

Palestine Polytechnic University



College of Administrative Science and Informatics
Information systems Department

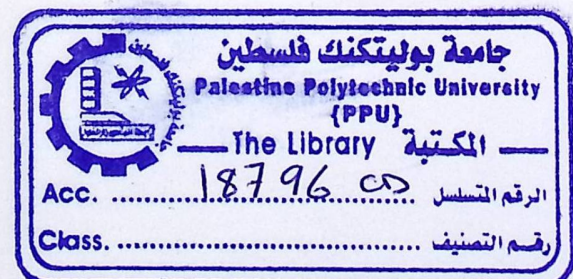
Novel Algorithm To Determine Highly Homologous Segmental Genome Duplication

Prepared by:
Talat Al_sharabati

Project Supervisor
Dr.Mahmoud Al-saheb
Dr.yaqoub Ashhab

This project is submitted in partial fulfillment of the requirements for the degree of
B.Sc. in Information systems in Palestine Polytechnic University

January 2006



Abstract

In the world of genomic research, projects aimed at revealing the complete sequence of a given genome are usually cumbersome and tedious. The major problem in these projects is that the long sequence of the genome should pass two processes; 1) randomly cutting it into a huge number of short DNA sequences known as Whole genome-shotgun reads (WGSRs), 2) thereafter using computer programs to reconstruct the whole genome sequence. The main drawback during the reconstruction process is the small fragments that come from duplicated segment in the original genome. Segmental duplication is a well known phenomenon which refers to blocks of DNA sequence that exist in more than one copy in the genome. The objective of my project is to develop a Windows-based system that can efficiently analyze and delineate segmental duplications in WGSRs databases. The main advantage of this system is its simplicity for users as well as capability to be used for any WGSRs database. Perl was used for developing this system because it is a very strong language in string processing and analysis. In addition, Perl has many bioinformatics modules that can facilitate programming for biological applications.

Dedication

To my parent and family...

To my instructor...

To my friends at the college...

To my partner who is waiting behind the occupation prison , Bellal Al-qwasmah...

To the coming generation who will benefit from this project...

To all who have participated at completing this project...

Acknowledgement

Table of Contents

First I would thank god for giving me the bless of thinking and studying
I would like to acknowledge my gratitude to all those who have helped me in the writing of this project. First and foremost, I wish to express my sincere thanks to Dr.Yaqoub al-ashhab and Dr.Mahmmoud al- sahib for their insightful comments on an earlier version of the project.

I am also thankful to may friends for offering invaluable suggestions about the project.

Special thanks go to Rula Al_sharabati for correcting my grammatical errors and for translating.

Finally, I also thank my parent for encouraging, and giving me the suitable environment

Chapter one System Specification

1.1 Introduction	7
1.2 Objectives	7
1.3 Functional Requirements	7
1.4 Non-Functional requirements	8
1.5 Allocation of Roles of System Developers	8
1.6 Feasibility Study	9
1.6.1 Introduction	9
1.6.2 Cost-Benefit Analysis	9
1.6.3 Experimental Study	10
1.6.4 Risk Analysis	13
1.6.5 Technical Feasibility	14
1.6.6 Legal Feasibility	14
1.6.7 Time Feasibility	14

Chapter three (Software Requirements Specification)

3.1 Introduction	16
3.2 Functional details description	15
3.3 Information Description	19
3.3.1 System Data Flow Diagram	19
3.3.2 Data Dictionary	20

Gene Duplication Finder (GD Finder) is a program designed to analyze databases that contain high number of shotgun reads that are produced by genome projects. Gene Duplication Finder can easily and accurately delineate the genomic segments that extra very similar copies somewhere else in the same genome. The program rely a Blast-based approach to analyze the whole genome shotgun reads in order to detect the shotgun reads that have similar extra copies. After detecting the duplicated shotgun reads, GD Finder will construct and determine physical limits of the duplicated segments.

The program was designed in the Information Technology Department, Palestine Polytechnic University, Hebron, and Palestine.

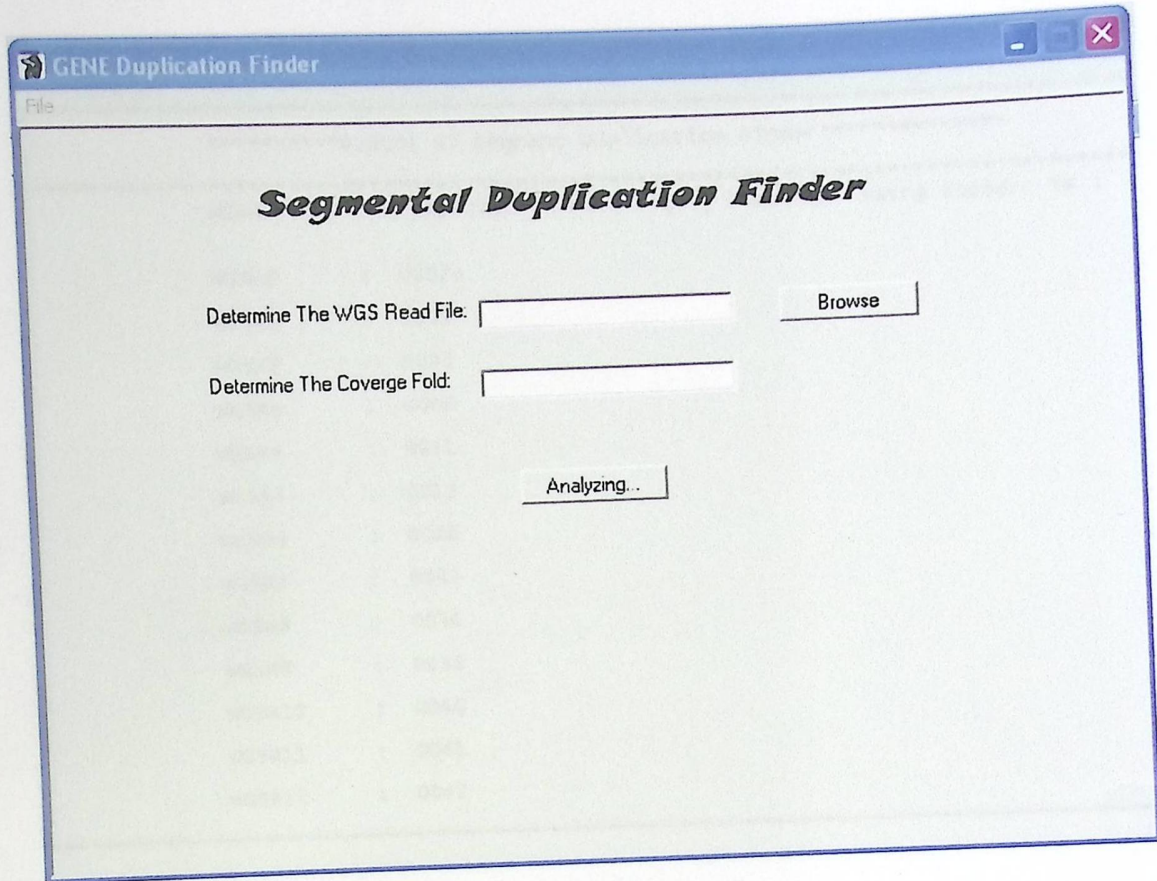
- How to Install the program

The following steps are needed to install the program:

1. Inside GD Finder folder, Click on Setup
2. Choose where to install (desired path).

- The main window of GD Finder allows the user to determine the WGSRs file and the coverage fold (as shown in the next figure).

Note: The coverage fold should be obtained when you download the WGSRs from the web sit of the genome project that you are interested in.



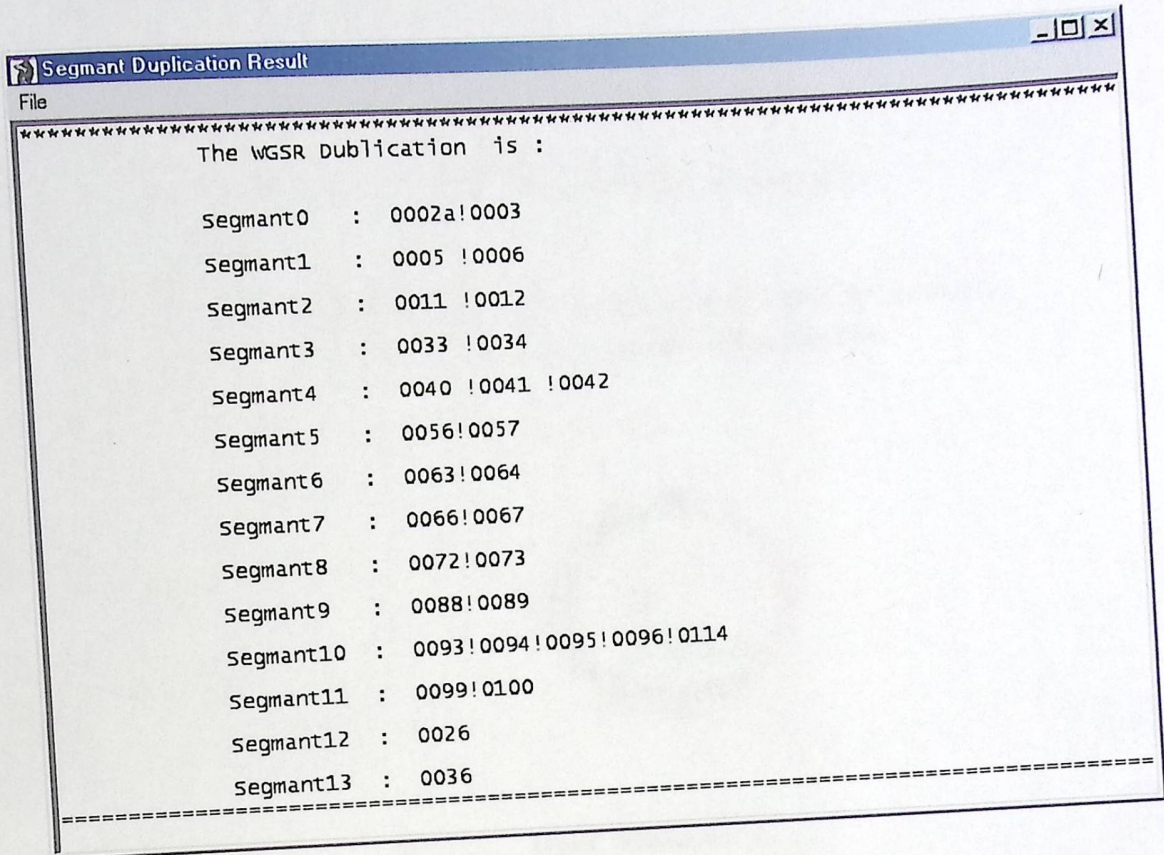
- After determining the path of the WGSRs database and the coverage fold, you can start the automatic analysis of the whole database by clicking the bottom **analyzing** as shown in the previous Figure.
- The program will automatically move to the next screen, which is shown below.
- In order to construct the segments that contain the duplicated sequences, you have to choose construct from the menu bar, then click duplication segments.

```
WGSRs Duplication Result
File Constact

*****Result of segmant Duplication Finder*****
*****
*****WGSRs;that are expected to have highly homologur extra copier: is :
*****

WGSRO      : 0002a
WGSR1      : 0003
WGSR2      : 0005
WGSR3      : 0006
WGSR4      : 0011
WGSR5      : 0012
WGSR6      : 0026
WGSR7      : 0033
WGSR8      : 0034
WGSR9      : 0036
WGSR10     : 0040
WGSR11     : 0041
WGSR12     : 0042
```

- The program will automatically analyze the results of WGSRs duplication finder.
- The final results of the segment duplications finder will appear in a new screen such as the one shown below.



The WGSR Dublication is :

- Segment0 : 0002a!0003
- Segment1 : 0005 !0006
- Segment2 : 0011 !0012
- Segment3 : 0033 !0034
- Segment4 : 0040 !0041 !0042
- Segment5 : 0056!0057
- Segment6 : 0063!0064
- Segment7 : 0066!0067
- Segment8 : 0072!0073
- Segment9 : 0088!0089
- Segment10 : 0093!0094 !0095!0096!0114
- Segment11 : 0099!0100
- Segment12 : 0026
- Segment13 : 0036

Acknowledgement

Table of Contents

First I would thank god for giving me the bless of thinking and studying
I would like to acknowledge my gratitude to all those who have helped me in
the writing of this project. First and foremost, I wish to express my sincere
thanks to Dr.Yaqoub al-ashhab and Dr.Mahmmoud al- sahib for their insightful
comments on an earlier version of the project.

I am also thankful to may friends for offering invaluable suggestions about the
project.

Special thanks go to Rula Al_sharabati for correcting my grammatical errors
and for translating.

