevator does not always go all the way up and down. For instance, the elevator while going up from the parking garage to the lobby – if it senses no request for the restaurant from either within the elevator or from the lobby and also senses that there

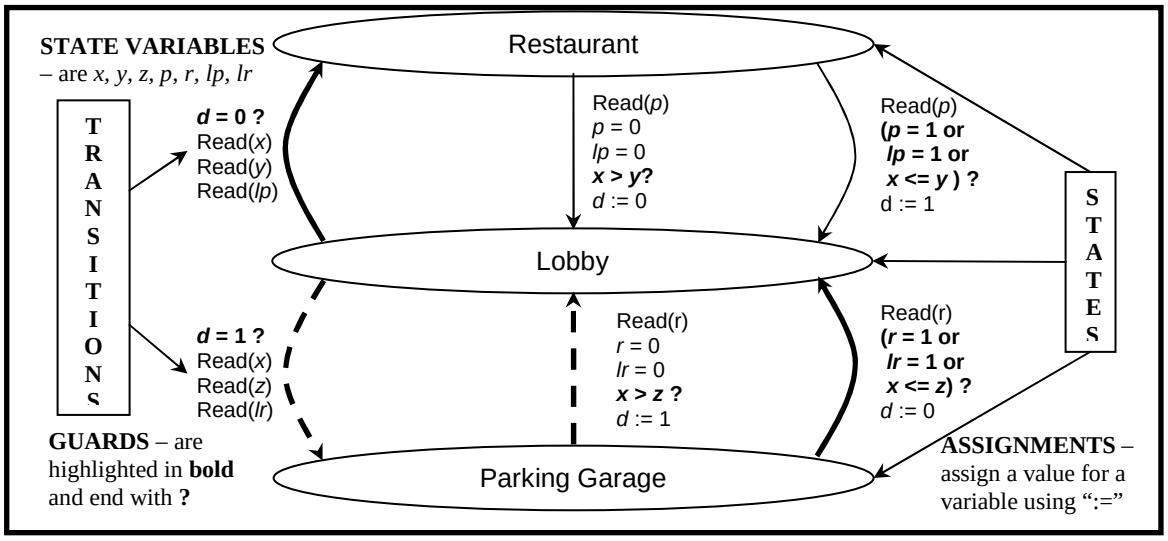


Figure 1.1: Constraint Automata for an Elevator System [12]

are more people waiting for it in the lobby than in the restaurant, then it will reach the lobby and go back down to the parking garage. Else, it will go up to the restaurant.

Similarly, the elevator while going down from the restaurant to the lobby – if it senses no request for the parking garage from either within the elevator or from the lobby, and also there are more people waiting for it in the lobby than in the parking garage, then it will reach the lobby and go back up to the restaurant. Else, it will go down to the parking garage.

The above two instances clearly show that the operation of the elevator can be represented using a constraint automaton (Figure 1.1). This constraint automaton has three ***states*** – Restaurant, Lobby, and Parking Garage, each corresponding to and named after a possible position of the elevator. The following are the ***variables*** used:

- *x* - number of people sensed to be waiting for the elevator in the lobby
- *y* - number of people sensed to be waiting for the elevator in the parking garage

- *z* - number of people sensed to be waiting for the elevator in the restaurant
- *d* - Is the next move of the elevator down? YES → *d = 1*, NO → *d = 0*
- *p* - Anyone in the elevator requested parking garage? YES → *p = 1*, NO → *p = 0*
- *r* - Anyone in the elevator requested restaurant? YES → *r = 1*, NO → *r = 0*
- *lp* - Anyone in the lobby pushed the down button? YES → *lp = 1*, NO → *lp = 0*
- *lr* - Anyone in the lobby pushed the up button? YES → *lr = 1*, NO → *lr = 0*

Based on Figure 1.1, let us consider the following situation – the elevator while going up from the parking garage to the lobby – if it senses no request for the restaurant from either within the elevator ($r = 0$) or from the lobby ($lr = 0$), and also senses that there are more people waiting for it in the lobby than in the restaurant ($x > z$), then it will reach the lobby and go back down to the parking garage (this is done by assigning $d := 1$). This entire process is represented by  arrow. Else, it will go up to the restaurant (when either $r = 1$ or $lr = 1$ or $x <= z$ then $d := 0$ is assigned which makes the elevator go to the restaurant). This entire process is represented by  arrow.

## 1.2 Genome Map Assembly Problem

### 1.2.1 Genome

The genome of an organism is its set of chromosomes, containing all of its genes and the associated *deoxyribonucleic acid* (DNA) [4]. It contains the entire set of hereditary

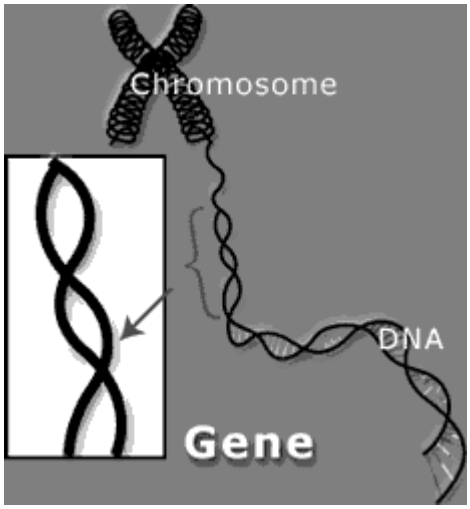


Figure 1.2: Components of a Genome [4]

instructions for building, running and maintaining an organism, and passing life on to the next generation. Figure 1.2 shows the components of a genome.

- *Chromosome* – a threadlike body in the cell nucleus that carries the genes in a linear order.
- *DNA* – a nucleic acid found in the nucleus of a cell and consisting of a polymer formed from nucleotides and shaped like a double helix.
- *Gene* – a segment of DNA found on a chromosome that codes for a particular protein.

### 1.2.2 Deoxyribonucleic Acid (DNA)

DNA is a molecule, a nucleic acid, consisting of nucleotides (Figure 1.3) and shaped like a double helix [1, 17] (Figure1.4). It associated with the transmission of genetic information that is the hereditary material in all living cells [4].

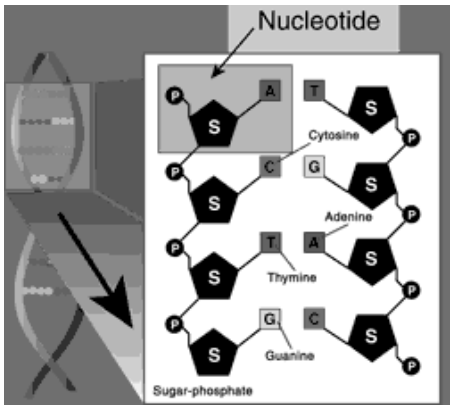


Figure 1.3: Nucleotide [4]

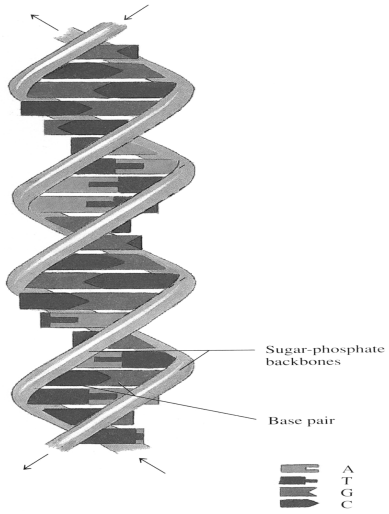


Figure 1.4: Double Helical Structure of a DNA [1]

Each nucleotide (Figure 1.3), the smallest unit of a DNA, has three parts: a sugar molecule (S), a phosphate group (P), and a structure called a nitrogenous base (A, C, G, and T) [1, 4, 17]. The DNA is built on the repeating sugar-phosphate units. The sugars are molecules of *deoxyribose* from which DNA receives its name. Joined to each deoxyribose is one of the four possible nitrogenous bases: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). These bases carry the genetic information, so the words "nucleotide" and "base" are often used interchangeably. They can be arranged in any order along a strand of DNA. The sequence of these bases along a DNA strand constitutes the genetic information.

Since the DNA is a double-stranded helix, the bases on one strand are connected to bases on the other strand to form a base pair. Adenine (A) always pairs with Thymine (T) and Guanine (G) always pairs with Cytosine (C).

### 1.2.3 Genome Sequencing

Genome Sequencing is finding the order of DNA nucleotides, or bases, in a genome – the order of As, Cs, Gs, and Ts that make up an organism's DNA [4]. An example of a genome sequence is given in Figure 1.5.


Figure 1.5: Genome Sequence [4]

As shown above, a genome sequence is simply a very long string of letters (A, C, G, and T) forming a mysterious and complex sequence. Thus, sequencing a genome does not immediately lay open the genetic secrets of an entire species. Scientists still have to translate the complex string of letters to understand: How a genome works? What are the various genes that make up a genome? What do they (genes) do? How are the different genes related? They have to figure out what the letters of the genome sequence mean. Thus, sequencing a genome is an important step towards understanding it.

Scientists also hope that being able to study the entire genome sequence will help them understand how the genome as a whole works – how genes work together to direct growth, development and maintenance of an organism. Genes account for less than 25 percent of the DNA in the genome, and so knowing the entire genome sequence will help scientists study the parts of the genome outside the genes. This includes the regulatory regions that control how genes are turned on and off, as well as long stretches of “nonsense” or “junk” DNA – so called because it is not yet known what, if anything, it does.

### 1.2.4 Genome Mapping

A genome map helps scientists navigate around the genome. Like road maps, a genome map is a set of landmarks that tells people where they are, and helps them get where they want to go. The landmarks on a genome map may include short DNA sequences, regulatory sites that turn genes on and off, and genes themselves [4]. Often, genome maps help scientists find new genes.

Road maps chart well-known territory surveyed with astonishing precision, but a genome map is a map of a new frontier. In that sense, a genome map is more like the maps of North America made when Europeans were just beginning to explore the continent. Some parts of the genome have been mapped in great detail, while others remain relatively uncharted territory. It may turn out that a few landmarks on current genome maps appear in the wrong place or at the wrong distance from other landmarks. But over time, as scientists continue to explore the genome frontier, maps will become more accurate and more detailed. So, a genome map is a work in progress.

### 1.2.5 Genome Sequencing vs. Genome Mapping

A genome map is less detailed than a genome sequence [4]. A sequence spells out the order of every DNA base in the genome, while a map simply identifies a series of landmarks in the genome. The landmarks on a map are DNA sequences, and mapping is the cousin of sequencing. For example, consider a genome sequence [Figure 1.6(a)] and its corresponding genome map [Figure 1.6(b)].


Figure 1.6(a): Genome Sequence

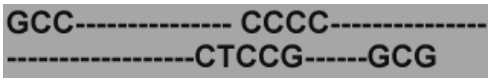


Figure 1.6(b): Genome Map

In the genome map, GCC is one landmark; CCCC is another. In the corresponding sequence, each base (A, C, G, or T) is a landmark. In other words, the sequence is simply the most detailed possible map. For large genomes, creating a reasonably comprehensive genome map is quicker and cheaper than sequencing the entire genome. Simply put, mapping involves less information to collect and organize than sequencing does.

### 1.2.6 Advantages of a Genome Map

Several advantages of a genome map have been identified [4].

1. A map can help in sequencing the genome. The more detailed and accurate a map, the easier it is to snap pieces of a genomic jigsaw puzzle into place to obtain its sequence.
2. A map aids in understanding the genome sequence. A sequence is pretty much featureless: just a long string of DNA bases or “letters”. For the most part, scientists can not look at a sequence and see immediately which parts are genes or other interesting features, and which parts are “junk”. But the landmarks on a genome map provide clues about where the important parts of the genome sequence can be found, thus aiding in understanding the sequence better.
3. A map helps scientists find new genes.
4. A map enables scientists to compare the genomes of different species, yielding insights into the process of evolution.

5. A map also enables scientists to easily modify genes to obtain genetically modified plants, find cures for diseases and many other practical applications which aid in the development of the human race.

### 1.2.7 Restriction Enzymes

Restriction Enzymes are precise, molecular scalpels that allow an investigator (scientist) to manipulate DNA segments. Restriction Enzymes, also called *restriction endonucleases*, which recognize specific base sequences in double-helical DNA and cleave, at specific places, both strands of a duplex containing the recognized sequences [1]. They are indispensable tools for analyzing chromosome structure, sequencing very long DNA molecules, isolating genes, and creating new DNA molecules that can be cloned.

Werner Arber and Hamilton Smith discovered restriction enzymes, and Daniel Nathans pioneered their use in the late 1960s [1]. Many restriction enzymes recognize specific sequences of four to eight base pairs. More than 100 restriction enzymes have been purified and characterized. Their names consist of a three-letter abbreviation (e.g., *Eco* for *Escherichia coli, Hin* for *Haemophilus influenzae, Hae* for *Haemophilus aegyptius*) followed by a strain designation (if needed) and a roman numeral (if more than one restriction enzyme from the same strain has been identified).

Restriction enzymes are used to cleave DNA molecules into specific fragments that are more readily analyzed and manipulated than the entire parent molecule. For example, a 5.1-kb circular duplex DNA of a tumor-producing SV40 virus is cleaved at 1

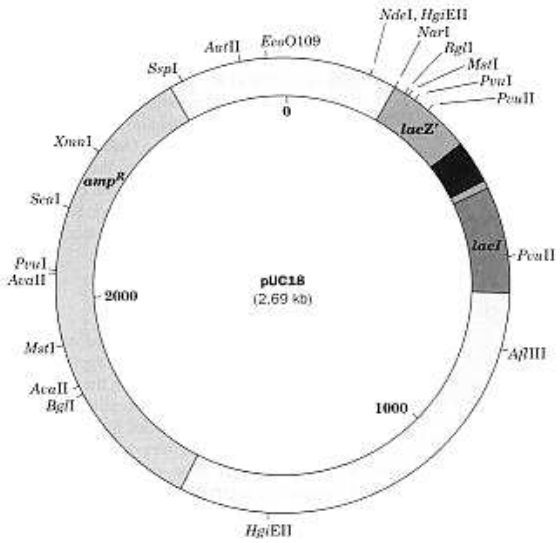


Figure 1.7: Restriction Map of plasmid pUC18 [17]

site by *Eco*RI, 4 sites by *Hpa*I, and 11 sites by *Hin*dIII. A piece of DNA produced by the action of one restriction enzyme can be specifically cleaved into smaller fragments by another restriction enzyme. The pattern of such fragments, called a restriction map, can serve as a fingerprint of a DNA molecule. Figure 1.7 shows the restriction map of the plasmid pUC18 indicating the positions of its $amp^R$, *lacZ'*, and *lacI* genes.

### 1.2.8 Genome Map Assembly

DNA sequences are really huge having a length of around 3-4 million base fragments. But current sequencing machines can not sequence DNA fragments of length more than a couple of thousand base fragments. So the DNA sequences have to be cut down into small fragments using some kind of a restriction enzyme. Once the DNA sequence is fragmented, all the information about the original order is lost. Hence these fragments

have to be arranged or assembled into a single set of sequence called a genome map, to obtain the original sequence. This process is called *Genome Map Assembly* [7]. The problem of executing the Genome Map Assembly process is called the *Genome Map Assembly Problem (GMAP)* [11].

## 1.3 Genome Map Assembly using Constraint Automata

In a genome map assembly, various fragments that are obtained by cutting a DNA have to be assembled in an order that would give us the original sequence. It is possible to undertake this process using a constraint automaton. To achieve this, an abstraction of the Genome Map Assembly Problem is used. This abstraction, called Big-Bag Matching Problem[11], is NP-complete [13] and can be solved with the help of a constraint automaton, the Constraint-Automata Solution [12]. It has been proved that this automaton is NP-complete, which means that it has an exponential time complexity (time needed by an algorithm expressed as a function of the size of a problem) in the worst case [11]. Several variations of the Genome Map Assembly Problem are shown to be NP-complete [9].

This thesis implements the Genome Map Assembly automaton and empirically investigates whether the time complexity for the average case is linear or exponential. A key concept of “backtracking” used to modify the existing automaton, will be proposed and implemented in Perl.

# Chapter 2

# Constraint Automata Solution

The Genome Map Assembly Problem (GMAP) is addressed using an abstraction of the GMAP, the Big-Bag Matching Problem (BBMP), and a Constraint-Automata Solution for the BBMP. The Constraint-Automata Solution proposed by Revesz [11] is modified to increase its applicability.

## 2.1 Big-Bag Matching Problem

An abstraction of the Genome Map Assembly Problem, following Revesz [11], with new examples and discussion, is explained in this section.

A *bag* is a multiset, a generalization of a set in which each element can occur multiple times [11]. A *big-bag* is a multiset whose elements are bags that can occur multiple times [11]. Each permutation of the bags and permutation of the elements of each bag within a big-bag is called a *presentation* [11]. A big-bag can have several different presentations.

For example, we have big-bag A = {[1, 3], [2, 4]}. Following are the various presentations of A:

- $a_1$ = {[1, 3], [2, 4]};
- $a_2$ = {[3, 1], [2, 4]};
- $a_3$ = {[1, 3], [4, 2]};
- $a_4$ = {[3, 1], [4, 2]};
- $a_5$ = {[2, 4], [1, 3]};
- $a_6$ = {[2, 4], [3, 1]};
- $a_7$ = {[4, 2], [1, 3]}; and
- $a_8$ = {[4, 2], [3, 1]}.

The $i^{th}$ element of a presentation $a$, written as $a$[i], is the $i^{th}$ symbol seen when the presentation is read from left to right. For example, in $a_1$ the third element is 2 and the fourth element in $a_6$ is 1, i.e., $a_1[3] = 2$ and $a_6[4] = 1$.

Two big-bags A and B match if there are presentations $a$ for A and $b$ for B such that $a$[i] = $b$[i] for each $1 \leq i \leq n$. Suppose big-bag A = {[1, 3], [2, 4], [5, 7, 8]} and big-bag B = {[1, 2, 4, 8], [3], [5, 7]}. $A$ and $B$ match because $A$ can be presented as $a_1$ = {[3, 1], [2, 4], [8, 5, 7]} and $B$ can be presented as $b_1$ = {[3], [1, 2, 4, 8], [5, 7]} and $a_1$[i] = $b_1$[i] for $1 \leq i \leq n$ where $n = 7$.

The *big-bag matching decision problem* (BBMD) is the problem of deciding whether two big-bags match [11]. The *big-bag matching problem* (BBM) is the problem of finding matching presentations for two given big-bags if they match [11].

## 2.2 The Genome Map Assembly Problem

To define the Genome Map Assembly Problem (GMAP) we have to know how to get the input data for the problem. The following is a procedure, taken from Revesz [11], which uses three restriction enzymes a, b, and $c$ for obtaining the input data:

1. Take a copy of the DNA.

2. Apply restriction enzyme *a* to the copy.
3. Separate the fragments.
4. For each fragment apply restriction enzyme *b* $\cup$ *c*, cutting the fragment into sub-fragments.
5. Find the lengths of the subfragments.
6. Repeat Steps 1-5 using *b* instead of *a* and *a* $\cup$ *c* instead of *b* $\cup$ *c*.

Each execution of Steps 1-5 is the generation of a big-bag. The fragments obtained in step 3 are the bags of the big-bag, and the subfragments obtained in step 4 are the elements of the bags.

We use *Lambda* to explain the procedure for obtaining input to the GMAP. Lambda is a bacteriophage, a virus that infects bacteria [21]. It was first isolated from E. coli (Escherichia coli), a common bacterium that has been studied intensively by geneticists because it has a small genome and is usually harmless and easy to grow. The genetic material of Lambda consists of a double-stranded DNA molecule. It is used as a cloning vector, accommodating fragments of DNA up to 15,000 base pairs long. The following 156 base sequence is a part of one of the two strands of the complete double-helical Lambda sequence whose actual length is 48502 base pairs [20]:

CATCGATCTCGGGAGGGATCCATTATCGATTCCCGGGCTCGGGGGATCCTTCC
ATCGATGGGCCCGAGGCGGATCCCTACTATCGATCCCGGGGGGATCCTTAAT
TCTCGAGAAGGCCTATCGATCAAGGATCCTATCGATCCCGAGTCCCGGGAT

where A, C, G, and T, are the nucleotides. We apply the following three restriction enzymes [26] on the above DNA sequence:

- ClaI (AT^CGAT)[1] henceforth represented as restriction enzyme *a*.
- BamHI (G^GATCC) henceforth represented as restriction enzyme *b*,
- AvaI (C^YCGRG) henceforth represented as restriction enzyme *c*, where Y = C or T and R = A or G. Hence AvaI's sequence is recognized as any one of the following sequences CCCGGG, or CCCGAG, or CTCGGG, or CTCGAG.

After applying restriction enzyme *a*, the given sequence is fragmented into the following bags:

A1: CAT
A2: CGATCTCGGGAGGGATCCATTAT
A3: CGATTCCCGGGCTCGGGGGATCCTTCCAT
A4: CGATGGGCCCGAGGCGGATCCCTACTAT
A5: CGATCCCGGGGGGATCCTTAATTCTCGAGAAGGCCTAT
A6: CGATCAAGGATCCTAT
A7: CGATCCCGAGTCCCGGGAT

Next we apply the restriction enzymes $b \cup c$ on each of the above bags (fragments) to obtain the following result:

| | | |
|---|---|---|
| A1 | [CAT] | [A1{3}] |
| A2 | [CGATC] [TCGGGAGG] [GATCCATTAT] | [A2{5,8,10}] |
| A3 | [CGATTC] [CCGGGC] [TCGGGG] [GATCCTTCCAT] | [A3{6,6,6,11}] |
| A4 | [CGATGGGC] [CCGAGGCG] [GATCCCTACTAT] | [A4{8,8,12}] |
| A5 | [CGATC] [CCGGGGG] [GATCCTTAATTC] [TCGAGAAGGCCTAT] | [A5{5,7,12,14}] |
| A6 | [CGATCAAG] [GATCCTAT] | [A6{8,8}] |
| A7 | [CGATC] [CCGAGTC] [CCGGGAT] | [A7{5,7,7}] |

The above table, which is the big-bag *A*, shows the bags, the DNA subfragments in each bag and their corresponding lengths in each of the column. In a similar fashion we can obtain the following big-bags *B* and *C*.