Finally, I also thank my parent for encouraging, and giving me the suitable
environment

Chapter one (System Requirements)	7
1.1 Introduction	7
1.2 System Goals	7
1.3 System Constraints	7
1.4 System Requirements	7
1.5 System Architecture	7
1.6 System Development	7
1.7 System Testing	7
Chapter two (Software Requirements Specification)	16
2.1 Introduction	16
2.2 Functional Requirements	16
2.3 Information Requirements	19
2.3.1 User Data Flow Diagram	19
2.3.2 Data Dictionary	20

Table of Contents

Abstract.....	I
Dedication.....	II
Acknowledgment	III
Table of Contents.....	IV
List of Tables.....	VII
List of Figures.....	VIII

Chapter One (Introduction)

1.1 Overview.....	1
1.2 Genome and Gene.....	1
1.3 Gene Duplication.....	2
1.4 How genome are sequenced.....	3
1.5 Detection of segmental duplication.....	4

Chapter two (System Specification)

2.1 Introduction.....	7
2.2 System Objectives.....	7
2.3 Functional Requirement.....	7
2.4 Non-Functional requirements.....	8
2.5 Allocation of Roles of System Developers.....	8
2.6 Feasibility Study.....	9
2.6.1 Alternatives.....	9
2.6.2 Cost-Benefit Analysis.....	9
2.6.3 Economical Study.....	10
2.6.4 Risk Analysis	13
2.6.5 Technical Feasibility	14
2.6.6 Legal Feasibility	14
2.6.7 Time Feasibility	14

Chapter three (Software Requirements Specification)

3.1 Introduction.....	16
3.2 Functional details description.....	16
3.3 Information Description.....	19
3.3.1 System Data Flow Diagrams.....	19
3.3.2 Data Dictionary	20

Chapter Four (System Design)

4.1 Introduction.....	22
4.2 Input/output Design	23
4.2.1 Segmental Duplication Finder.....	23
4.2.2 Gene Duplication Result.....	24
4.2.3 Segmental Gene Duplication Result.....	24
4.3 Functional Design.....	26
4.4 Test Plan.....	34

Chapter Five (Implementation and Coding)

5.1 Introduction	35
5.2 Coding Programming Language.....	35
5.3 Establishment of Development Environment.....	37

Chapter Six (System Testing)

6.1 Introduction.....	42
6.2 Unit and Module Testing.....	43
6.3 Integration Testing.....	45
6.4 System Testing.....	46
6.5 Acceptance Testing.....	46
6.6 Sample Snapshots.....	46
6.6.1 Segmental Duplication Finder.....	46
6.6.2 Gene Duplication Result.....	47
6.6.3 Segmental Gene Duplication Result.....	48

Chapter Seven (Maintenance)

7.1 Introduction.....	50
7.2 Establishment of the production environment.....	50
7.3 Migration and Deployment Plan.....	50
7.4 Maintenance Plan.....	51
References.....	53
Appendix.....	54

List of Tables

Table	Page
Table 2.1 Development Hardware Costs	10
Table 2.2 Development Software cost	11
Table 2.3 Development Human Resource Cost	11
Table 2.4 Implementation Hardware Cost.....	12
Table 2.5 Implementation Software Cost.....	12
Table 2.6 Total Cost	13
Table 2.7 Time Schedule	15
Table 3.1 Data Dictionary.....	21
Table 6.1 Testing Schedule.....	42
Table 6.2 Information gene Test Cases.....	45

Figure 4.7 Extract information from output Megalign file browser.....	37
Figure 4.8 Searching to find the duplication on gene browser.....	37
Figure 5.1 Install Active Perl 5.8.7.8.11.....	38
Figure 5.2 Finish install DevSoft Perl Editor VS 6.0.2.....	39
Figure 5.3 DevSoft Perl Editor VS 6.0.2.....	40
Figure 5.4 Information gene Executive Paths browser.....	41
Figure 6.1 Segmental Duplication Result Snapshot.....	47
Figure 6.2 Gene Duplication Result Snapshot.....	48
Figure 6.3 Segmental Duplication Result Snapshot.....	49
Figure 7.1 Software Change Request Form.....	52

CHAPTER ONE

List of Figures

Figures	Page
Figure 1.1 Gene Duplication.....	2
Figure 1.2 Whole Genome Construction and Assembly Process	3
Figure 1.3 Detection of Gene Duplication	6
Figure 2.1 Gantt chart	15
Figure 3.1 System Data Flow	19
Figure 4.1 Segmental Duplication Finder	23
Figure 4.2 Gene Duplication Result	24
Figure 4.3 Segment Gene Duplication Result	25
Figure 4.4 determine the location genome data file Flowchart.....	26
Figure 4.5 Formatting data file Flowchart	27
Figure 4.6 Run MegaBlast Flowchart	28
Figure 4.7 Extract information from output MegaBlast file flowchart.....	31
Figure 4.8 Searching to find the duplication on gene flowchart.....	33
Figure 5.1 Install Active Perl 5.8.7.813.....	38
Figure 5.2 Finish install DzSoft Perl Editor V5.6.0.2.....	39
Figure 5.3 DzSoft Perl Editor V5.6.0.2	40
Figure 6.1 Information gene Execution Paths flowchart.....	44
Figure 6.2 Segmental Duplication Finder Snapshot.....	47
Figure 6.3 Gene Duplication Result Snapshot	48
Figure 6.4 Segmental Duplication Result Snapshot	49
Figure 7.1 Software Change Request Form.....	52

CHAPTER ONE

1

Introduction

Chapter One

- ▶ *Overview.*
- ▶ *Genome and Gene.*
- ▶ *Gene Duplication.*
- ▶ *How genome are sequenced.*
- ▶ *Detection of segmental duplication*

1.1. Overview

Computers and the World Wide Web are rapidly and dramatically changing the face of biological research. "Theoretical and computational biology have existed for decades on the "fringe" of biological science. But within just a few short years, the flood of new biological data that produced by genomics efforts and, by necessity, the application of computers to the analysis of this genomic data has begun to affect every aspect of the biological sciences. Research that used to start in the laboratory now starts at the computer, as scientists search databases for information that might suggest new hypotheses".¹

Bioinformatics is a new field of science. It refers to the application of computational tools and techniques to the management and analysis of biological data. The term bioinformatics is relatively new, and some time it is interchangeable with term "computational biology". In particular, bioinformatics is often the term used when referring to the data and the techniques used in large-scale sequencing and analysis of entire genomes.²

1.2. Genome and Gene:

Genome is the sum of all genetic material encased in every cell of a given organism. The genome contains the genetic material (DNA) is a form of either one or more very long strings (each called chromosome), that represent the specific order of 4-chemical units known as nucleotides (A, C, G, T).³

A gene can be defined as a substring of a given chromosome. It represents the specific order of the 4-nuclotides in that substring. According to biologist, gene is defined as the

¹ Jambeck, Per and Gibas, Cynthia. *Developing Bioinformatics Computer Skills*. 1ST Edition O'Reilly & Associates.2001.

² Lesk, Arthur. *Introduction to Bioinformatics*. Oxford University Press. 1ST Edition 2002

³ Lesk, Arthur. *Introduction to Bioinformatics*. Oxford University Press. 1ST Edition 2002.

unit of inheritance that encodes specific information which can dictate a specific biological function in the cell.

1.3. Gene Duplication:

Gene duplication is a very common phenomenon in most organisms. It occurs when an error in DNA replication leads to the duplication of a region of DNA containing a (generally functional) gene. The significance of this process for evolutionary biology is that if a gene is under natural selection, many mutations will lead to loss of functionality and thus are selected against. When a gene is duplicated selection may be removed from one copy and now the other gene locus is free to mutate and discover new functions. Alternatively, the gene may acquire deleterious mutations and become a pseudo-gene.⁴

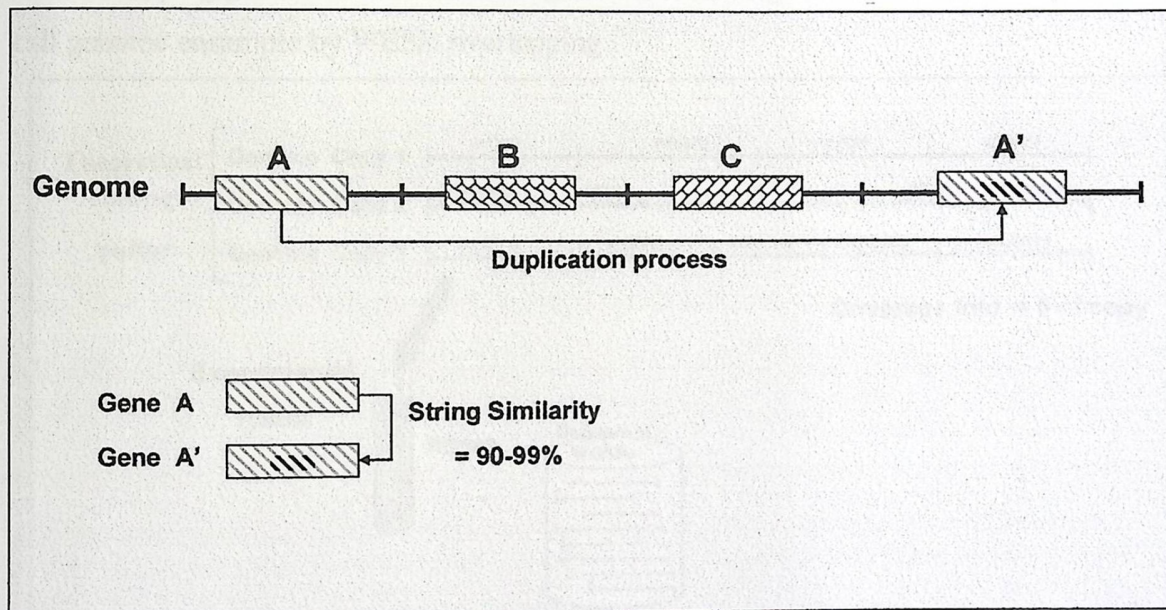


Figure [1.1] Gene Duplication Diagram.

Figure 1.1 shows a hypothetical genome as a single string chromosome that contains several genes (indicated by shaded boxes). The difference in shading between A (old copy) and A' (new copy) represents the new changes that were acquired by the recently duplicated copy A'.

⁴ Brown, T. A. *Genomes*. BIOS Scientific publisher. Oxford, UK 2nd Edition 2002.

1.4. How genome are sequenced:

Since the genomes are usually very long single or multiple chromosomes (strings), there is no biological technique to read and determine the whole sequence of a given genome at once. Therefore, biologists usually solve this technical problem by dividing the long chromosome into a large number of small fragments, these fragments are known as WGSRs (Whole Genome Shotgun Reads). The generation of these fragments is totally a random process, which means that the cutting process will generate a random number of WGSRs, each has a random length. After finishing the cutting process, each WGSR is sequenced by a simple biological technique and the order of its nucleotides (characters) is determined. The next step is to find the order of the fragment in relating to each other. This is achieved by having more than one string of the whole genome there afire, by find the overlapping parts of the different fragment the whole genome is built. This process is call genome ensample by WESR overlapping.⁵

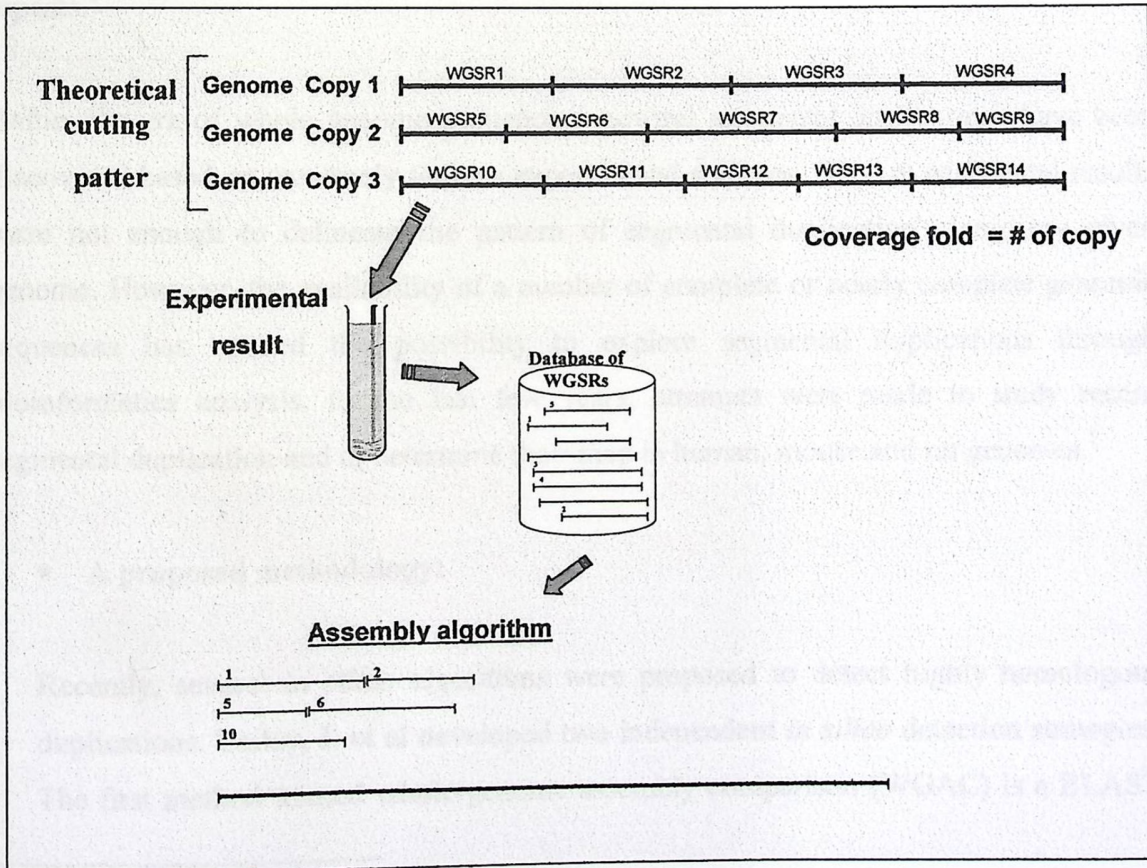


Figure [1.2] Whole Genome Construction and Assembly Process.

⁵ Brown, T. A. Genomes. BIOS Scientific publisher. Oxford, UK 2nd Edition 2002.

It is important to note that the number of copies of the genome (whole string) that is under sequencing and assembly process should be at least two in order to facilitate the process of finding overlapping tips in WGSRs. The number of genome copies is usually referred as coverage fold. For example, when three copies of a given genome are cut down into small fragments (WGSRs) in order to construct and assemble the whole genome, the coverage fold is said to be 3.

1.5 Detection of segmental duplication:

Recent segmental duplications are blocks of genomic DNA sequences that are highly similar (90%-98%) and they range in size from 1kb to 200 kb. segmental duplications can be divided into two classes; inter-chromosomal and intra-chromosomal. Due to their size, these segments can include full and/or partial genes, regulatory sequences and high-copy repeats.⁶

Before the era of whole genome sequencing, several segmental duplications have been discovered based on extremely tedious experimental analyses. Such experimental results were not enough to delineate the pattern of segmental duplication along any given genome. However, the availability of a number of complete or nearly complete genomic sequences has opened the possibility to explore segmental duplications through bioinformatics analysis. In the last few years, attempts were made to study recent segmental duplication and to determine their map in human, mouse and rat genomes.⁷

▪ A proposed methodology:

Recently, several *in silico* algorithms were proposed to detect highly homologous duplications. Bailey, J. et al developed two independent *in silico* detection strategies. The first method termed whole-genome assembly comparison (WGAC) is a BLAST

6 Eichler EE. **Recent duplication, domain accretion and the dynamic mutation of the human genome.** Trends Genet. 2001 Nov;17(11):661-9.