[1] " ^ " is the site where the restriction enzyme would cut the DNA.

| | |
|---|---|
| [B1{3,5,8}]<br>[B2{10,6,6,6}]<br>[B3{11,8,8}]<br>[B4{12,5,7}]<br>[B5{12,14,8}]<br>[B6{8,5,7,7}] | [C1{3,5}]<br>[C2{8,10,6}]<br>[C3{6}]<br>[C4{6,11,8}]<br>[C5{8,12,5}]<br>[C6{7,12}]<br>[C7{14,8,8,5}]<br>[C8{7}]<br>[C9{7}] |

For the complete step-by-step process, refer to Appendix A.

Once the DNA sequence is fragmented into big-bags using the above procedure, all the information about the original order is lost. After analyzing these fragments, they have to be arranged or assembled into a single set of sequences called genome map.

## 2.3 Constraint Automaton

This section presents an algorithm, taken from Revesz [11], for solving the big-bag matching problem. The big-bag matching problem is an abstraction of the Genome Assembly Map Problem. It is solved by the constraint automaton shown in Figure 2.1. The automaton uses the constraints "⊆" (subset) and "–" (set difference).

This section presents an algorithm, taken from Revesz [3], for solving the big-bag matching problem. The big-bag matching problem is an abstraction of the Genome Assembly Map Problem. It is solved by the constraint automaton shown in Figure 2.1. The automaton uses the constraints "⊆" (subset) and "–" (set difference).

The automaton starts in the INIT state and ends in the HALT state. From the INIT state, the automaton moves from left to right by adding either bag A or B. Each bag represents a *contiguity constraint* for the elements it contains, i.e., any valid presentation

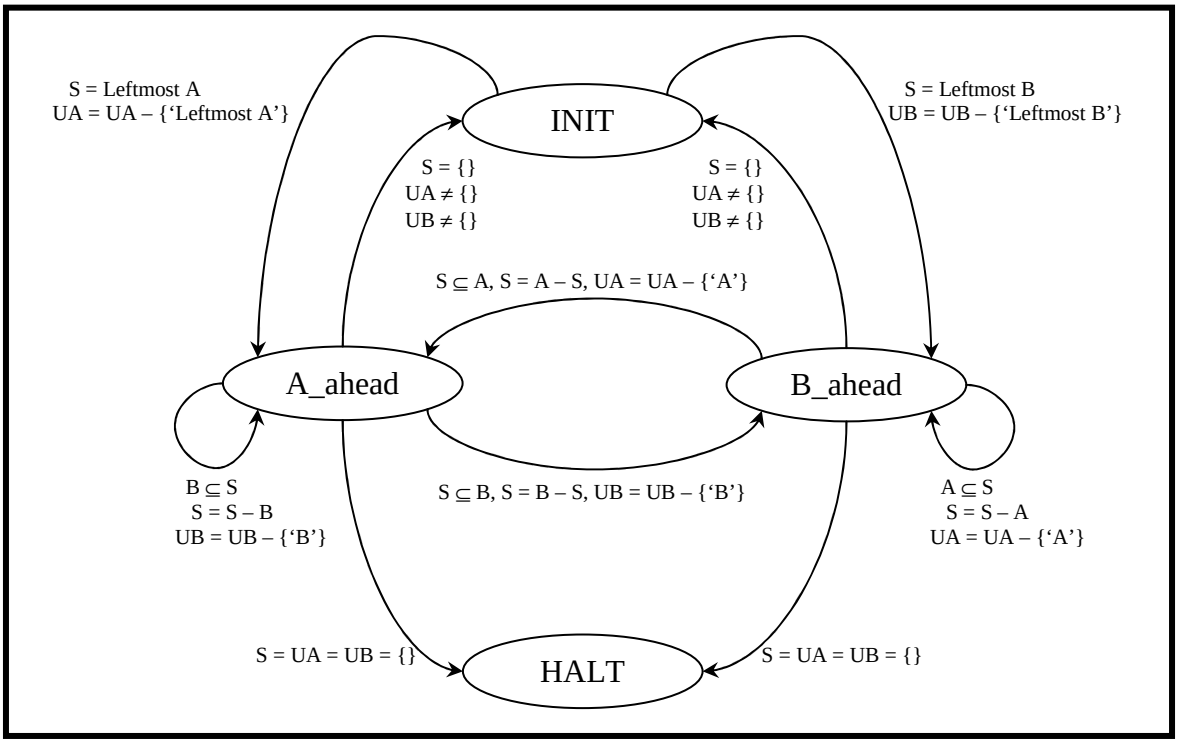


Figure 2.1: Constraint Automaton [11]

must contain the elements within a bag next to each other. Therefore, adding a new bag really adds a new contiguity constraint to a set of other such constraints. Only if the set of constraints is solvable (and there could be several solutions) is the automaton allowed to continue with the next transition.

At any point either the list of A bags will be ahead of the list of B bags (and the automaton will be in state “A_ahead”) or vice versa (and it will be in the state “B_ahead”) or neither will be ahead, in which case it goes back to the INIT state and starts the matching algorithm again.

The automaton has the following states: INIT, HALT, A_ahead and B_ahead. The state variables for each state are: *UA* and *UB* indicating the number of unused *A* and *B* bags, and *S* the bag of elements by which either the *A* list or the *B* list is currently ahead.

Whenever there are possible alternatives the automaton makes a guess. If the automaton cannot reach the HALT state after a series of guesses, then it will backtrack and try a new guess.

Assuming the automaton makes only correct guesses and no backtracking, the working of the automaton is explained below.

1. The automaton guesses the first bag and moves from the INIT state to either the "A_ahead" or the "B ahead" state.
2. If the automaton is in the "A_ahead" state, then it guesses the next *B* bag. If the next *B* bag contains *S*, which is the set of elements by which *A* is currently ahead, then the automaton will move to the "B_ahead" state and set the value of *S* to be the value of new bag *B* minus the current value of *S* ($S = B - S$). If the next bag *B* is contained in *S*, then the automaton stays in the "A_ahead" state and subtracts from *S* the value of the new bag ($S = S - B$). In both cases, the elements of *S* and the new bag are matched as far as possible and the new bag is taken out from *UB*.
3. If the automaton is in the "B_ahead" state, then it guesses the next *A* bag. If the next *A* bag contains *S*, which is the set of elements by which *B* is currently ahead, then the automaton will move to the "A_ahead" state and set the value of *S* to be the value of new bag *A* minus the current value of *S* ($S = A - S$). If the next bag *A* is contained in *S*, then the automaton stays in the "B_ahead" state and subtracts

from *S* the value of the new bag (*S* = *S* – *A*). Again, in both cases, the elements of *S* and the new bag are matched as far as possible and the new bag is taken out from *UA*.

4. When all the bags are used and *UA*, *UB*, and *S* are empty, then the automaton enters the HALT state and stops.

## 2.4 Modification of Constraint-Automata Solution

"Backtracking" was incorporated in the Constraint-Automata Solution to get the Modified Constraint-Automata Solution explained in the next section. Table 2.1 outlines how the Modified Constraint-Automata Solution extends the original solution.

Table 2.1: Comparison of original and modified solutions

| **Constraint-Automata Solution [11]** | **Modified Constraint-Automata Solution** |
|---|---|
| Uses correct starting bag | Can use incorrect starting bag |
| One solution | More than one solution possible |
| "Backtracking" not included | Includes "backtracking" |

## 2.5 Constraint Automaton with Backtracking

Figure 2.2 shows a constraint automaton that incorporates backtracking. The working of the automaton is similar to the one shown in Figure 2.1. The constraints used are "⊆" (subset) and "–" (set difference).

The automaton starts in the INIT state and ends in the HALT state. From the INIT state, the automaton moves form left to right by adding either bag A or B. The automaton has the following states: INIT, A_ahead, B_ahead, HALT and Backtrack. The state variables are: *UA* and *UB* indicating the set of unused *A* and *B* bags, respectively, *S* indicating the set of elements by which either the *A* or the *B* list is currently ahead, *Choices* indicating the set of options from which the next bag can be chosen, *SelBag* indicating the bag that was selected from *Choices* as the next bag, and *Cflag* indicating whether *Choices* is empty or not (if empty then it is set to "0", else it is set to "1"). The value of each state variable is saved after each transition.

The working of the automaton can be thought of as a pre-order traversal of a tree, where Node 0 is the INIT state, and every other node in the tree is the state in which the automaton is after a transition. The total number of nodes in the tree is the total number of bags in the big-bags A and B combined. The working of the automaton is explained below.

1. The automaton is in the "INIT" state. If the next bag to be chosen is from the list of *A* bags then *Choices* will contain *UA*, else it will contain *UB*. The leftmost bag of *Choices* is removed from *Choices* and set to *SelBag*, the *Cflag* is set to "0" if *Choices* is empty or "1" if it is not empty, the elements of *SelBag* is set to *S*, and the automaton moves to either the "A_ahead" or the "B_ahead" state.
2. The set of options from which the next bag can be chosen is found by determining the bags which are either a subset or a superset of *S*. *Choices* contains this set of options.

3. If the automaton is in the “A_ahead” state, then the set of options is determined from *UB*. *SelBag* is the leftmost bag in *Choices*. If *S*, which is the set of elements by which *A* is currently ahead, is a subset of *SelBag*, then the automaton will move to the “B_ahead” state and set the value of *S* to be the value of *SelBag* minus the current value of *S* (*S* = *SelBag* – *S*). If *SelBag* is a subset of *S*, then the automaton remains in the “A_ahead” state and subtracts from *S* the value of *SelBag* (*S* = *S* – *SelBag*). In both cases, the elements of *S* and *SelBag* are matched as far as possible.
4. If the automaton is in the “B_ahead” state, then the set of options is determined from *UA*. *SelBag* is the leftmost bag in *Choices*. If *S*, which is the set of elements by which *B* is currently ahead, is a subset of *SelBag*, then the automaton will move to the “A_ahead” state and set the value of *S* to be the value of *SelBag* minus the current value of *S* (*S* = *SelBag* – *S*). If *SelBag* is a subset of *S*, then the automaton remains in the “B_ahead” state and subtracts from *S* the value of *SelBag* (*S* = *S* – *SelBag*). In both cases, the elements of *S* and *SelBag* are matched as far as possible.
5. If, in Steps 3 and 4, the difference of the values of *SelBag* and *S* is an empty set and *UA* and *UB* are **not** empty, then the automaton moves to the “INIT” state. If the difference is an empty set and *UA* and *UB* are also empty, then a solution has been found for the problem and the automaton moves to the “HALT” state and stops.

6. If, in Steps 3 and 4, *SelBag* is neither a subset nor a superset of *S*, then the automaton moves to the “Backtrack” state. In this state, the automaton will check for the last node whose *Cflag* has the value “1”. The automaton then backtracks to this node and the information saved at this node is retrieved. Based on this information, the automaton will move to the “A_ahead”, the “B_ahead” or the “INIT” state. If none of the nodes have their corresponding *Cflag* set to “1”, then there is no solution for the problem and the automaton moves to the “HALT” state and stops.

## 2.6 Constraint Automaton for ALL Solutions

Figure 2.3 shows a constraint automaton that finds all possible solutions for a given problem. The automaton works in the same manner as the one shown in Figure 2.2, but includes additional steps to save the solutions once they are found.

A solution for the problem is found when *S*, *UA*, and *UB* are empty. The automaton then, instead of moving to the “HALT” state, saves the solution and moves to the “Backtrack” state. The execution of the automaton continues until all possible solutions are found, i.e., until the *Cflag*s of all the nodes is set to “0”.

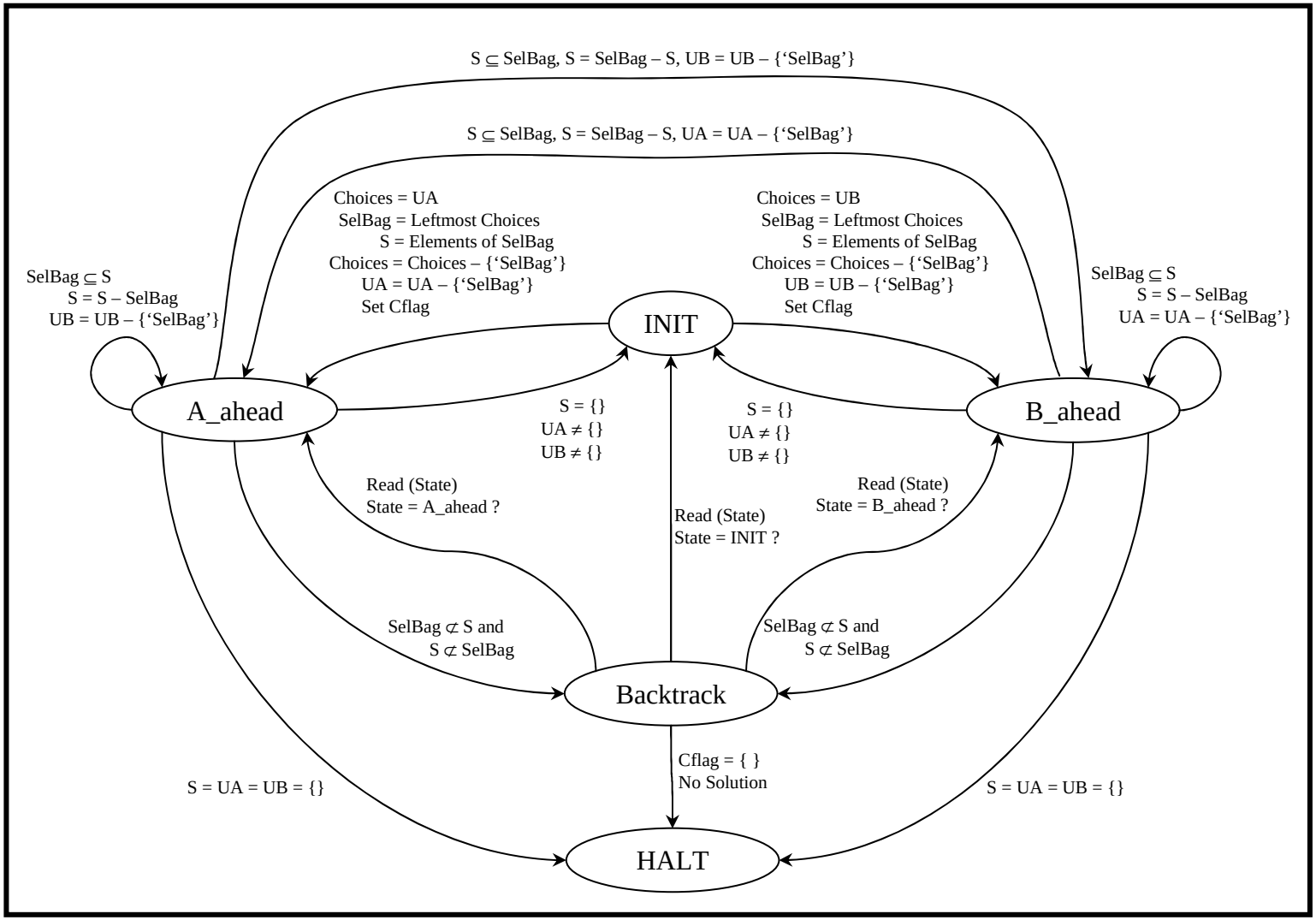


Figure 2.2: Constraint Automaton with Backtracking

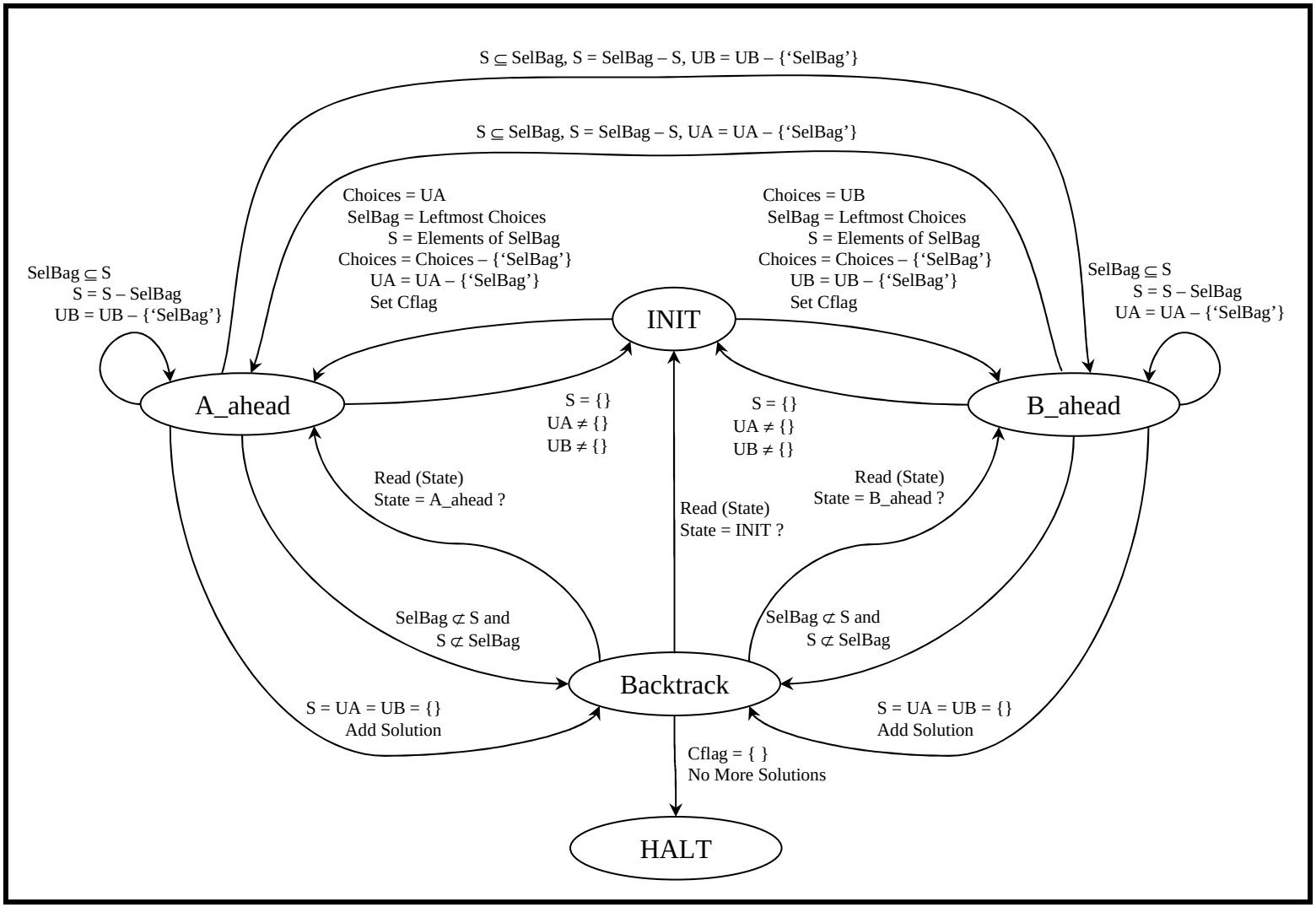


Figure 2.3: Constraint Automaton for ALL solutions

# Chapter 3

# Methodology

To begin with, we generate the input for the constraint automaton by applying restriction enzymes (REs) on a DNA sequence. The generated inputs are then given to the constraint automaton to determine the final order of the bags.

Perl v5.8.0 [25] was used to implement the generation of input and the constraint automaton. The programs were executed in *Cygwin* v1.5.5-1 in Windows XP Professional Edition operating system environment. *Cygwin* is a Linux-like environment for Windows [19].

## 3.1 Generation of Input

The program is executed in the following manner

**./dna.pl re1 re2 re3 dna_file**

The program (dna.pl) takes as input the names of the three restriction enzymes (re1, re2, and re3) and the name of the file that contains the DNA sequence (dna_file). The restriction enzymes are then searched in a restriction enzyme database to get their corresponding recognition sequence along with their cleavage site. The first restriction

enzyme *a* is applied to the DNA sequence and is fragmented. These fragments are the bags in the big-bag. The fragments are again fragmented by applying the union of the second *b* and the third *c* restriction enzymes to them. The lengths (number of nucleotides) of these subfragments are the elements of the bags of the big-bag. The process is repeated again by applying restriction enzyme *b* and then by applying the union of restriction enzymes *a* and *c*.

The output of the program is two big-bags: big_bagA and big_bagB. These big-bags are then given as inputs to the shuffle program, which shuffles the bags of the big-bag and the elements in the bags of the big-bag. This is done so as to simulate the real-life laboratory environment. The resulting big-bags of the Lambda sequence using the restriction enzymes *Cla*I, *Bam*HI, and *Ava*I is shown in Table 3.1.

Table 3.1: Big-bags of Lambda Sequence

| **Big_bagA (unshuffled)** | **big_bagB (unshuffled)** | **big_bagA (shuffled)** | **big_bagB (shuffled)** |
|---|---|---|---|
| [A1{3}] | [B1{3,5,8}] | [A1{5,7,7}] | [B1{12,14,8}] |
| [A2{5,8,10}] | [B2{10,6,6,6,}] | [A2{5,7,12,14}] | [B2{10,6,6,6}] |
| [A3{6,6,6,11}] | [B3{11,8,8,}] | [A3{5,8,10}] | [B3{12,5,7}] |
| [A4{8,8,12}] | [B4{12,5,7}] | [A4{8,8,12}] | [B4{3,5,8}] |
| [A5{5,7,12,14}] | [B5{12,14,8}] | [A5{3}] | [B5{11,8,8}] |
| [A6{8,8}] | [B6{8,5,7,7}] | [A6{6,6,6,11}] | [B6{8,5,7,7}] |
| [A7{5,7,7}] | | [A7{8,8}] | |

Each bag in a big-bag is delimited by “[” and “]”. The first entry within a bag is the Bag identifier, and the contents of the bag are enclosed in “{” and “}”. The contents which are the length of the subfragments are separated by commas.

## 3.2 Implementation of Constraint Automata

The program is executed in the following manner

**./gmap.pl big_bag X big_bag Y**

The program (gmap.pl) takes as input the name of the big-bags X and Y. The program can be executed in two modes: ONE, in which the program will find the first solution; and ALL, in which the program will find all the possible solutions. The default mode is ALL.

The number of bags in each of the big-bags is compared. The big-bag with the fewer number of bags is assigned to big-bag B and the other bag is assigned to big-bag A. The starting bag is always the first bag of big-bag B.