7 Bailey JA, Eichler EE. **Genome-wide detection and analysis of recent segmental duplications within mammalian organisms.** Cold Spring Harb Symp Quant Biol. 2003;68:115-24.

based approach that performs an all-to-all comparison of assembled genomic sequence.⁸ The second method have been developed assuming that the genome assembly is correct so it compare the WGSRs with the theoretically assembled genome.⁹

The method we present in this project is partially based on one of Bailey's approaches termed whole-genome shotgun detection (WSSD) with some modifications that were introduce to improve the performance and accuracy. Our system will be able to distinguish unique and duplicated sequence on the basis of:

1. The depth of coverage.
2. The average degree of sequence identity of whole-genome shotgun sequence reads aligned to each others

In essence, duplicated regions will show an increased depth-of-coverage and a significant reduction in the average degree of sequence identity, due to the matches with the duplicated new copies.

8 Bailey JA, Yavor AM, Massa HF, Trask BJ, Eichler EE. **Segmental duplications: organization and impact within the current human genome project assembly.** *Genome Res.* 2001 Jun;11(6):1005-17.

9 Bailey JA, Gu Z, Clark RA, Reinert K, Samonte RV, Schwartz S, Adams MD, Myers EW, Li PW, Eichler EE. **Recent segmental duplications in the human genome.** *Science.* 2002 Aug 9;297(5583):1003-7.

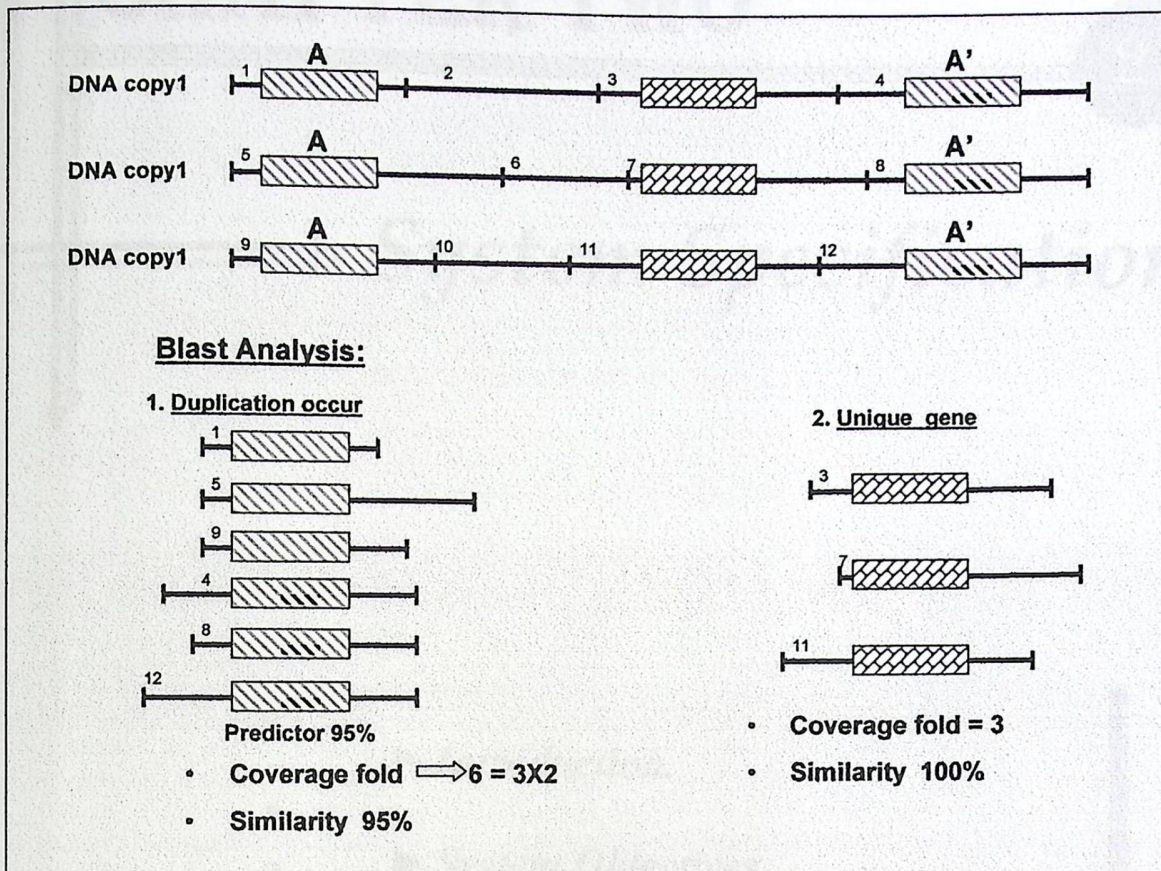


Figure [1.3] Detection of Gene Duplication.

▪ What is Blast

The Basic Local Alignment Search Tool (BLAST) is one of the most popular software tools in biological research. It tests a query sequence against a library of known sequences in order to find similarity. BLAST is actually a collection of programs with versions for query-to-database pairs such as nucleotide-nucleotide, protein-nucleotide, protein-protein, nucleotide-protein and more.¹⁰

¹⁰ Lesk, Arthur. *Introduction to Bioinformatics*. Oxford New York. First published 2002

CHAPTER TWO

2

System Specification

C h a p t e r T w o

- ▶ *Introduction.*
- ▶ *System Objectives.*
- ▶ *Functional Requirement.*
- ▶ *Non-Functional Requirement.*
- ▶ *Allocation of Roles of System Developers.*
- ▶ *Feasibility Study.*

2.1 Introduction

In this chapter I will describe the system specifications. The description will include the following topics:

1. System objectives.
2. Functional and Non-Functional requirements.
3. Allocation of roles of system developers.
4. Constraints.
5. Feasibility study (Alternatives, Cost-Benefit analysis, and Risks Analysis).
6. Resources and Cost.
7. Time Schedule for development.

2.2 System Objective

The overall goal of this system is to develop efficient software that can identify segmental duplication in genome of interest to molecular biology researcher.

2.3 Functional requirements

The followings are the functional requirements that should provide by the system:

1. The system should be able to determine the location of the genome data file.
2. The system should be able to determine number of coverage fold.
3. The system should perform data formatting that is necessary for the subsequent Mega-Blast program.
4. The system should be able to run the Mega-Blast program on the formatted data.
5. The system should be able to analyze the Mega-Blast output in order to determine the duplicated segments in the tested genome sequences.

2.4 Non-Functional requirements

A group of non-functional requirements is highly demanded in order to have a useful and easy applicable system. These non-functional requirements include:

- **Product Requirement**

1. User friendly: attractive and competitive methodology for displaying interface.
2. Usability: easy and simple for use GUI.
3. High speed program: The system enables the user to perform all the processes in a short period of time, and take the results of the processes very quickly.
4. High performance system: The system will provide very high quality output, with less cost.
5. Portability: running on different operating systems without any problems on the application.
6. Accuracy: the system must provide a high level of accuracy.

- **Process requirement**

The system and its explanatory documentation must be delivered on 18 January, 2006.

2.5 Allocation and roles of system developers:

1. Leader: responsible of planning, scheduling and controlling flow of system development processes.
2. Programmer: responsible of the system programming, implementation testing, so he must have enough experiences in Perl development environment.
3. Software engineer: responsible for the documentation and tracing of the development stages of the software.

2.6 Feasibility Study

For a system to be developed from scratch, the most important issue is to evaluate its benefits versus its cost.

In this section, we will describe the system alternatives that could be adapted; a Cost-Benefit analysis is conducted to justify the decision for developing the system, and an evaluation of the risks that may face the system and the development process.

2.6.1 Alternatives:

- **Environment:**

We will use the windows XP as an operating system to implement this program. We decided to use this operating system because of its easiness, flexibility and wide distribution.

Perl is a popular programming language that's extensively used in areas such as bioinformatics and web programming. Perl has become popular with biologists because it's so well-suited to several bioinformatics tasks.

The following sections illustrate some of Perl's strong points.

1. Ease of Programming.
2. Free Source.
3. Rapid Prototyping.
4. Portability, Speed, and Program Maintenance

So I have chosen this language for the previous reasons.

2.6.2 Cost-benefit analysis

Experimental analysis was the only way for molecular biologist to determine the nature and location of segmental duplication in the genome of interest. In addition to include laborious procedure, such experimental work would require at least one to three years (ref). In contrast to the tedious experimental work, the proposed bioinformatics system would provide answers for biological questions in very short time. The other advantage of this system over the experimental procedure is the possibility to apply it to any genome that would be sequenced in the future.

Biology Provider benefits:

1. Decrease the biology time and efforts works.
2. Increase the efficiency by decreasing the error rate during process.
3. Significantly reducing the cost of such research activities.

2.6.3 Economical Study

In this section, resources and costs are described both for the development and implementation requirements.

- **Development Costs :**

Hardware

The following table lists the costs for the hardware that needed to develop this project:

No.	Item	Quantity	Specifications	Cost
1.	Microsoft compatible PC	1	Pentium 4 2400 MHz 512KB cache memory RAM 512 MB Hard Disk drive 40 GHz Floppy drive 1.44 CD-ROM 52X Monitor 15" Keyboard and mouse	\$900
2.	Printer	1	HP 3420	\$85
Total				\$985

Table [2.1] Development Hardware Costs

Software

Table [2.2] shows the software products required and their costs for the system development and implementation.

No.	Software	Cost
1.	Windows XP professional	\$200
2.	Active Perl-5.8.7.813 - MSWin32	Free
3.	BLASTALL	free
4.	DzSoft.Perl.Editor.v5.6.0.2.WinALL	\$80
5.	Microsoft Office2003	\$450
Total		\$730

Table [2.2] Development Software cost

Human Resource cost:

As shown in Table [2.3] the team costs are estimated as the market prices for software developers.

Number	Team Role	Hours/week	Cost/hour	Total
1	Programming	20	\$5	\$100
2	Software Engineering	20	\$5	\$100
3	Interface Design GUI	20	\$5	\$100
Total cost/week				\$300

Table [2.3] Development Human Resource Cost.

- **Implementation Cost**

This section lists the cost needed to implement this project:

Hardware:

The system can work on a computer Pentium II but it is better to work on a

computer which has these qualifications.

No.	Item	Quantity	Specifications	Cost
3.	Microsoft compatible PC	1	Pentium 4 2400 MHz 512KB cache memory RAM 512 MB Hard Disk drive 40 GHz Floppy drive 1.44 CD-ROM 52X Monitor 17" Keyboard and mouse	\$900
Total				\$900

Table [2.4] Implementation Hardware Cost

Software

In this section the software needed to implement this project listed:

No.	Software	Cost
1.	Windows XP	\$200
2.	BLASTALL	free
Total		\$200

Table [2.5] Implementation Software Cost.

▪ **Other accessories cost:**

Other costs such as books, papers, pens, and transportations are estimated to be \$20/ week.

▪ **Total Cost:**

Table 2.6 below represents the total cost of all system cost:

Number	System Cost	Total
1	Development cost	\$5215
2	Implementation cost	\$1100
4	Other cost	\$300
Total cost		\$6615

Table [2.6] Total Cost

2.6.4 Risk Analysis

This section contains the risks that may appear in the project, and the possible solutions:

1. The Israeli occupation has began an arrest campaign in the west bank which may disruptive our project because no one can predict the future.

The project was divided between me and him so I will face a problem to fill the gap of his absence.

2. Shortage of development time:

The time that specified to develop such as project is not enough as we hope to be. I may face problems in collection information specially I have to work alone at this project

3. We have little biology background which would help us developing this system.

For that reason I have to spend a lot of time surfing the web, visiting the library, and asking people who are specialized in this field.

2.6.5 Technical feasibility

This project requires a programming experience in Perl. Team work members have fairly good experience and capabilities to develop such applications. They have experience in different programming languages such as C, Visual Basic, Perl and others. In addition, we can solve some obstacles through seeking help from experts people in the university or through bioinformatics forums on the web.

2.6.6 Legal feasibility

Perl is distributed under the GNU's General Public License, which implies that anyone can use, modify, and distribute the source code and documentation of perl. Any modifications made to the source code derived from the GNU-licensed source code must be freely made available to other. This distribution model has encouraged volunteers worldwide to contribute to the perl software.

2.6.7 Time Feasibility

In this section we show how we have allocated the given period over the development stages. The time interval that was available to develop the system was 15 weeks. We have distributed this interval over all of the development process. Table [2.6] shows the time schedule for all development tasks.

- **Time Schedule:**

As shown below in Table [2.7], all system development tasks are distributed over the available fifteen weeks. Some of these tasks were performed in parallel. Figure [2.1] shows the timeline distribution precisely.

Task	Work	Time in weeks
T1	Information gathering and System specification.	2
T2	Software requirement specification.	2
T3	System Design.	6
T4	Coding and implementation.	6
T5	System Testing.	3
T6	System Maintenance.	2
T7	Documentation.	15

Table [2.7] Time Schedule

- Gantt chart for time schedule:

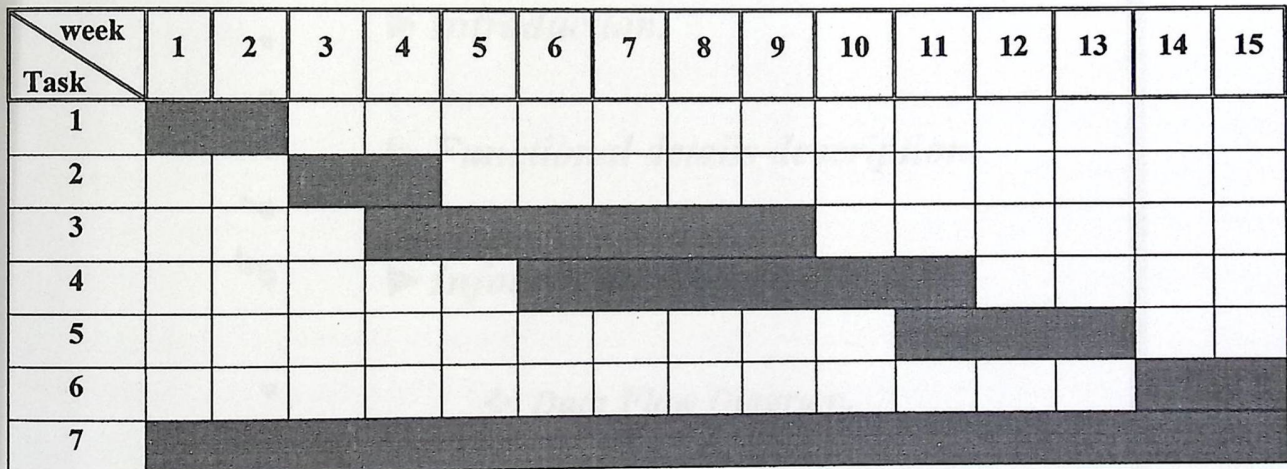


Figure [2.1] Gantt chart

CHAPTER THREE

3

Software Requirements Specifications

Chapter Three

▶ *Introduction.*

▶ *Functional details description.*

▶ *Information Description.*

❖ *Data Flow Diagram.*

❖ *Data Dictionary*

3.1 Introduction

In this section the software specifications will be addressed and identified in more technical terms, and in more details.