The working of the program is similar to a pre-order traversal of a tree. Every node in the tree is a bag of either of the two big-bags. The following two big-bags are used to explain the working of the program:

| | | |
|---|---|---|
| **[A1{5,8,13,13}]** | **[B1{8}]** | **[B3{5,13}]** |
| **[A2{7,8,17}]** | **[B2{17}]** | **[B4{7,8,13}]** |

The corresponding tree implementation is shown in Figure 3.1. A partial pre-order traversal of the tree is shown step-wise in Table 3.2.

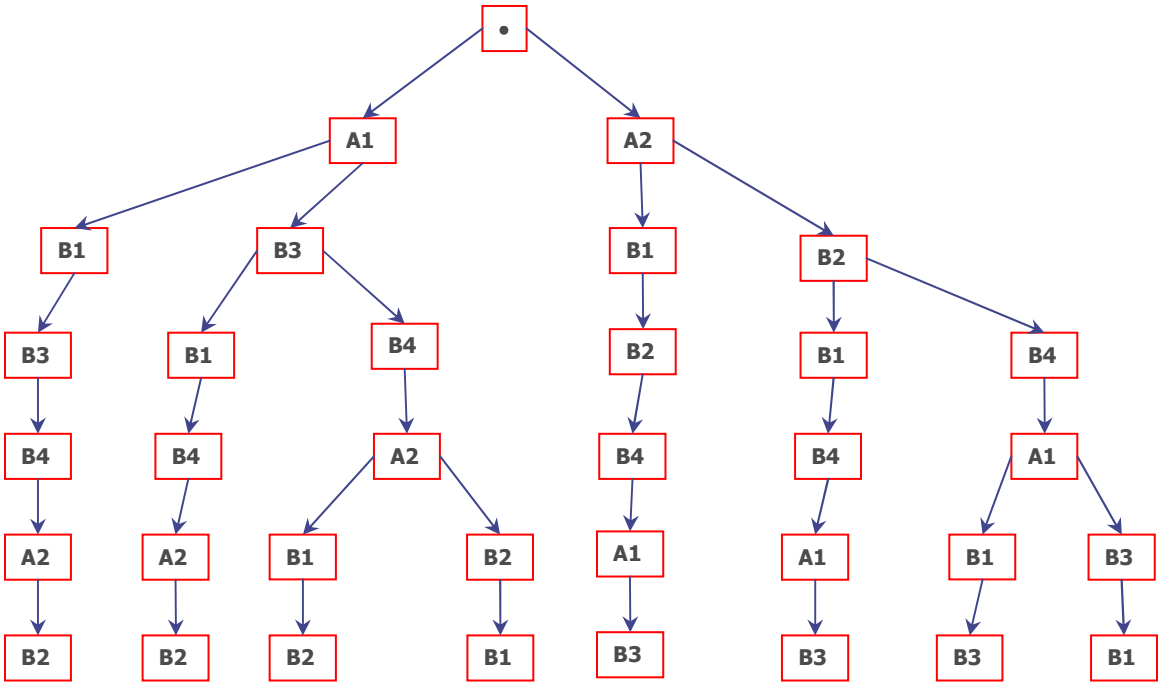


Figure 3.1: Tree Implementation

Table 3.2: Pre-order traversal of tree

| Node | CurrBag | S | UA | UB | State | Options | SelBag | Choices | Cflag |
|---|---|---|---|---|---|---|---|---|---|
| 0 | - | { } | B1,B2,B3,B4 | A1,A2 | INIT | A1,A2 | A1 | A2 | 1 |
| 1 | A1 | {5,8,13,13} | B1,B2,B3,B4 | A2 | B_AHEAD | B1,B3 | B1 | B3 | 1 |
| 2 | B1 | {5,13,13} | B2,B3,B4 | A2 | B_AHEAD | B3 | B3 | { } | 0 |
| 3 | B3 | {13} | B2,B4 | A2 | B_AHEAD | B4 | B4 | { } | 0 |
| 4 | B4 | {7,8} | B2 | A2 | A_AHEAD | A2 | A2 | { } | 0 |
| 5 | A2 | {17} | B2 | { } | B_AHEAD | B2 | B2 | { } | 0 |
| 6 | B2 | { } | { } | { } | HALT | - | - | - | - |
| Cflag != 0; Backtrack to Node1 | | | | | | | | | |
| 1 | A1 | {5,8,13,13} | B1,B2,B3,B4 | A2 | B_AHEAD | B3 | B3 | { } | 0 |
| 2 | B3 | {8,13} | B1,B2,B4 | A2 | B_AHEAD | B1,B4 | B1 | B4 | 1 |
| 3 | B1 | {13} | B2,B4 | A2 | B_AHEAD | B4 | B4 | { } | 0 |
| 4 | B4 | {7,8} | B2 | A2 | A_AHEAD | A2 | A2 | { } | 0 |
| 5 | A2 | {17} | B2 | { } | B_AHEAD | B2 | B2 | { } | 0 |
| 6 | B2 | { } | { } | { } | HALT | - | - | - | - |
| Cflag != 0; Backtrack to Node2 | | | | | | | | | |

### 3.2.1 ONE mode

The execution starts at the origin (Node 0), which is the initial state (INIT). The set *S* is empty. Since the number of bags in A is smaller than that of B, the set *UB* contains the bags of big-bag A and *UA* contains the bags of big-bag B. The various options from which the automata can choose a starting bag are all the bags in *UB*. Hence, the set *Options* contain all the bags of *UB*. This set is sorted in ascending order.

The program then takes the first bag in *Options* (in this case, A1) as the starting bag (*SelBag*) and puts the remaining bags (in this case, A2) into another set called *Choices*. Since *Choices* is not empty, the flag *Cflag* for that node is set to "1". This means that there is another branch possible from this node.

The selected bag, A1, is then the node 1 of the tree and the current bag (*CurrBag*). *S* now contains the elements of bag A1. *UA* contains all the bags of big-bag B and *UB* contains all the bags of big-bag A except A1, which has been used. Since *S* contains the elements by which big-bag A is ahead of big-bag B, the automata is in the "B_ahead" state. The next bag to be chosen is from *UA*, which contains the unused A bags. There are two possible choices: B1, B3. We select bag B1, put bag B3 in *Choices*, and set *Cflag* to "1". Since the elements of B1 are contained in *S*, we subtract the elements from *S*. The resulting *S* still contains the elements by which big-bag A is ahead. So the automata will remain in the "B_ahead" state. Bag B1 is then removed from *UB*.

The execution continues in a similar fashion. At node 6, *S*, *UA*, and *UB* are empty and hence a solution is found. The automaton then moves into the "HALT" state and the execution stops.

### 3.2.2 ALL mode

In the ALL mode the execution is the same as that in the ONE mode, but the automaton does not stop after finding the first solution. Instead the solution is saved and the automaton moves into the “Backtrack” state. The last node to have its *Cflag* set to “1” is found and the automaton backtracks to that node.

In this example, node 1 is the last node to have its *Cflag* set to “1”. So the automaton backtracks to node 1, and the first bag in *Choices* is set to *SelBag*. Now *Choices* is empty, so *Cflag* is set to “0”. The execution then continues as explained above. When none of the nodes have their *Cflag* set to “1”, the automaton moves into the “HALT” state and the execution stops.

## 3.3 CPU Time

In “ONE” mode, the amount of CPU time consumed was measured as the time to find the first solution. In “ALL” mode, the amount of CPU time consumed was measured as the time to find all the possible solutions.

All experiments were carried out on an Athlon XP 1800+ desktop with 512 MB RAM and running the Windows XP Professional operating system.

# Chapter 4

# Analysis and Results

The input generation Perl program is executed on 156 sequences of the human DNA chromosomes along with 5 sets of three Non-Overlapping Restriction Enzymes (triples) [26], to generate the input big-bags. The perl implementation of the constraint automata takes these inputs and finds one or all possible solutions. Several Perl reference books were consulted to develop the generation and the implementation programs [2, 3, 5, 6, 8, 10, 15, 16, 18, 23, 24, 25].

## 4.1 Generation of Data

The DNA sequences for generating the input for the Constraint Automata were taken from the Human Genome Resources website of the National Center for Biotechnology Information (NCBI) [22]. These sequences are various parts of the chromosomes of the human DNA. These sequences are subjected to restriction enzymes.

Restriction enzyme, as mentioned in previous chapters, "scans" the DNA looking for a particular sequence it represents (set of four to six nucleotides), and cuts the DNA at the particular cleavage site. Hence the choice of the Restriction Enzyme triple should not

contain REs which have a similar pattern of nucleotides. For example, the Lambda sequence that has been used extensively in this thesis, has three Non-Overlapping Restriction Enzymes (ClaI → AT^CGAT, BamHI → G^GATCC, and AvaI → C^YCGRG) that were used. On close examination of the other REs (Restriction enzyme database {bionet.311}) it was found that EcoRV (GAT^ATC) and MboI (^GATC) were a couple of other REs that could scan and cut the Lambda sequence. But, EcoRV cannot form a part of the triple as EcoRV (GAT^ATC) overlaps with ClaI (AT^CGAT) – the ending sequence of EcoRV is the starting sequence of ClaI (ATC), and also the ending of sequence of ClaI is the starting sequence of EcoRV (GAT). This leads to the conclusion that either one could be a part of the triple but not both. In real-life situations more than 3 REs are used to cut a DNA but for the sake of simplicity we restrict our study to the use of 3 REs, hence the name triple.

The 156 sequences (various parts of human chromosomes) are subjected to the following five sets of non-overlapping restriction enzyme triples, which were carefully selected from a restriction enzyme database [26], to give us the input big-bags for the constraint automata.

1. HindIII, AccI, AceI
2. AauI, HindII, CacBI
3. BclI, AhyI, TaaI
4. PsiI, PciI, HhaII
5. PdiI, SurI, SspI

## 4.2 Perl Implementation of the Constraint Automaton

The perl implementation of the constraint automaton is then run using the input bags generated for each of the 156 sequences and the CPU time taken to find the first solution is calculated. This time is noted as the execution time. Further, execution time is used as an index of the time complexity of the constraint automata. For this purpose, charts are plotted with the number of fragments (of each sequence) on the X-axis and the execution time (for each sequence) in seconds on the Y-axis. Figures 4.1 through 4.5 show the charts generated for each Restriction Enzyme triple.

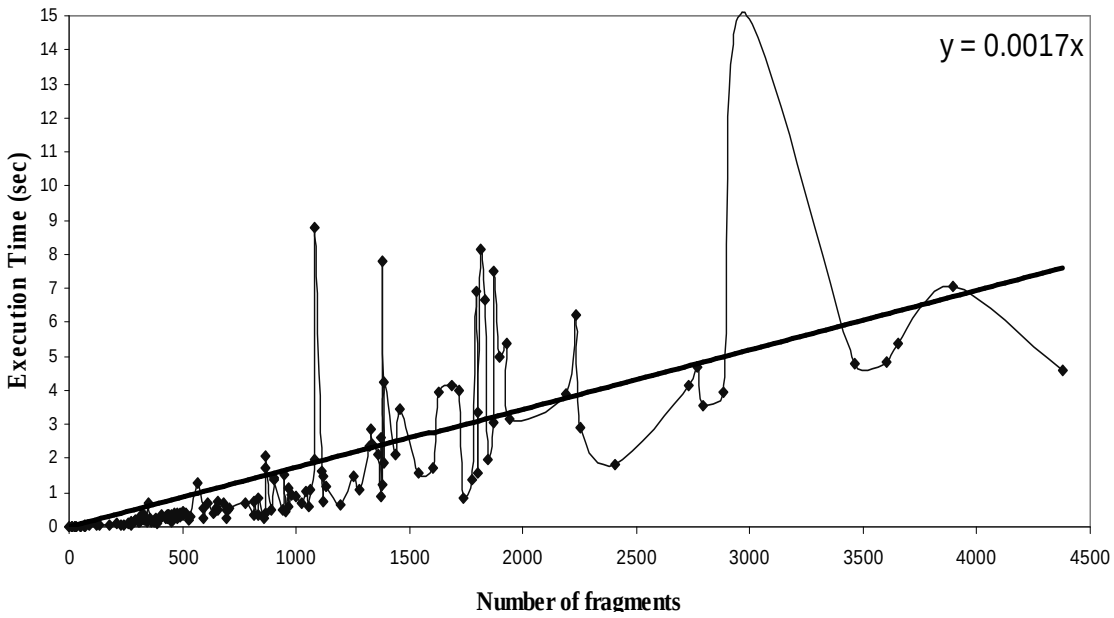


Figure 4.1: Time Complexity for RE Triple: HindIII, AccI, AceI

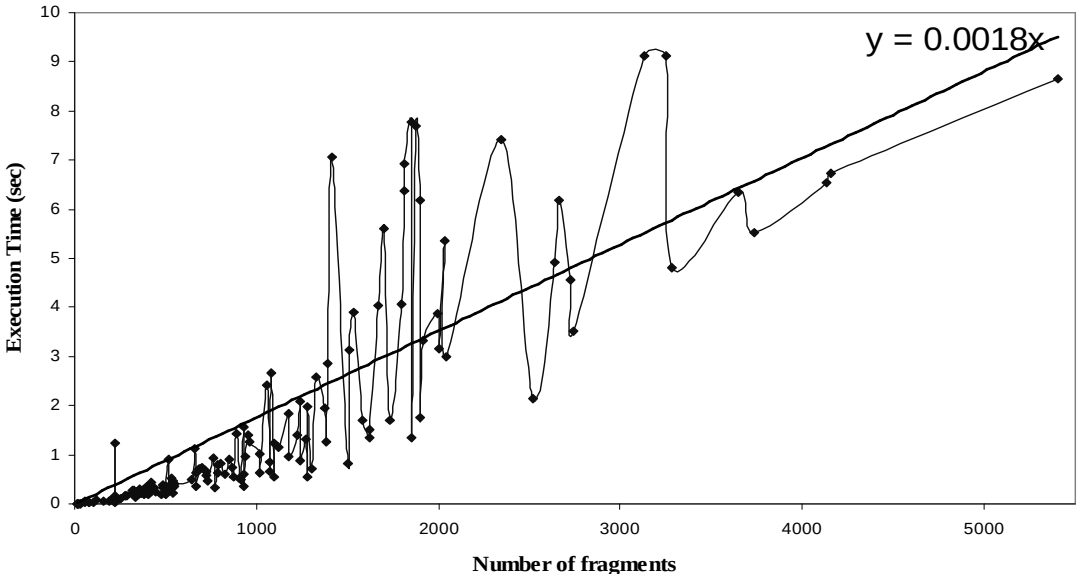


Figure 4.2: Time Complexity for RE Triple: AauI, HindII, Cac8I

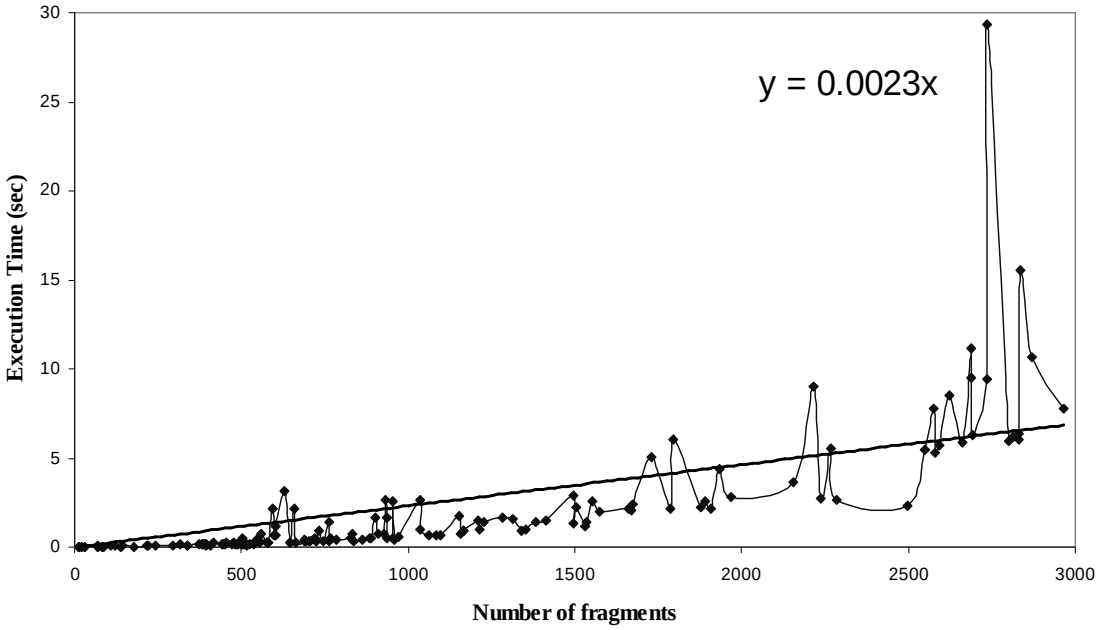


Figure 4.3: Time Complexity for RE Triple: BclI, AhyI, TaaI

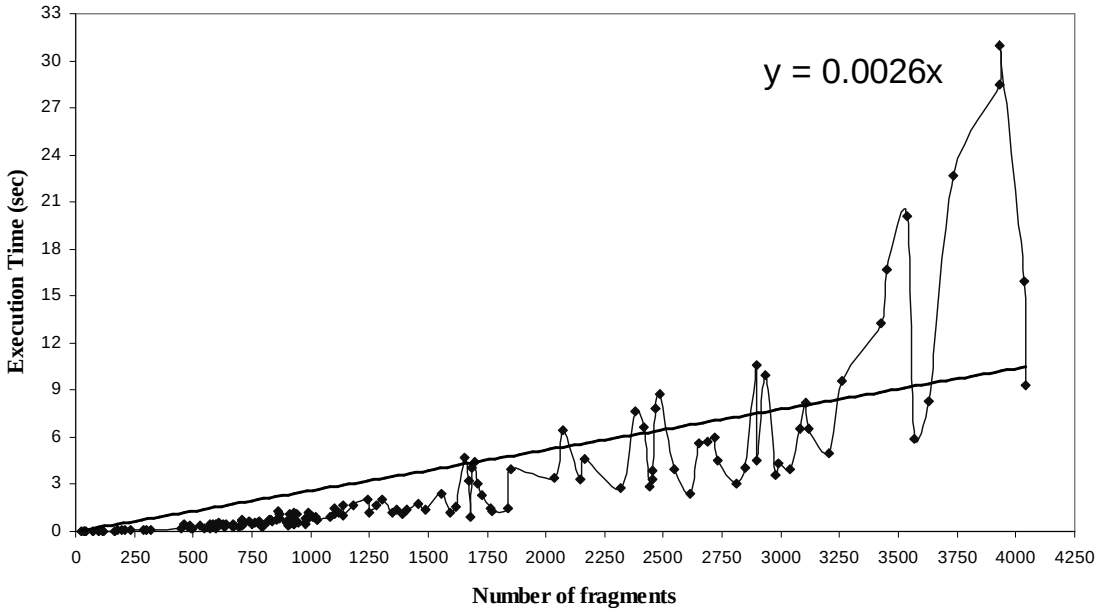


Figure 4.4: Time Complexity for RE Triple: PsiI, PciI, HhaII

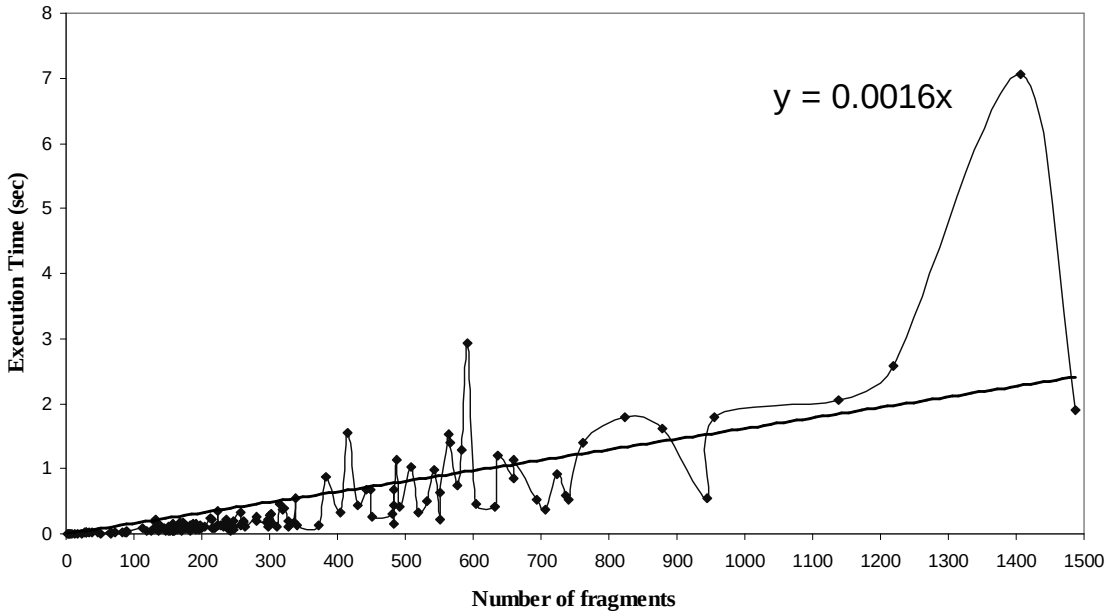


Figure 4.5: Time Complexity for RE Triple: PdiI, SurI, SspI

## 4.3 Summary

The five graphs show a linear relationship between the execution time for the constraint automata and the number of fragments in each input bag. It is thus observed that as the number of fragments increases, it results in the increase of the execution time, thus indicating a linear time complexity for the Constraint-Automata Solution.

# Chapter 5

# Conclusions and Future Work

## 5.1 Conclusions

The Genome Map Assembly Problem (GMAP) was solved using a Constraint Automaton (used for solving "Big-Bag Matching Problem") proposed by Revesz [11]. The automaton was modified to incorporate backtracking, which increased its applicability. It was implemented using a Perl script running on windows operating system.

A GMAP was constructed for 156 DNA sequences which were fragmented using three non-overlapping restriction enzymes. The resulting fragments of each sequence were then shuffled to represent a real-life situation. Each sequence could be represented in two big-bags which were then matched using the modified constraint automaton. One of the matched solutions would be the actual sequence.

The time taken for the automaton to generate a solution was noted which was then graphed with the number of fragments that each sequence was cut into. This was done for 5 sets of three non-overlapping restriction enzymes which cut the 156 sample DNA sequences. The graphs (see Chapter 4) clearly indicate a linear relationship between the

number of fragments of a sequence to the execution time, thus reinforcing the hypothesis that the modified constraint automaton would have a linear time complexity.