In this section we will cover:

- Functional description of our system, in which all the supported functions and services will be identified and modeled.
- Behavioral models in which the behavior of the system will be modeled by using a data flow diagram.
- Data dictionary in which a complete description of each system entity will be provided, with its name, type, input and output.

3.2. Functional Detail Description:

This section lists the major functions in the project and a description for each.

Function:	The system should be able to determine the location of the genome data file.
Description:	In this part of the system, the user can determine the path of the query using file browser or using file name.
Input:	Specifying the file path.
Source:	The place that contains the file, it may be on some location on drive C.
Output:	Place that contains the file; it may be on some location on drive C.
Destination:	A textbox that will be filled by the query.
Require:	nothing.
Precondition:	Test the existence of the file.
Post condition:	nothing.

Function: Coverage fold.

Description: In this part of the system, the user can determine the number of whole copy gene.

Input: number of coverage fold.

Source: the user of system

Output: nothing.

Destination: A textbox that will be filled by number.

Require: nothing.

Precondition: Test the existence of the numeric.

Post condition: nothing.

Function: The system should perform data formatting that is necessary for the subsequent Mega-Blast program

Description: In this part of the system, the systems will change the file to new formatting, to give MeagBlast the ability of reading it.

Input: The file which is selected by the user.

Source: A textbox that will be filled by the Data.

Output: New formatting of the file.

Destination: The input of the MeagBlast.

Require: Determine the location of the genome data file.

Precondition: Test the existence of the Data.

Post condition: Run MegaBlast to get similar EST's.

Function: Run MegaBlast

Description: This program is optimized for aligning sequences that differ slightly as a result of sequencing or other similar

Input: Formatting data file.

Source: Data file.

Output: file results from MegaBlast program which can be run through the system. It contains the headers of the EST's, the query gene aligned with each EST and information about alignment process.

Destination: To analyzing the MegaBlast output.

Require: Make formatting data.

Precondition: Test the existence of the formatting data.

Post condition: Run MegaBlast to get similar EST's.

Function: Analyze to determine the duplicated segments in the tested genome sequences

Description: Apply the defined criteria to predict if the string contains a duplicated segment.

Input: Output MegaBlast file.

Source: Data file.

Output: Dublication on genome.

Destination: nothing.

Require: MegaBlast Output file.

Precondition: Test the existence of the MegaBlast output file.

Post condition: display the result of anlayze.

3.3 Information Description

This section talks about the data and information in the project, describe the dataflow, and a description for each entities names.

3.3.1 System Data Flow Diagram (DFD)

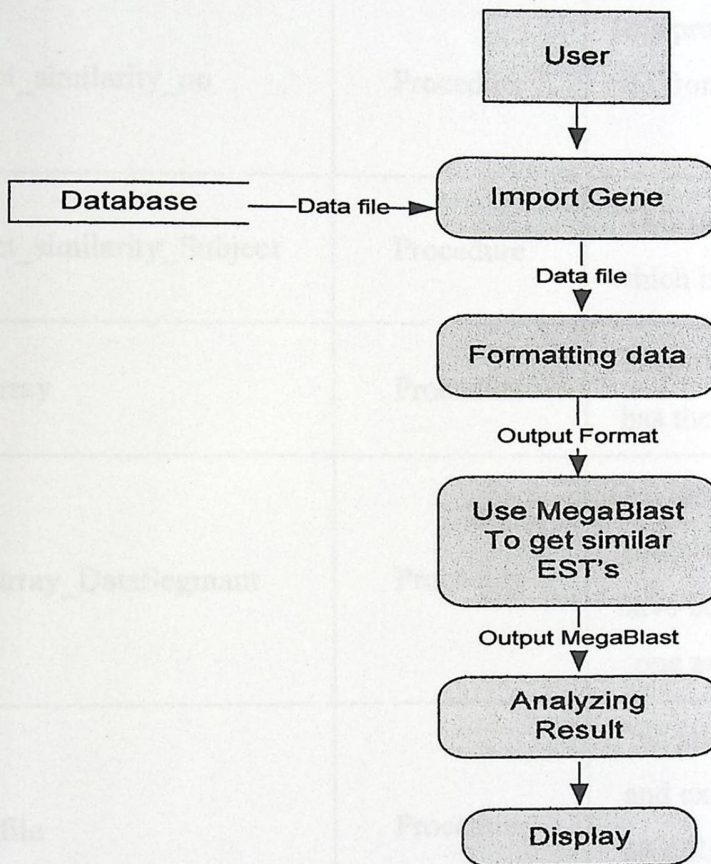


Figure [3.1] System Data Flow

3.3.2 Data dictionary:

Entity name	Type	Description
Reads_WGSR	Procedure	This procedure takes the files that the MegaBlast has carried out then it reads data and extracts the wanted information
extract_similarity_no	Procedure	This procedure extracts the similarity rate from the files that the MegaBlast has carried out.
extract_similarity_Subject	Procedure	This procedure extract the subjects which have similarity with WGSRs .
sort array	Procedure	This procedure orders the array which has the extracted data from the file.
sort_array_DataSegmant	Procedure	This procedure collects the arrays that contain all the collected data which have been extracted from all files in one array then it orders the array.
filter file	Procedure	This procedure investigates all arrays and examines if it have duplication WGSRs and appears it, if there is no it remove the array.
Run_MegaBlast	Function	This function runs the MegaBlast for each WGSR and compares it with all database file
Display_WGSR_Duplication	Procedure	This Procedure appears WGSRs which had duplication as a report
Collection_WGSR	Procedure	This Procedure collects all WGSRs which had duplication then it collect all of them to gather as segments.

MakeFile_Segment	Procedure	This procedure collect the segments which have been collected at the previous point and it puts them in one file
Rur_megablast_Segment	Procedure	This procedure brings the segment and compares it with database file by the MegaBlast.

Table [3.1] Data Dictionary

CHAPTER FOUR

4

System Design

Chapter Four

- ▶ *Introduction.*
- ▶ *Input/output design.*
 - ❖ *Segmental Duplication Finder*
 - ❖ *Gene Duplication Result*
 - ❖ *Segmental Gene Duplication Result*
- ▶ *Functional Design.*
- ▶ *Test Plan*

4.1 Introduction

In this section the functions design will be implemented using functional oriented methodology, where each function will be designed accordingly.

The following will be covered in this section

- Input output design: a design for the input output screens.
- Functions design: where each function will be designed by using a flow chart, its interface and constraints will be identified.
- Test plan: a test plan will be identified.

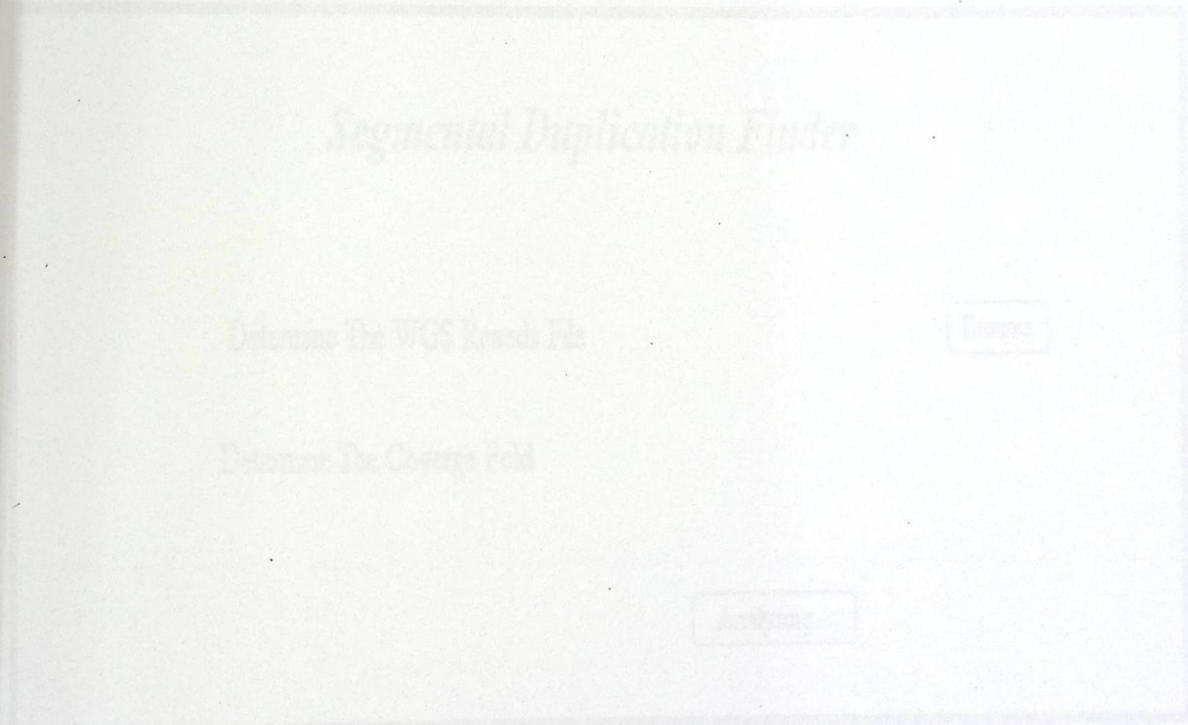
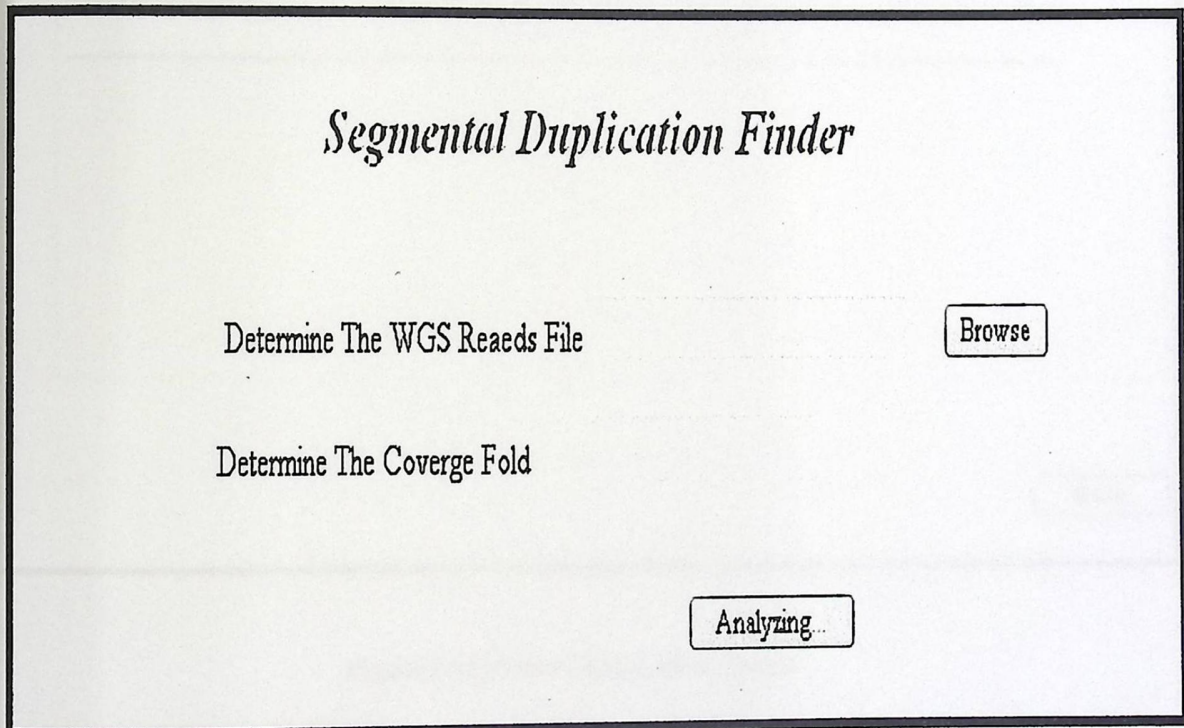


Figure (4.1) Segmental Duplication Finder

4.2 Input / Output Design

4.2.1 Segmental Duplication Finder

This interface allows user to enter the necessary information to search and to find the duplication on gene file. The figure [4.1] shows two text boxes. In the first text box the user will enter the name and location of the WGS reads file. The second text box the user will enter the coverage fold. After click on the (Analyzing) button, the program will check the existing and validity of the WGS reads file in first text box, Check the validity of coverage fold in the second one.



The screenshot shows a window titled "Segmental Duplication Finder". Inside the window, there are two text input fields. The first field is labeled "Determine The WGS Reaeds File" and has a "Browse" button to its right. The second field is labeled "Determine The Coverge Fold". At the bottom center of the window, there is an "Analyzing.." button.

Figure [4.1] Segmental Duplication Finder

4.2.2 Gene Duplication Result

This screen display the result of duplication on WGS reads file that the project found it.

The screenshot shows a web application interface with a title bar that reads "Gene Duplication Result". Below the title bar, there is a section labeled "Header of the gene duplication:" followed by a "Report:" section. The report section contains three horizontal lines, with three dots centered below them, indicating a list of items. At the bottom right of the interface, there is a "Back" button.

Figure [4.2] Gene Duplication Result

4.2.3 Segmental Gene Duplication Result

This screen display the result of duplication on Segmental WGS reads that the project found it.

Segmental Gene Duplication Result

Header of the Segmental gene duplication:

Report :

Figure [4.3] Segment Gene Duplication Result

4.3 Functions design

This section describes the functional design for each module in the software system.

- **Determine the location genome data file :**

In this function we will describe logical flow to determine the location of file and the coverage fold, then we will check the validity of them and averting the error in the next function.

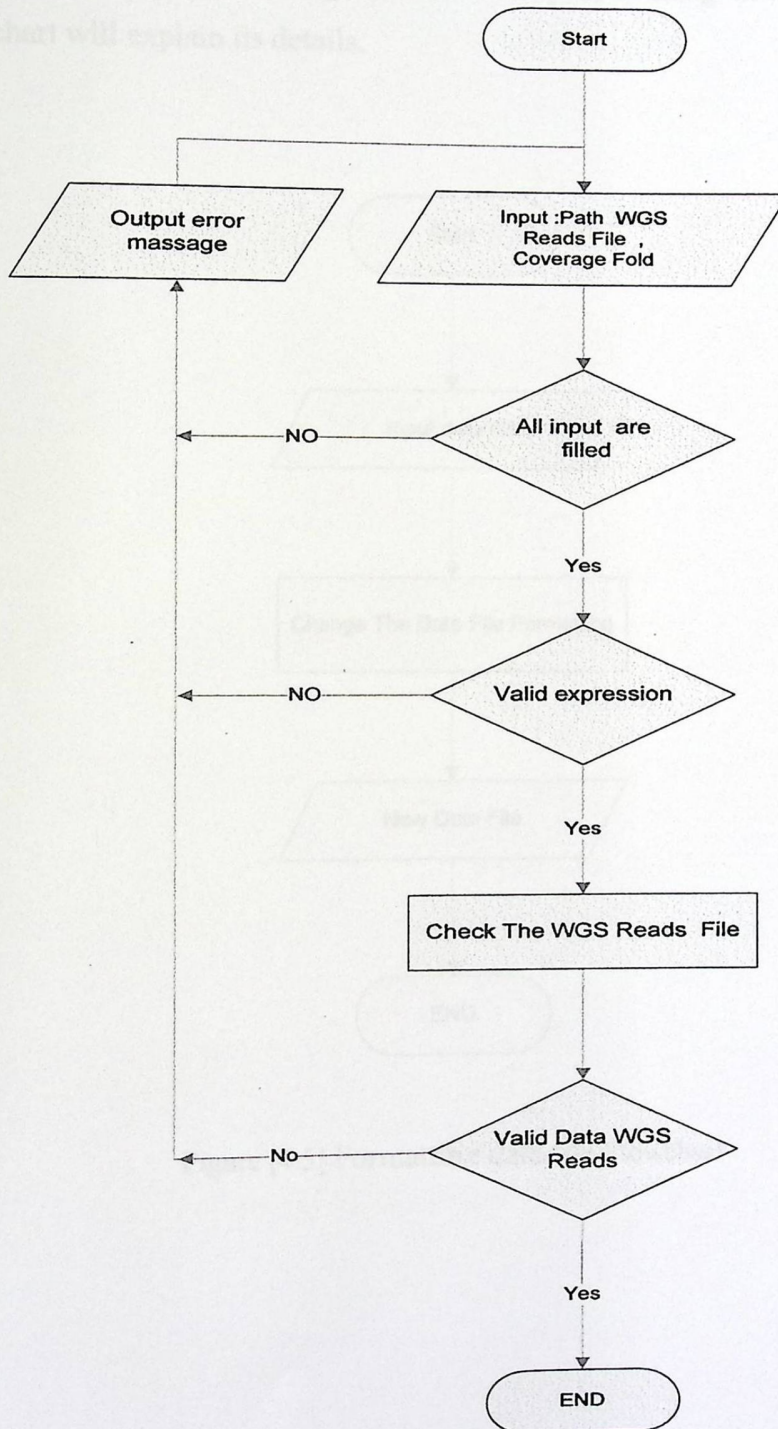


Figure [4.4] determine the location genome data file Flowchart

- **Formatting The Data File**

In this function we will describe logical flow to change the file to new formatting and to give MeagBlast the ability of reading it. The following flowchart will explain its details.

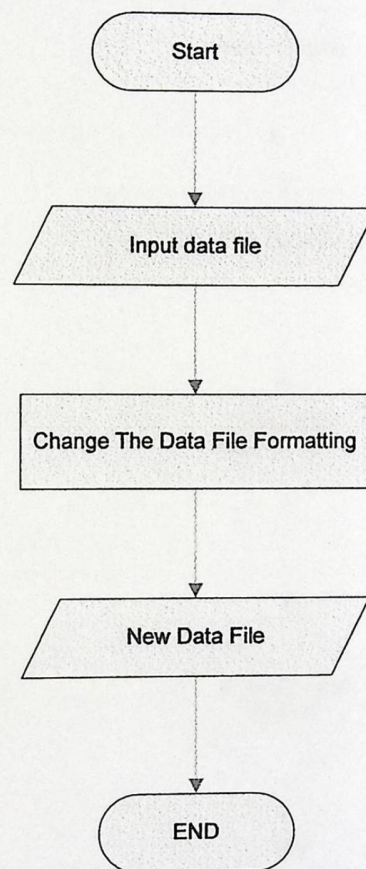


Figure [4.5] Formatting data file Flowchart

- **Run MegaBlast**

In step, Function will run the MegaBlast to produce optimized for aligning sequences that differ slightly as a result of sequencing or other similar. Execute MegaBlast on all WGS reads.

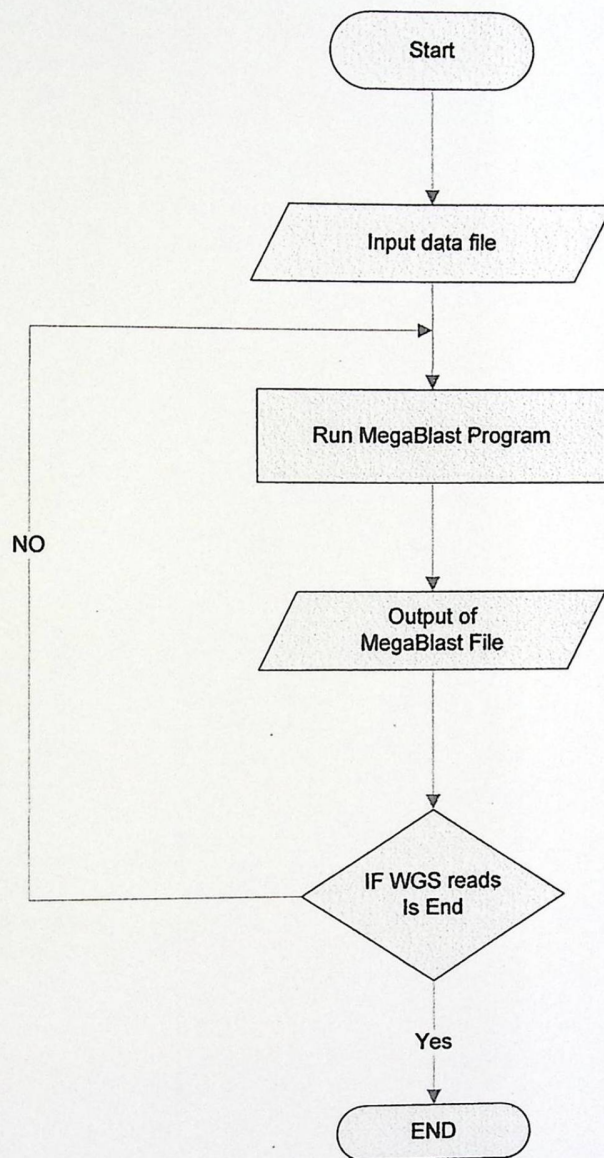


Figure [4.6] Run MegaBlast Flowchart

- **Run MegaBlast**

In step, Function will run the MegaBlast to produce optimized for aligning sequences that differ slightly as a result of sequencing or other similar. Execute MegaBlast on all WGS reads.

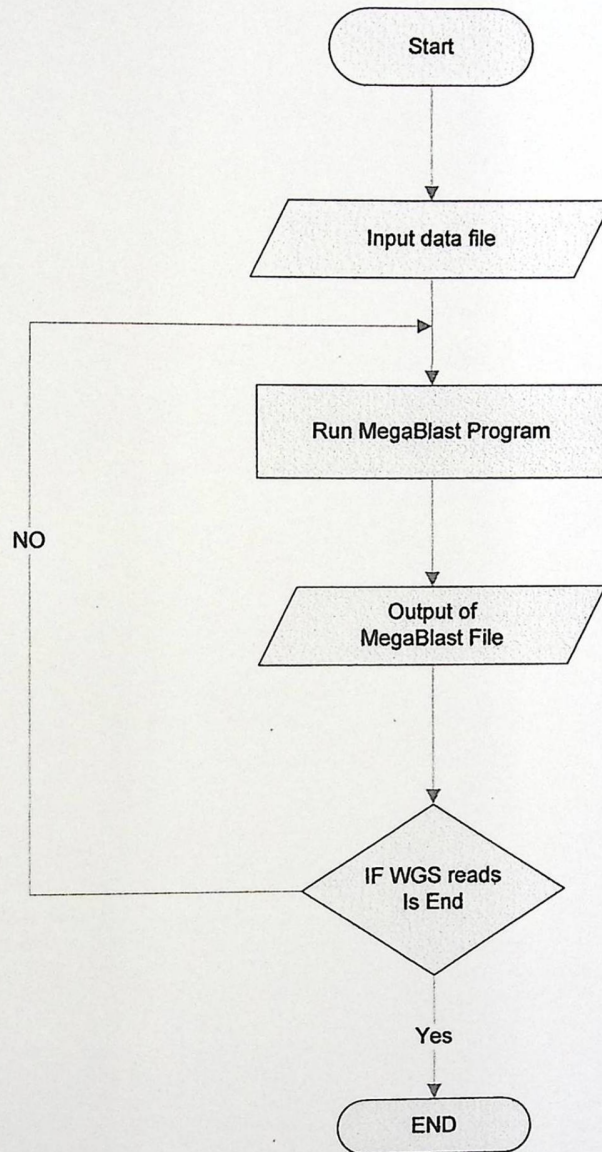


Figure [4.6] Run MegaBlast Flowchart

- **Run MegaBlast**

In step, Function will run the MegaBlast to produce optimized for aligning sequences that differ slightly as a result of sequencing or other similar. Execute MegaBlast on all WGS reads.