## 5.2 Future Work

The modified constraint automaton needs to be tested on real world applications of genome sequencing. It can be further modified to include error-tolerance which could be used to study the time complexities of error-prone data (real world application). It is hypothesized that the modified constraint automaton with error tolerance will also show a linear time complexity. These are some issues that need more time to modify, implement and test the constraint automaton solution before any results can be obtained.

# Appendix

# Creation of Big-Bags for Lambda

## A.1 Lambda sequence

Lambda is a bacteriophage, a virus that infects bacteria. It was first isolated from E. coli (Escherichia coli), a common bacterium that has been studied intensively by geneticists because it has a small genome and is usually harmless and easy to grow. The genetic material of Lambda consists of a double-stranded DNA molecule. It is used as a cloning vector, accommodating fragments of DNA up to 15,000 base pairs long. The following sequence is a part of one of the two strands of the complete double-helical Lambda sequence whose (48502 base pairs):

CATCGATCTCGGGAGGGATCCATTATCGATTCCCGGGCTCGGGGGATCCTTCC
ATCGATGGGCCCGAGGCGGATCCCTACTATCGATCCCGGGGGGATCCTTAAT
TCTCGAGAAGGCCTATCGATCAAGGATCCTATCGATCCCGAGTCCCGGGAT

Where A, C, G, and T, are the nucleotides.

## A.2 Restriction Enzymes Used

We apply the following three restriction enzymes on the above DNA sequence:

ClaI (AT^CGAT) henceforth represented as restriction enzyme *a*.

BamHI (G^GATCC) henceforth represented as restriction enzyme *b*,

AvaI (C^YCGRG) henceforth represented as restriction enzyme *c*, where Y = C or T and R = A or G. Hence AvaI's sequence is recognized as anyone of the following sequences CCCGGG, or CCCGAG, or CTCGGG, or CTCGAG.

The "^" symbol in the sequence of each restriction enzyme indicates the place where the RE would cut the DNA.

## A.3 Obtaining Input Data

We have three cases:

1. Apply restriction enzyme *a* and then apply restriction enzymes *b* ∪ *c*.
2. Apply restriction enzyme *b* and then apply restriction enzymes *a* ∪ *c*.
3. Apply restriction enzyme *c* and then apply restriction enzymes *a* ∪ *b*.

### A.3.1 Case 1

The restriction enzyme *a* occurs at the following places on the given sequence:

C**ATCGAT**CTCGGGAGGGATCCATT**ATCGAT**TCCCGGGCTCGGGGGATCCTTC
C**ATCGAT**GGGCCCGAGGCGGATCCCTACT**ATCGAT**CCCGGGGGGATCCTTA
ATTCTCGAGAAGGCCT**ATCGAT**CAAGGATCCT**ATCGAT**CCCGAGTCCCGGGA
T

When this sequence is fragmented based on the cleavage site, we get the following fragments of the original sequence:

A1: CAT
A2: CGATCTCGGGAGGGATCCATTAT
A3: CGATTCCCGGGCTCGGGGGATCCTTCCAT
A4: CGATGGGCCCGAGGCGGATCCCTACTAT

A5: CGATCCCGGGGGGATCCTTAATTCTCGAGAAGGCCTAT
A6: CGATCAAGGATCCTAT
A7: CGATCCCGAGTCCCGGGAT

The restriction enzymes *b U c* occurs at the following places on each fragments:

A1: CAT
A2: CGATCTCGGGAGGGATCCATTAT
A3: CGATTCCCGGGCTCGGGGGATCCTTCCAT
A4: CGATGGGCCCGAGGCGGATCCCTACTAT
A5: CGATCCCGGGGGGATCCTTAATTCTCGAGAAGGCCTAT
A6: CGATCAAGGATCCTAT
A7: CGATCCCGAGTCCCGGGAT

The fragments when cut at the respective cleavage sites of restriction enzymes *b U c* we get the following sub-fragments:

| | | |
|---|---|---|
| A1 | [CAT] | [A1{3}] |
| A2 | [CGATC] [TCGGGAGG] [GATCCATTAT] | [A2{5,8,10}] |
| A3 | [CGATTC] [CCGGGC] [TCGGGG] [GATCCTTCCAT] | [A3{6,6,6,11}] |
| A4 | [CGATGGGC] [CCGAGGCG] [GATCCCTACTAT] | [A4{8,8,12}] |
| A5 | [CGATC] [CCGGGGG] [GATCCTTAATTC] [TCGAGAAGGCCTAT] | [A5{5,7,12,14}] |
| A6 | [CGATCAAG] [GATCCTAT] | [A6{8,8}] |
| A7 | [CGATC] [CCGAGTC] [CCGGGAT] | [A7{5,7,7}] |

This entire multi-set (shown as the 3rd column of the above table) is named *big_bag_A* which contains *bags* (A1, A2,…, A7) and each bag contains the *length of the sub-fragment* represented by a number (obtained by counting the number of nucleotides in each sub-fragment).

## A.3.2 Case 2

The restriction enzyme *b* occurs at the following places on the given sequence:

CATCGATCTCGGGAGGGATCCATTATCGATTCCCGGGCTCGGGGGATCCTTC
CATCGATGGGCCCGAGGCGGATCCCTACTATCGATCCCGGGGGGATCCTTA

ATTCTCGAGAAGGCCTATCGATCAAGGATCCTATCGATCCCGAGTCCCGGGA
T

When this sequence is fragmented based on the cleavage site, we get the following fragments of the original sequence:

B1: CATCGATCTCGGGAGG
B2: GATCCATTATCGATTCCCGGGCTCGGGG
B3: GATCCTTCCATCGATGGGCCCGAGGCG
B4: GATCCCTACTATCGATCCCGGGGG
B5: GATCCTTAATTCTCGAGAAGGCCTATCGATCAAG
B6: GATCCTATCGATCCCGAGTCCCGGGAT

The restriction enzymes *a U c* occurs at the following places on each fragment:

B1: CATCGATCTCGGGAGG
B2: GATCCATTATCGATTCCCGGGCTCGGGG
B3: GATCCTTCCATCGATGGGCCCGAGGCG
B4: GATCCCTACTATCGATCCCGGGGG
B5: GATCCTTAATTCTCGAGAAGGCCTATCGATCAAG
B6: GATCCTATCGATCCCGAGTCCCGGGAT

The fragments when cut at the respective cleavage sites of restriction enzymes *a U c* we get the following sub-fragments:

| | | |
|---|---|---|
| B1 | [CAT] [CGATC] [TCGGGAGG] | [B1{3,5,8}] |
| B2 | [GATCCATTAT] [CGATTC] [CCGGGC] [TCGGGG] | [B2{10,6,6,6}] |
| B3 | [GATCCTTCCAT] [CGATGGGC] [CCGAGGCG] | [B3{11,8,8}] |
| B4 | [GATCCCTACTAT] [CGATC] [CCGGGGG] | [B4{12,5,7}] |
| B5 | [GATCCTTAATTC] [TCGAGAAGGCCTAT] [CGATCAAG] | [B5{12,14,8}] |
| B6 | [GATCCTAT] [CGATC] [CCGAGTC] [CCGGGAT] | [B6{8,5,7,7}] |

This entire multi-set (shown as the 3rd column of the above table) is named *big_bag_B* which contains *bags* (B1, B2,…, B6) and each bag contains the *length of the sub-fragment* represented by a number (obtained by counting the number of nucleotides in each sub-fragment).

### A.3.3 Case 3

The restriction enzyme *c* occurs at the following places on the given sequence:

CATCGATCTCGGGAGGGATCCATTATCGATTCCCGGGCTCGGGGGATCCTT
CCATCGATGGGCCCGAGGCGGATCCCTACTATCGATCCCGGGGGGATCCTT
AATTCTCGAGAAGGCCTATCGATCAAGGATCCTATCGATCCCGAGTCCCGG
GAT

When this sequence is fragmented based on the cleavage site, we get the following fragments of the original sequence:

C1: CATCGATC
C2: TCGGGAGGGATCCATTATCGATTC
C3: CCGGGC
C4: TCGGGGGATCCTTCCATCGATGGGC
C5: CCGAGGCGGATCCCTACTATCGATC
C6: CCGGGGGGATCCTTAATTC
C7: TCGAGAAGGCCTATCGATCAAGGATCCTATCGATC
C8: CCGAGTC
C9: CCGGGAT

The restriction enzymes *a U b* occurs at the following places on each fragment:

C1: CATCGATC
C2: TCGGGAGGGATCCATTATCGATTC
C3: CCGGGC
C4: TCGGGGGATCCTTCCATCGATGGGC
C5: CCGAGGCGGATCCCTACTATCGATC
C6: CCGGGGGGATCCTTAATTC
C7: TCGAGAAGGCCTATCGATCAAGGATCCTATCGATC
C8: CCGAGTC
C9: CCGGGAT

The fragments when cut at the respective cleavage sites of restriction enzymes *a U b* we get the following sub-fragments:

| | | |
|---|---|---|
| C1 | [CAT] [CGATC] | [C1{3,5}] |
| C2 | [TCGGGAGG] [GATCCATTAT] [CGATTC] | [C2{8,10,6}] |
| C3 | [CCGGGC] | [C3{6}] |
| C4 | [TCGGGG] [GATCCTTCCAT] [CGATGGGC] | [C4{6,11,8}] |

| | | |
|---|---|---|
| C5 | [CCGAGGCG] [GATCCCTACTAT] [CGATC] | [C5{8,12,5}] |
| C6 | [CCGGGGG] [GATCCTTAATTC] | [C6{7,12}] |
| C7 | [TCGAGAAGGCCTAT] [CGATCAAG] [GATCCTAT] [CGATC] | [C7{14,8,8,5}] |
| C8 | [CCGAGTC] | [C8{7}] |
| C9 | [CCGGGAT] | [C9{7}] |

This entire multi-set (shown as the 3rd column of the above table) is named *big_bag_C* which contains *bags* (C1, C2,…, C9) and each bag contains the *length of the sub-fragment* represented by a number (obtained by counting the number of nucleotides in each sub-fragment).

## A.4 Summary

The big_bags obtained by hand above are compared to the big_bags obtained using the perl code. The results are presented in tabular form below:

| Big_Bag | By Hand | Perl Code |
|---|---|---|
| **A** | [A1{3}]<br>[A2{5,8,10}]<br>[A3{6,6,6,11}]<br>[A4{8,8,12}]<br>[A5{5,7,12,14}]<br>[A6{8,8}]<br>[A7{5,7,7}] | [A1{3}]<br>[A2{5,8,10}]<br>[A3{6,6,6,11}]<br>[A4{8,8,12}]<br>[A5{5,7,12,14}]<br>[A6{8,8}]<br>[A7{5,7,7}] |
| **B** | [B1{3,5,8}]<br>[B2{10,6,6,6}]<br>[B3{11,8,8}]<br>[B4{12,5,7}]<br>[B5{12,14,8}]<br>[B6{8,5,7,7}] | [B1{3,5,8}]<br>[B2{10,6,6,6}]<br>[B3{11,8,8}]<br>[B4{12,5,7}]<br>[B5{12,14,8}]<br>[B6{8,5,7,7}] |
| **C** | [C1{3,5}]<br>[C2{8,10,6}]<br>[C3{6}]<br>[C4{6,11,8}]<br>[C5{8,12,5}]<br>[C6{7,12}]<br>[C7{14,8,8,5}]<br>[C8{7}]<br>[C9{7}] | [C1{3,5}]<br>[C2{8,10,6}]<br>[C3{6}]<br>[C4{6,11,8}]<br>[C5{8,12,5}]<br>[C6{7,12}]<br>[C7{14,8,8,5}]<br>[C8{7}]<br>[C9{7}] |

# Glossary

*Adenine (A)*: A nitrogenous base, one member of the base pair AT (adenine-thymine) in DNA.