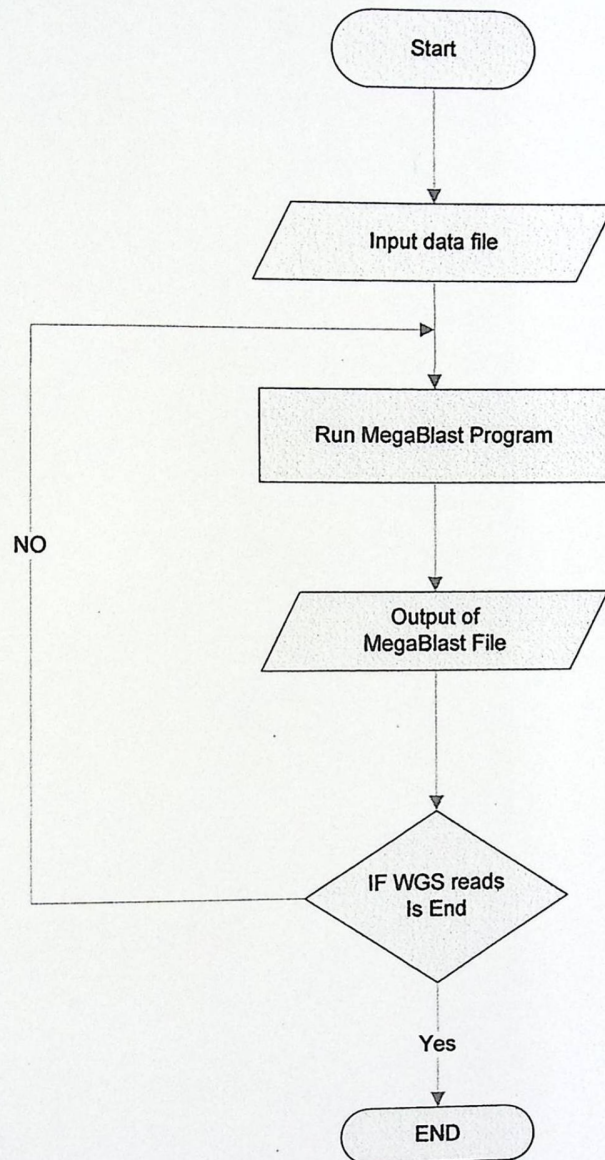


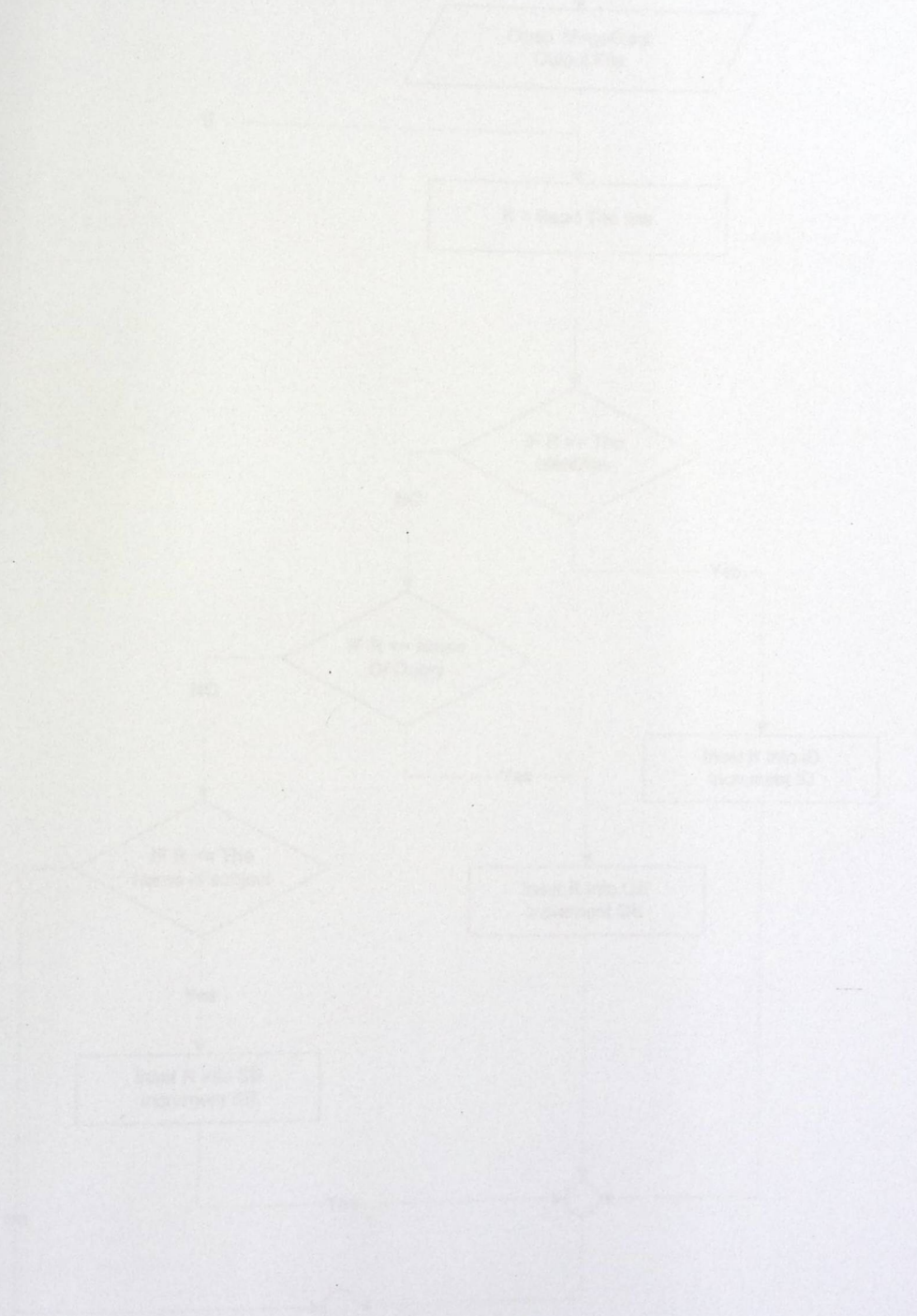
Figure [4.6] Run MegaBlast Flowchart

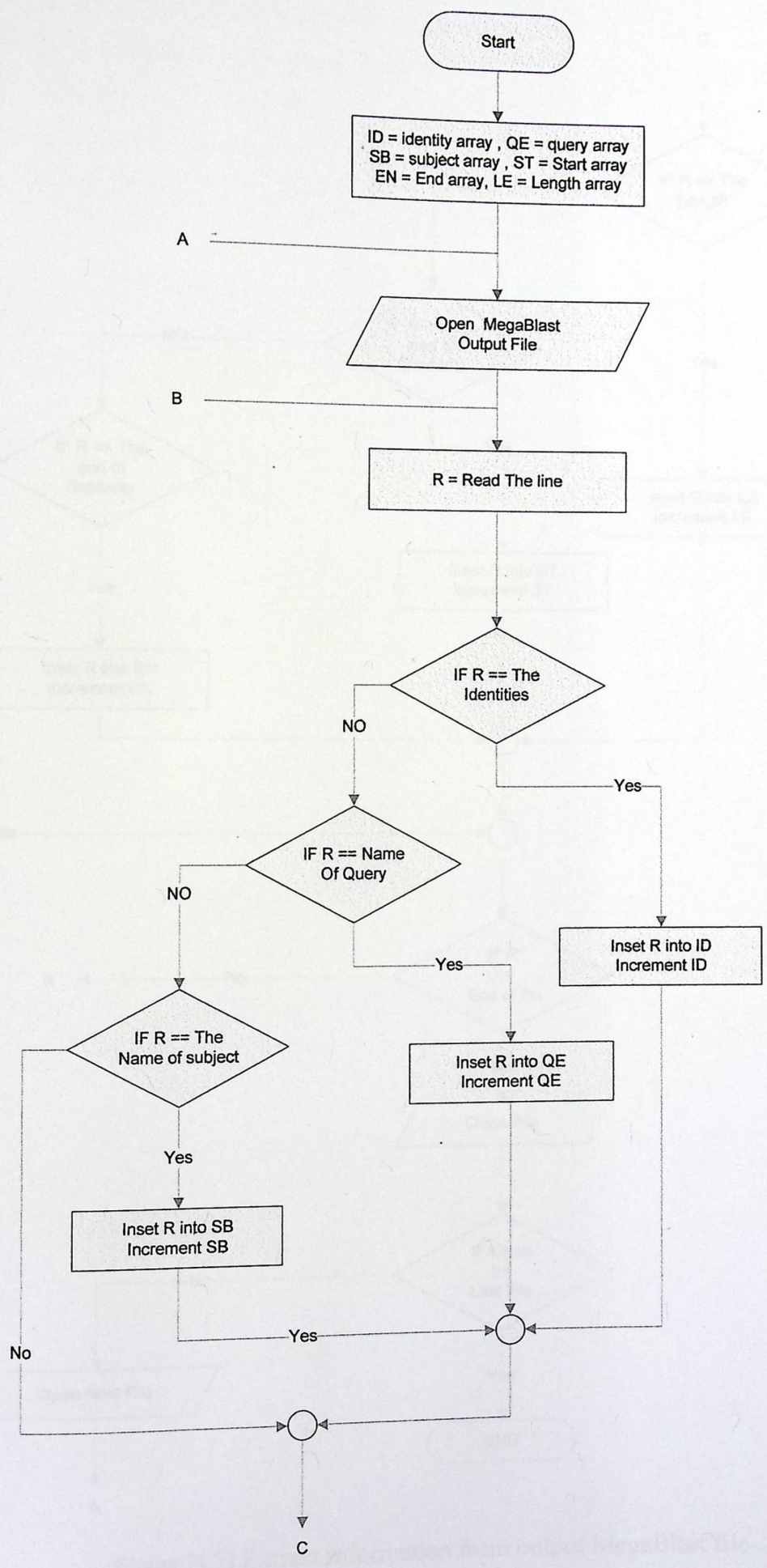
- **Analyzing Output MegaBlast**

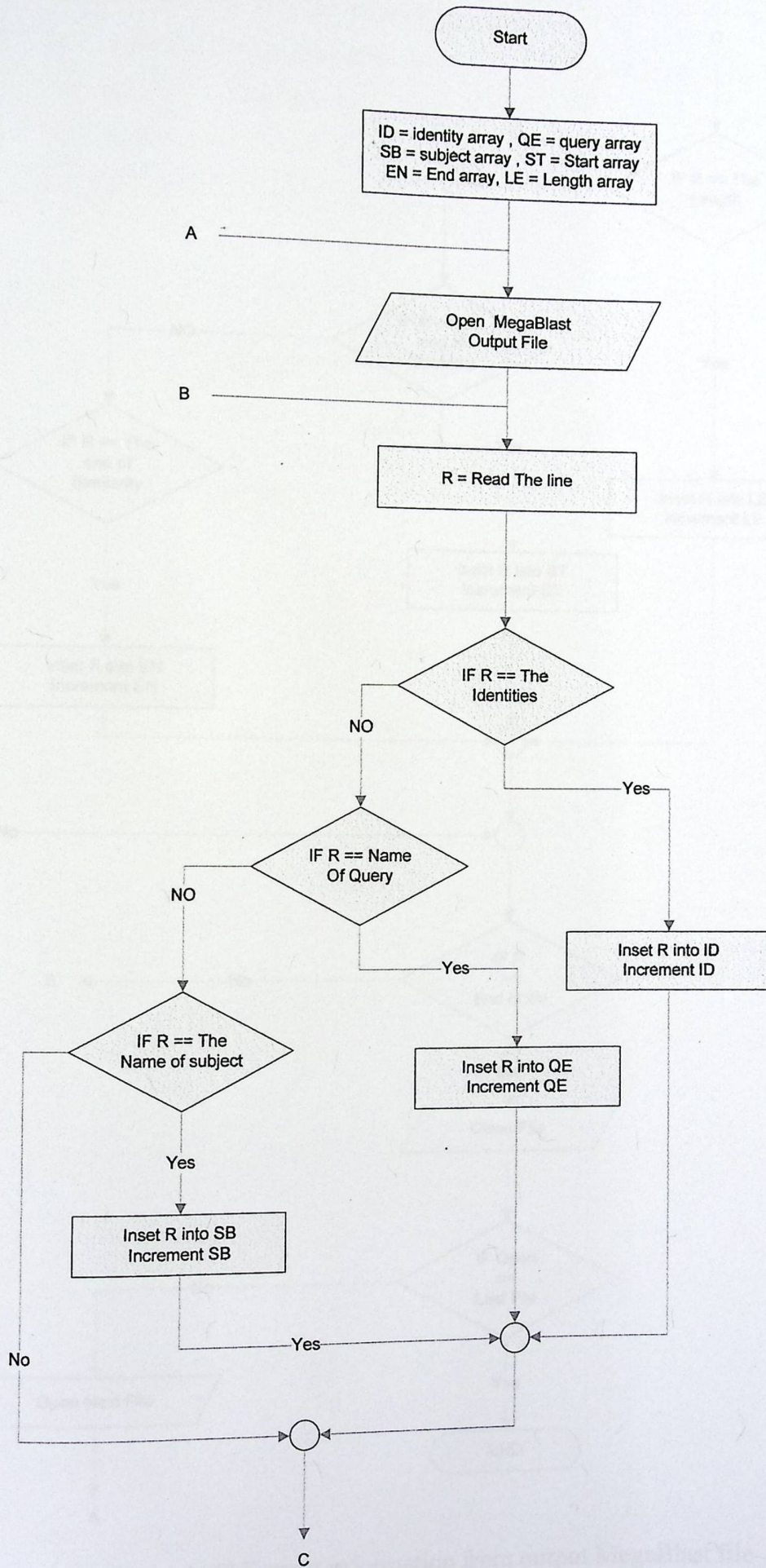
In this function we will describe two functions that are related to it:

1. **Extract information from output MegaBlast file**

In this part, we will make processing for the output MegaBlast file which will produce and extracted information that is essential to start searching the gene duplication, the following flow chart explains this clearly:







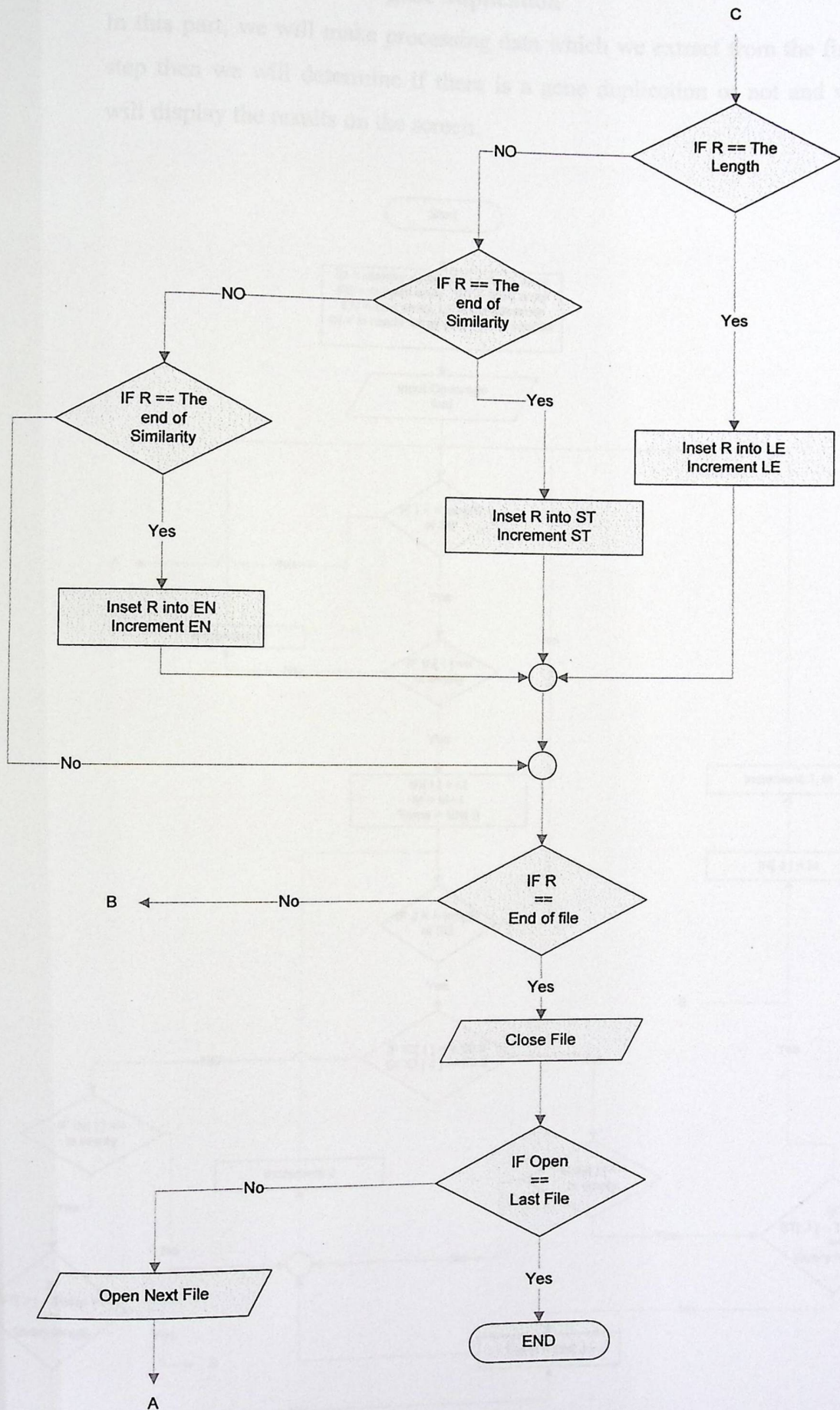
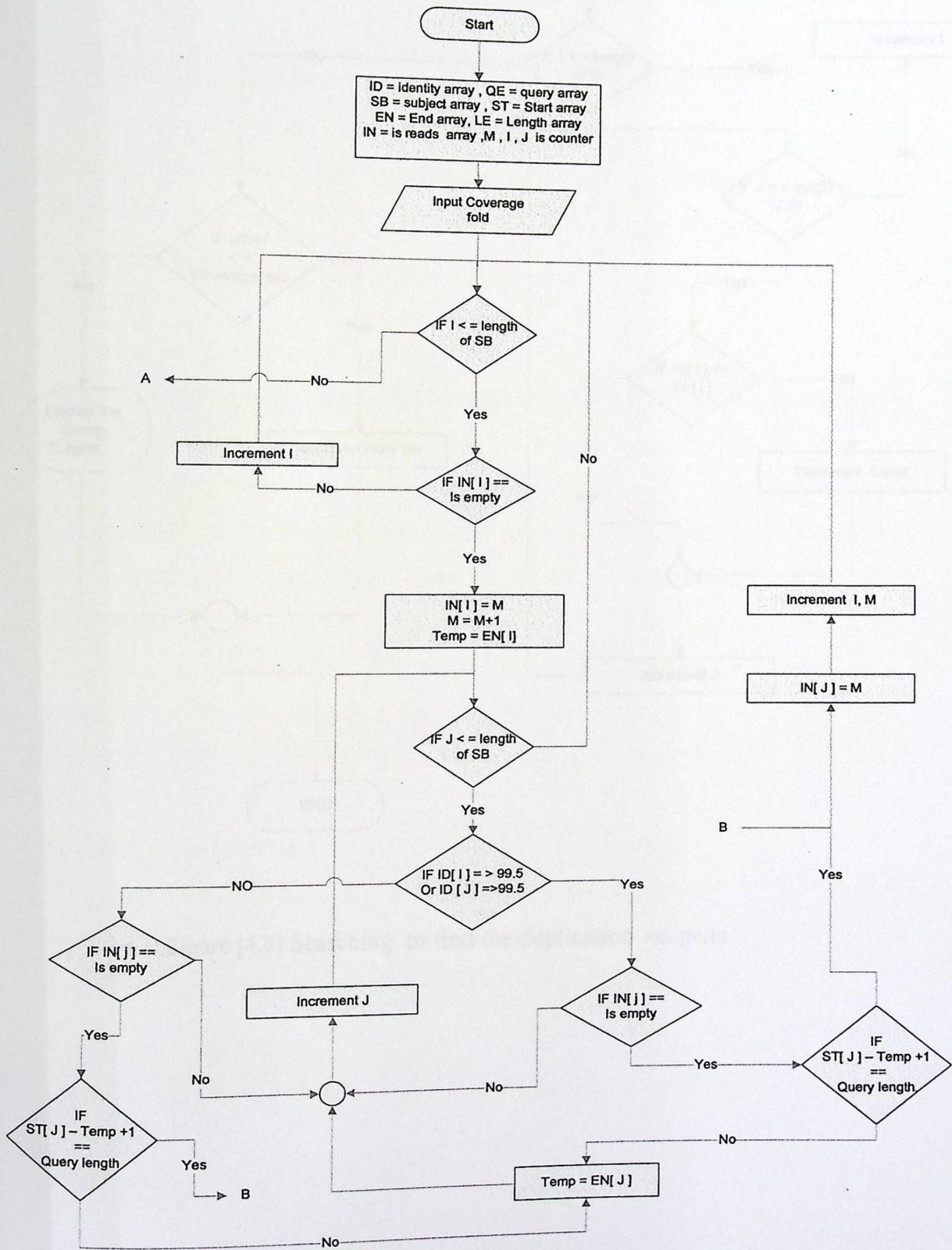
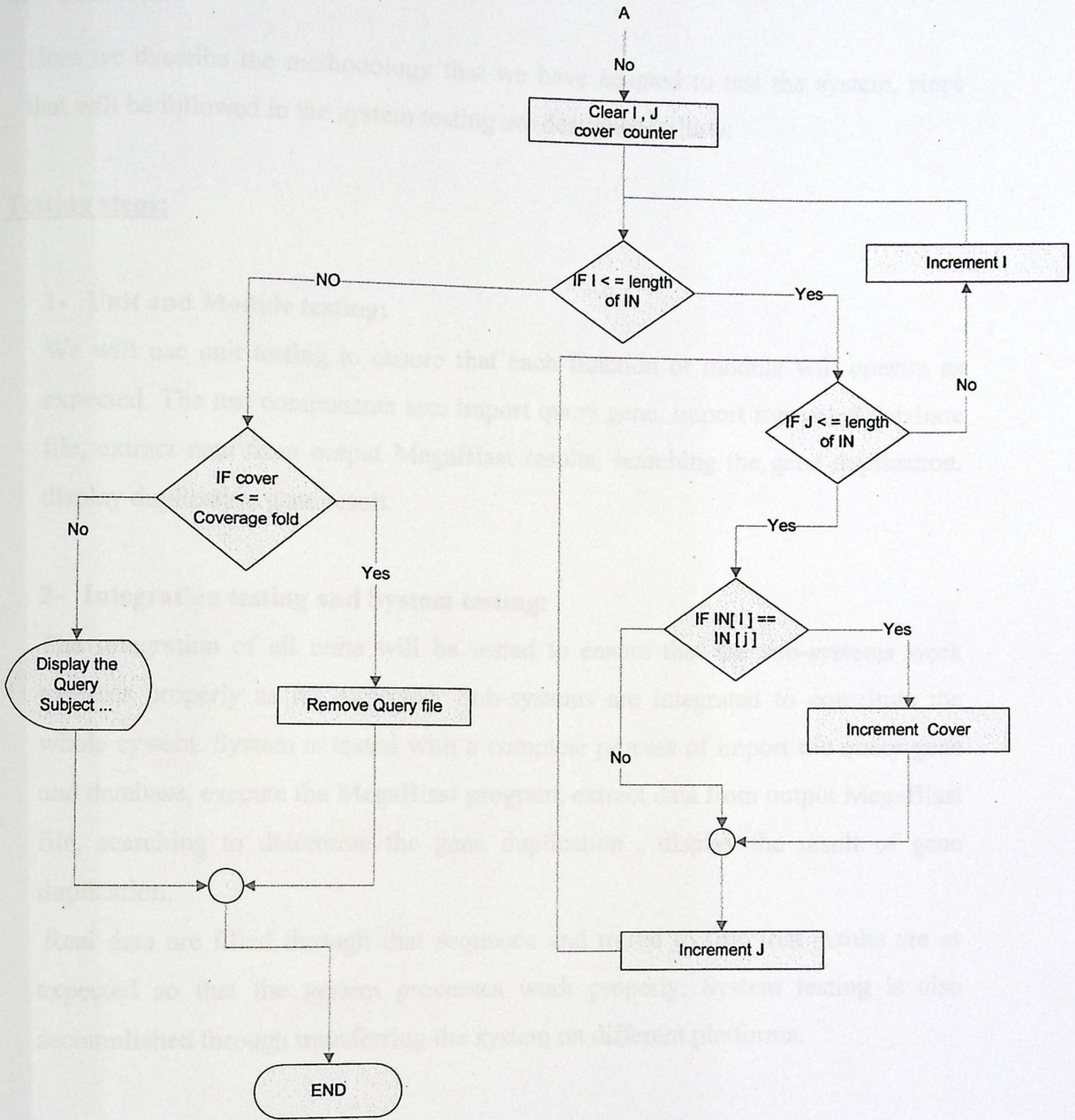


Figure [4.7] Extract information from output MegaBlast file

2. Searching to find the gene duplication

In this part, we will make processing data which we extract from the first step then we will determine if there is a gene duplication or not and we will display the results on the screen.





• Figure [4.8] Searching to find the duplication on gene

4.4 Test Plan

Here we describe the methodology that we have adapted to test the system, steps that will be followed in the system testing are described below:

Testing steps:

1- Unit and Module testing:

We will use unit testing to ensure that each function or module will operate as expected. The test components are; import query gene, import formatted database file, extract data from output MegaBlast results, searching the gene duplication, display duplication gene result.

2- Integration testing and System testing:

The integration of all units will be tested to ensure that the sub-systems work together properly as it's expected. Sub-systems are integrated to constitute the whole system. System is tested with a complete process of import the query gene and database, execute the MegaBlast program, extract data from output MegaBlast file, searching to determine the gene duplication , display the result of gene duplication.

Real data are filled through that sequence and traced to find that results are as expected so that the system processes work properly. System testing is also accomplished through transferring the system on different platforms.

3- Acceptance Testing:

A system is developing for a single user. The acceptance testing process continues until the system developer and the user agree that the delivered system is an acceptable implementation of the system requirements.

CHAPTER FIVE

5

Implementation & Coding

Chapter Five

- ▶ *Introduction.*
- ▶ *Coding Programming Language*
- ▶ *Establishment of development environment*

5.1 Introduction

This chapter describes the main steps must be followed to start in coding and programming to reach the design that is described in the previous chapters, and to discover the programming language that is used for this purpose.

This chapter focuses on the coding and implementation of novel algorithm to determine highly homologue segmental genome duplication project.

Coding refers to the process of writing the necessary program, which implements the main procedures and functions of the project. The code of the project is to be written from the scratch using perl language.

The project is to be implemented as a windows application, the project is to be programmed under windows XP operating system.

5.2 Coding Programming Language

There are many languages that can be used to develop a system such ours, but the most effective languages are visual c, visual basic, Perl language, here we describe the why our selection was on the Perl:

Perl is a popular programming language that's extensively used in areas such as bioinformatics and web programming. Perl has become popular with biologists because it's so well-suited to several bioinformatics tasks.

The following sections illustrate some of Perl's strong points.

1. Ease of Programming :

Computer languages differ in which things they make easy. By "easy" I mean easy for a programmer to program. Perl has certain feature that simplifies several common bioinformatics tasks. It can deal with information in ASCII text files or

flat files, which are exactly the kinds of files in which much important biological data appears, in the GenBank and PDB databases, among others.

2. Free Source :

Perl is distributed under the GNU's General Public License, which implies that anyone can use, modify, and distribute the source code and documentation of perl. Any modifications made to the source code derived from the GNU-licensed source code must be freely made available to other. This distribution model has encouraged volunteers worldwide to contribute to the perl software.

3. Rapid Prototyping

Another important benefit of using Perl for biological research is the speed with which a programmer can write a typical Perl program (referred to as rapid prototyping). Many problems can be solved in far fewer lines of Perl code than in C or Java. This has been important to its success in research. In a research environment there are frequent needs for programs that do something new, that are needed only once or occasionally, or that need to be frequently modified. This rapid prototyping ability is often a key consideration when choosing Perl for a job

4. Portability, Speed, and Program Maintenance

Portability means how many types of computer systems the language can run on. Perl has no problems there, as it's available for virtually all modern computers found in biology labs. If you write a DNA analyzer in Perl on your Mac, then move it to a Windows computer, you'll find it usually runs as is or with only minor retrofitting.

Speed means the speed with which the program runs. Here Perl is pretty good but not the best. For speed of execution, the usual language of choice is C. A program written in C typically runs two or more times faster than the comparable Perl program. In many organizations, programs are first written in Perl, and then only

the programs that absolutely need to have maximum speed are rewritten in C. The fact is, maximum speed is only occasionally an important consideration.

Program maintenance is the general activity of keeping everything working such as; adding features to a program, extending it to handle more types of input, porting it to run on other computer systems, fixing bugs, and so forth. Programs take a certain amount of time, effort and cost to write, but successful programs end up costing more to maintain than they did to write in the first place. It's important to write in a language, and in a style, that makes maintenance relatively easy, and Perl allows you to do so.

I have appended the source code written manually for the main functionalities in our application.

5.3 Establishment of Development Environment

- Purchase the computers and the software required for developing the system.
- Install windows XP.
- Install the required utilities.
- Install the Active Perl 5.8.7.813

Before installing the active perl some windows Prerequisites must be available its as following:

- Hardware: 90 MB hard disk space for typical install
- Perl for ISAPI: requires an ISAPI-compatible web server, such as IIS 4.0 or greater, or PWS 4.0 or greater
- Perl Script: requires an ActiveX scripting host such as Internet Explorer 4.0 or greater or Windows Scripting Host
- Perl Environment Variables: if Perl environment variables such as PERLLIB, PERL5LIB or PERL5OPT have been set on your system, you should unset them before installing Active Perl. Otherwise, these variables may cause

incompatible versions of Perl modules to be used during the installation process.

To install the active perl

- Download the active perl 5.8.7.813 on the internet from website www.activestate.com
- After download Run ActivePerl-5.8.7.813 .EXE.
- After determining the location of perl and installing it the following figure will appear.

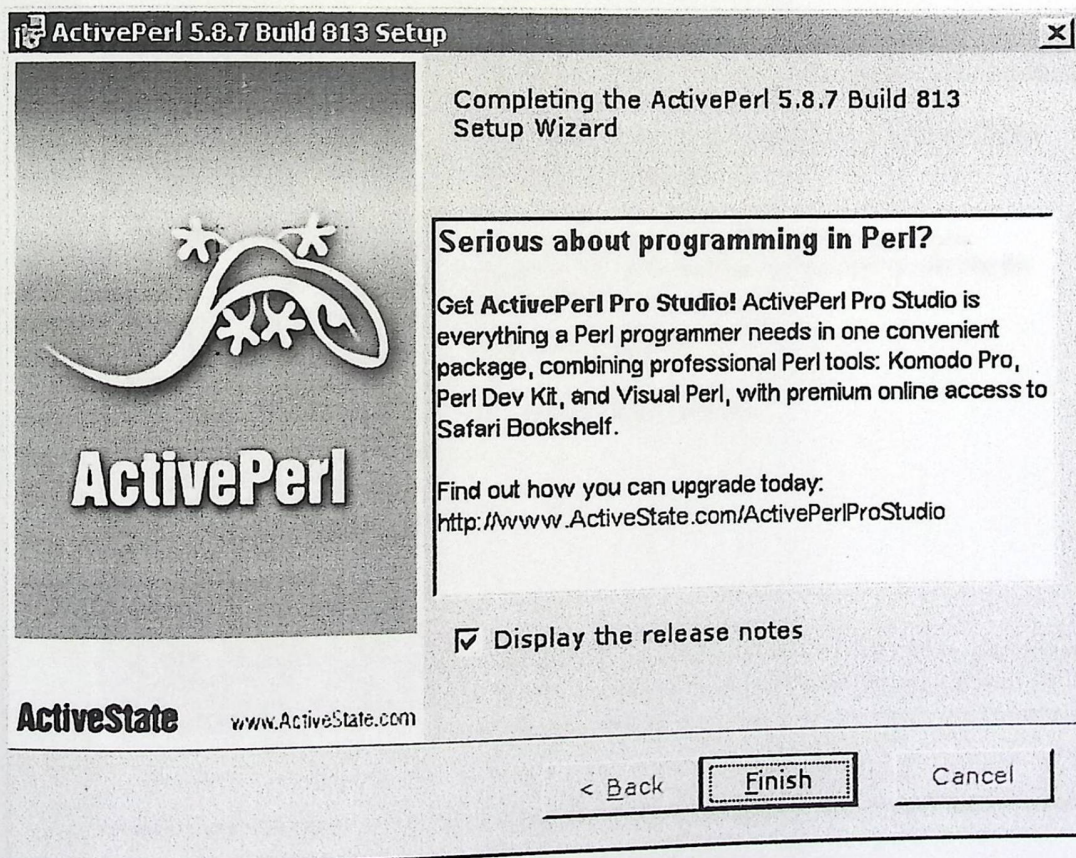


Figure [5.1] Install Active Perl 5.8.7.813.

- **Install DzSoft Perl Editor V5.6.0.2.**

DzSoft Perl Editor is a tool for writing, editing, and debugging Perl CGI scripts. It has a comfortable and intuitive interface both for beginners and advanced programmers. DzSoft Perl Editor is deceptively simple, but it is really a very powerful tool.