*Assembly*: Putting sequenced fragments of DNA into their correct chromosomal positions.

*Bacteriophage*: See *phage*.

*Bag*: It is a multiset whose elements are fragments that can occur multiple times.

*Base*: One of the molecules that form DNA molecules.

*Base Pair (bp)*: Two nitrogenous bases (adenine and thymine or guanine and cytosine) held together by weak bonds. Two strands of DNA are held together in the shape of a double helix by the bonds between base pairs.

*Base Sequence*: The order of nucleotide bases in a DNA molecule; determines structure of proteins encoded by that DNA.

*BBMP*: See *Big-Bag Matching Problem*.

*Big-Bag*: it is a multiset whose elements are bags that can occur multiple times.

*Big-Bag Matching Decision Problem*: It is the problem of deciding whether two big-bags match.

*Big-Bag Matching Problem (BBMP)*: It is the problem of finding matching presentations for two given big-bags if they match.

*Bioinformatics*: The science of managing and analyzing biological data using advanced computing techniques. It is especially important in analyzing genomic research data.

*Cell*: The basic unit of any living organism that carries on the biochemical processes of life.

*Chromosome*: It is the self-replicating genetic structure of cells containing the cellular DNA that bears in its nucleotide sequence the linear array of genes. In prokaryotes, chromosomal DNA is circular, and the entire genome is carried on one chromosome. Eukaryotic genomes consist of a number of chromosomes whose DNA is associated with different kinds of proteins.

*Cloning Vector*: It is a DNA molecule originating from a virus, a plasmid, or the cell of a higher organism into which another DNA fragment of appropriate size can be integrated without loss of the vector's capacity for self-replication.

*Constraint Automaton*: A constraint automaton consists of a set of *states*, a set of state *variables*, *transitions* between states, an *initial state*, and the domain and initial values of the state variables.

*Contiguity Constraint*: Any valid presentation satisfies a contiguity constraint if it contains the elements within a bag next to each other.

*Cygwin*: It is a Linux-like environment for Windows. The current version is v2.340.2.5.

*Cytosine (C)*: A nitrogenous base, one member of the base pair GC (guanine and cytosine) in DNA.

*Deoxyribonucleic Acid (DNA)*: DNA is a nucleic acid consisting of nucleotides and is shaped like a double helix. It is associated with the transmission of genetic information that is the hereditary material in all living cells. (Figure 1.4)

*Deoxyribose*: A type of sugar that is one component of DNA (deoxyribonucleic acid).

*DNA*: See *Deoxyribonucleic Acid.*

*DNA Sequence*: The relative order of base pairs, whether in a DNA fragment, gene, chromosome, or an entire genome.

*Double Helix*: The twisted-ladder shape that two linear strands of DNA assume when complementary nucleotides on opposing strands bond together.

*Enzyme*: A protein that acts as a catalyst, speeding the rate at which a biochemical reaction proceeds but not altering the direction or nature of the reaction.

*Escherichia Coli*: Common bacterium that has been studied intensively by geneticists because of its small genome size, normal lack of pathogenicity, and ease of growth in the laboratory.

*Gene*: The fundamental physical and functional unit of heredity. A gene is an ordered sequence of nucleotides located in a particular position on a particular chromosome that encodes a specific functional product (i.e., a protein).

*Genome*: The genome of an organism is its set of chromosomes, containing all of its genes and the associated DNA. It contains the entire set of hereditary instructions for building, running, and maintaining an organism, and passing life on to the next generation.

*Genome Map*: A genome map is an ordering of a set of clones according to their believed position on a DNA string.

*Genome Map Assembly*: It is the process of assembling the fragments into a logical set of sequences, in order to obtain the original DNA sequence.

*Genome Map Assembly Problem (GMAP)*: The problem of executing the genome map assembly is called the Genome Map Assembly Problem (GMAP).

*Genome Mapping*: The process of finding a genome map is called Genome Mapping.

*Genome Sequencing*: Genome sequencing is finding the order of DNA nucleotides, or bases, in a genome – the order of As, Cs, Gs, and Ts that make up an organism's DNA. (Figure 1.5)

*GMAP*: See *Genome Map Assembly Problem*.

*Guanine (G)*: A nitrogenous base, one member of the base pair GC (guanine and cytosine) in DNA.

*Junk DNA*: Stretches of DNA that do not code for genes; most of the genome consists of so-called junk DNA which may have regulatory and other functions. It is also called non-coding DNA.

*Kilobase (kb)*: Unit of length for DNA fragments equal to 1000 nucleotides.

*Lambda*: Lambda is a bacteriophage, a virus that infects bacteria. It was first isolated from E. coli (Escherichia coli), a common bacterium that has been studied intensively by geneticists because it has a small genome and is usually harmless and easy to grow. It is used as a cloning vector, accommodating fragments of DNA up to 15,000 base pairs long.

*Megabase (Mb)*: Unit of length for DNA fragments equal to 1 million nucleotides and roughly equal to 1 cM.

*Nitrogenous Base*: A nitrogenous base is one of the three parts of a nucleotide. They carry the genetic information, so the words "nucleotide" and "base" are used interchangeably. There are four nitrogenous bases: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T).

*Nucleic Acid*: A large molecule composed of nucleotide subunits.

*Nucleotide*: A nucleotide is the smallest unit of a DNA, consisting of three parts: a sugar molecule (S), a phosphate group (P), and a structure called a nitrogenous base (A, C, G, and T). (Figure 1.3)

*Nucleus*: The cellular organelle that contains most of the genetic material.

*Phage*: A virus for which the natural host is a bacterial cell.

*Physical Map*: A map of the locations of identifiable landmarks on DNA (e.g., restriction-enzyme cutting sites, genes), regardless of inheritance. Distance is measured in base pairs. For the human genome, the lowest-resolution physical map is the banding patterns on the 24 different chromosomes; the highest-resolution map is the complete nucleotide sequence of the chromosomes.

*Plasmid*: Autonomously replicating extra-chromosomal circular DNA molecules, distinct from the normal bacterial genome and nonessential for cell survival under nonselective conditions. Some plasmids are capable of integrating into the host genome. A number of artificially constructed plasmids are used as cloning vectors.

*Presentation*: Each permutation of the bags and permutation of the elements of each bag within a big-bag is called a presentation. A big-bag can have several different presentations.

*Protein*: A large molecule composed of one or more chains of amino acids in a specific order; the order is determined by the base sequence of nucleotides in the gene that codes for the protein. Proteins are required for the structure, function, and regulation of the body's cells, tissues, and organs; and each protein has unique functions. Examples are hormones, enzymes, and antibodies.

*Reachable Configurations*: It is the set of states and state values that a constraint automaton can enter.

*REBASE*: It is the restriction enzyme database managed by Dr. Richard J Roberts. It is available at http://rebase.neb.com/rebase/rebase.html.

*Restriction Digest*: The process of cutting the DNA is called a restriction digest or a digestion.

*Restriction Endonuclease*: Restriction endonuclease is another name for restriction enzyme.

*Restriction Enzyme*: A restriction enzyme is the tool used to cut DNA into smaller pieces. A restriction enzyme usually recognizes a specific short sequence of bases in the DNA. Different restriction enzymes recognize different sequences.

*Restriction Map*: A piece of DNA produced by the action of one restriction enzyme can be specifically cleaved into smaller fragments by another restriction enzyme. The

pattern of such fragments is called a restriction map. It can serve as a fingerprint of a DNA molecule.

*Restriction Site*: The sequence which a restriction enzyme recognizes and cuts is called a restriction site.

*Sequencing*: Determination of the order of nucleotides (base sequences) in a DNA.

*Thymine (T)*: A nitrogenous base, one member of the base pair AT (adenine-thymine) in DNA.

*Transition*: Each transition of a constraint automaton consists of a set of constraints, which are made up of state variables and variables.

*Virus*: A noncellular biological entity that can reproduce only within a host cell. Viruses consist of nucleic acid covered by protein; some animal viruses are also surrounded by membrane. Inside the infected cell, the virus uses the synthetic capability of the host to produce progeny virus.

# Bibliography


[1] J.M. Berg, J.L. Tymoczko, L. Stryer. *Biochemistry*. W.H. Freeman, 2000.

[2] M.C. Brown. *Perl: The Complete Reference*, 2ed. Osborne/McGraw-Hill, 2001.

[3] T. Christiansen, N. Torkington. *Perl Cookbook*, 2 ed. O'Reilly, 1998.

[4] S.E. DeWeerdt. *What's a Genome?*. The Center for Advanced Genomics, 2003.

[5] J.E.F. Friedl. *Mastering Regular Expressions*. O'Reilly, 2002.

[6] M. Glover, A. Humphreys, E. Weiss. *Perl 5 How-To*. Waite Group Press, 1996.

[7] E. Harley, A.J. Bonner. A flexible approach to genome map assembly. In *Proc. International Symposium on Intelligent Systems for Molecular Biology*, pages 161-69. AAAI Press, Menlo Park 1994.

[8] S. Holzner. *Perl Black Book*. Coriolis, 2001.

[9] R.M. Karp. Mapping the genome: Some combinatorial problems arising in molecular biology. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 278-85. ACM Press, 1993.

[10] E. Quigley. *Perl by Example*, 3 ed. Prentice Hall, 2002.

[11] P.Z. Revesz. Bioinformatics. In *Introduction to Constraint Databases*. Springer-Verlag, 2002.

[12] P.Z. Revesz. Constraint Automata. In *Introduction to Constraint Databases*. Springer-Verlag, 2002.

[13] P.Z. Revesz. Refining restriction enzyme genome maps. *Constraints*, 2(3-4): 361-75, 1997.

[14] P.Z. Revesz. The dominating cycle problem in 2-connected graphs and the matching problem for bag of bags are NP-complete. In *Proc. International Conference on Paul Erdos and His Mathematics*, pages 221-5, 1999.

[15] R.L. Schwartz, T. Phoenix. *Learning Perl*, 3 ed. O'Reilly, 2001.

[16] J.D. Tisdall. *Beginning Perl for Bioinformatics*. O'Reilly, 2001.

[17] D. Voet, J. Voet. *Biochemistry*, 3ed Vol. 1. John Wiley, 2003.

[18] L. Wall, T. Christiansen, J. Orwant. *Programming Perl*, 3 ed. O'Reilly, 2000.

## Websites

[19] Cygwin Online Community. http://www.cygwin.com/.

[20] Entrez Database. http://www.ncbi.nlm.nih.gov/entrez/.

[21] Human Genome Project. http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml.

[22] Human Genome Resources. http://www.ncbi.nlm.nih.gov/genome/guide/human/.

[23] Perl Documentation. http://www.perldoc.com/.

[24] Perl Online Community. http://www.perl.org/.

[25] Perl Website. http://www.perl.com/.

[26] Restriction Enzyme Database. http://rebase.neb.com/rebase/rebase.html.