To install DzSoft Perl Editor V5.6.0.2:

- Run Perl Editor V5.6.0.2.exe from CD Rom.
- After determining the location of Perl Editor V5.6.0.2 and installing it the following figure will appear.

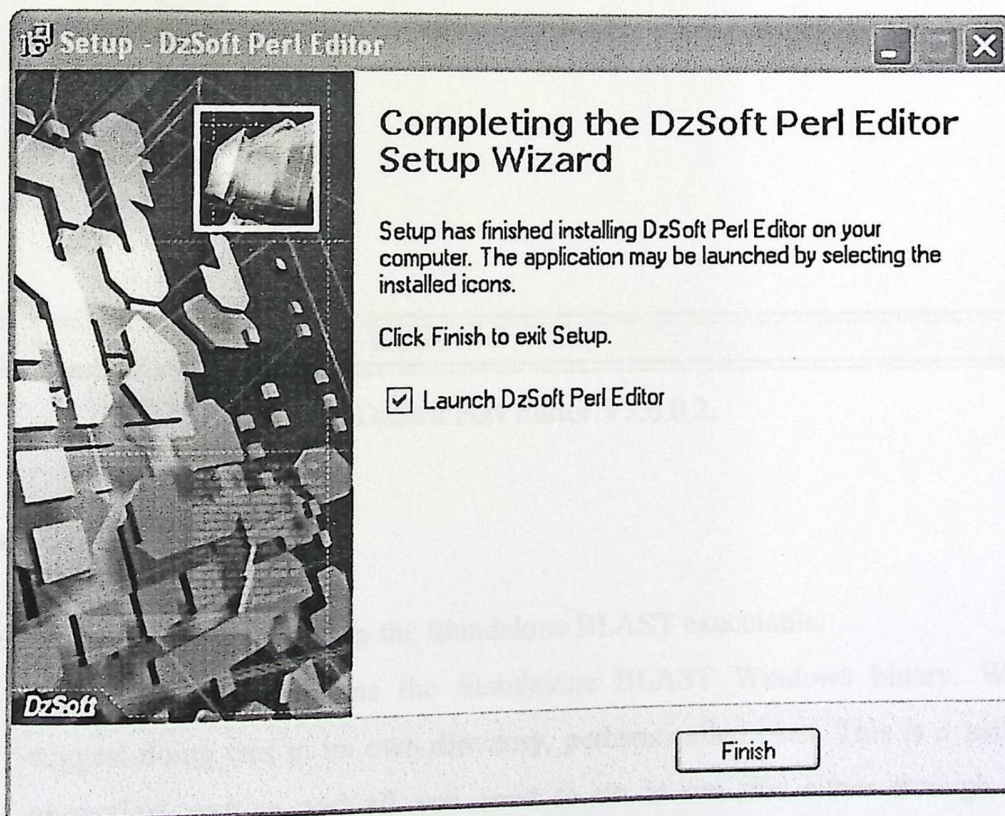


Figure [5.2] Finish install DzSoft Perl Editor V5.6.0.2.

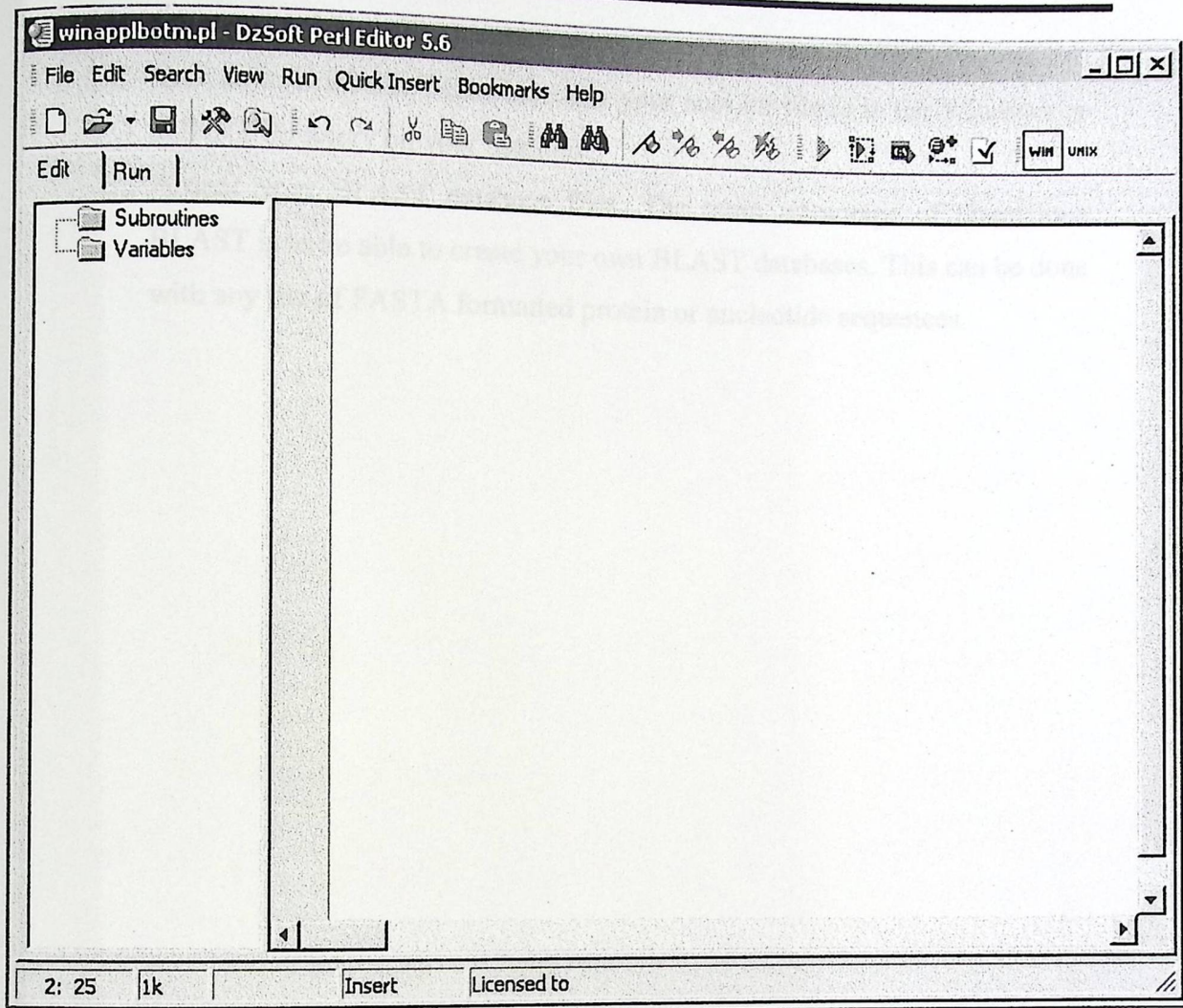


Figure [5.3] DzSoft Perl Editor V5.6.0.2.

- **Install BLAST all Program**

There are three steps needed to setup the Standalone BLAST executable:

1. Download and compress the Standalone BLAST Windows binary. We suggest doing this in its own directory, perhaps called blast. This is a 'self-extracting' archive and all you need to do is run this either through a Command Prompt (DOS Prompt) or by selecting "Run" from the Windows "Start button" and browsing the blastcz.exe file.

2. Create an ncbi.ini file. In order to operate Standalone BLAST, you need to have an ncbi.ini file. Make sure that your ncbi.ini file is in the Windows or WINNT directory on your machine.
3. Format your BLAST database files. The main advantage of Standalone BLAST is to be able to create your own BLAST databases. This can be done with any file of FASTA formatted protein or nucleotide sequences.

CHAPTER SIX

6

System Testing

Chapter Six

- ▶ *Introduction.*
- ▶ *Unit and Module Testing*
- ▶ *Integration Testing*
- ▶ *System Testing*
- ▶ *Acceptance Testing*
- ▶ *Snapshots*

6.1 Introduction

Testing the system to ensure that it meets its specifications is one of the most important stages in the software system development.

For the purpose of delivering a system that works properly as expected, certain testing procedures should be performed on system and its components; accordingly with an acceptance testing that may be stated as a result for the success of the testing process.

This chapter covers the testing for:

- System units and module testing.
- Integration testing.
- System testing.
- Acceptance testing.

Testing will take place in a time space that was assigned for the testing process.

Table (6.1) shows the testing schedule:

Time in week	1st week	2 nd week	3 rd week
Testing process			
Unit and module testing			
system testing.			
Integration testing.			
Acceptance testing.			

Table (6.1) Testing Schedule.

6.2 Unit and module testing

All system units and modules were tested against its specifications using the black box testing, the test ensured that the units and modules performed as expected.

The following are some samples for module testing and its associated results using whit box testing method.

- **Tested Function:** "Information gene ":

Method: path testing.

Test cases: each test case covers the set of input values in a certain execution path as shown in the function flowchart figure [6.1].

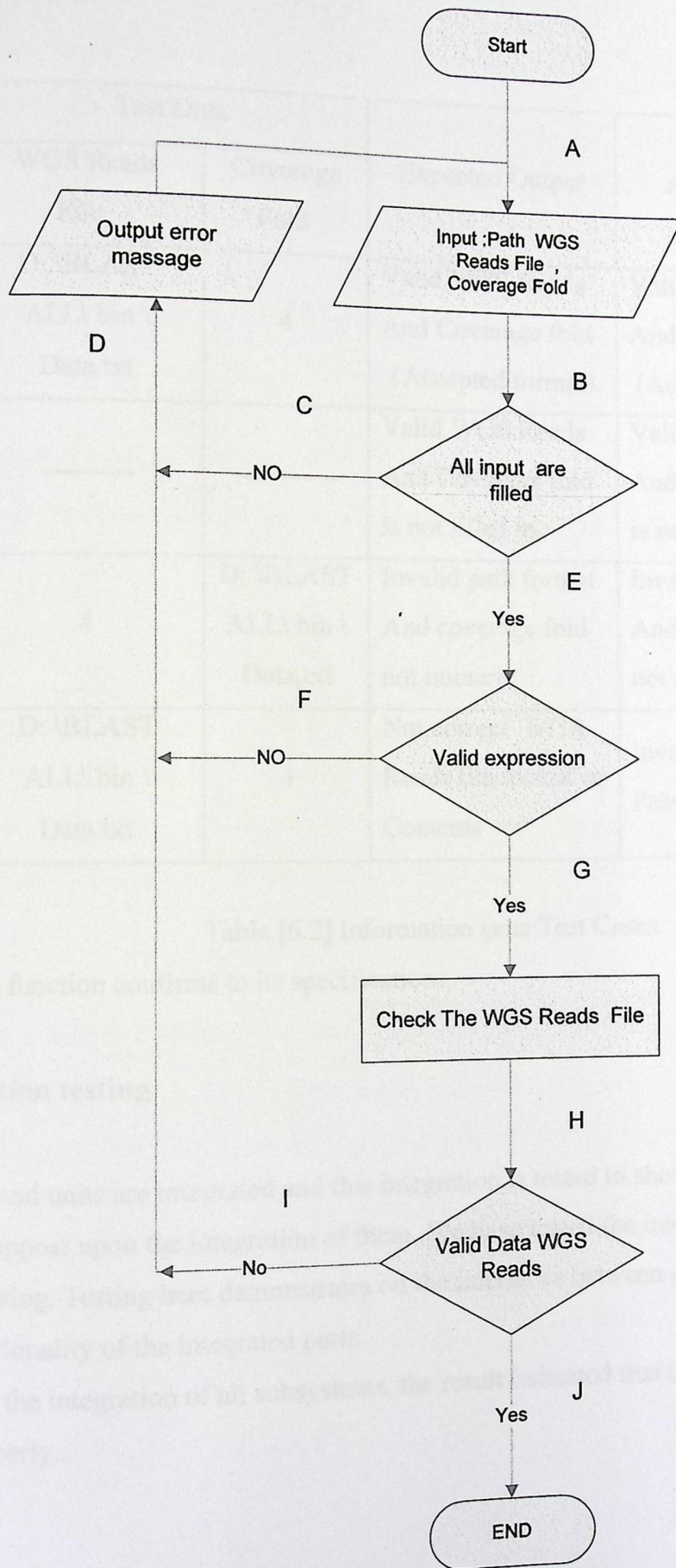


Figure [6.1] Information gene Execution Paths flowchar.

Test Cases Path	Test Data		Expected Output	Actual Output
	WGS Reads File	Coverage Fold		
A-B-E-G- H-J	D: \BLAST ALL\ bin \ Data.txt	4	Valid WGS Reads And Coverage fold {Accepted format}.	Valid WGS Reads And Coverage fold (Accepted format).
A-B-C-D	————	————	Valid WGS Reads And Coverage fold is not filled in.	Valid WGS Reads And Coverage fold is not filled in.
A-B-E-F-D	4	D: \BLAST ALL\ bin \ Data.txt	Invalid path format And coverage fold not numeric	Invalid path format And coverage fold not numeric
A-B-E-G- H-I-D	D: \BLAST ALL\ bin \ Data.txt	4	Not correct WGR Reads file format or Contents	Invalid login ID or Password.

Table [6.2] Information gene Test Cases.

Conclusion: function confirms to its specifications.

6.3 Integration testing

All module, and units are integrated and this integration is tested to show if there were defects that appear upon the integration of them. We have tested the integration using top-down testing. Testing here demonstrates on the interfaces between all modules, and the functionality of the integrated parts.

After testing the integration of all subsystems, the result indicated that they work together properly.

6.4 System Testing

The system was tested under several conditions, some errors were detected, and upon these results, we have solved these problems and we imposed the system another time to testing techniques to ensure that it disposed all types of defects and problems. Real data are filled through that sequence and traced to find that results are as expected so that the system processes work properly. System testing is also accomplished through transferring the system on different platforms.

6.5 Acceptance Testing

The system was tested against its requirements, we conclude that it achieves its functional requirements, and could operate soon in the real environment.

6.6 Sample Snapshots

We have selected some program snapshots to be displayed here to show how the real program behaves when working under certain situations and these snapshots are describe the main functions of our web based system as shown bellow:

6.6.1 Segmental Duplication Finder

This interface allows user to enter the necessary information to search and to find the duplication on gene file. The figure [4.1] shows two text boxes. In the first text box the user will enter the name and location of the WGS reads file. The second text box the user will enter the coverage fold. After click on the (Analyzing) button, the program will check the existing and validity of the WGS reads file in first text box, Check the validity of coverage fold in the second one.

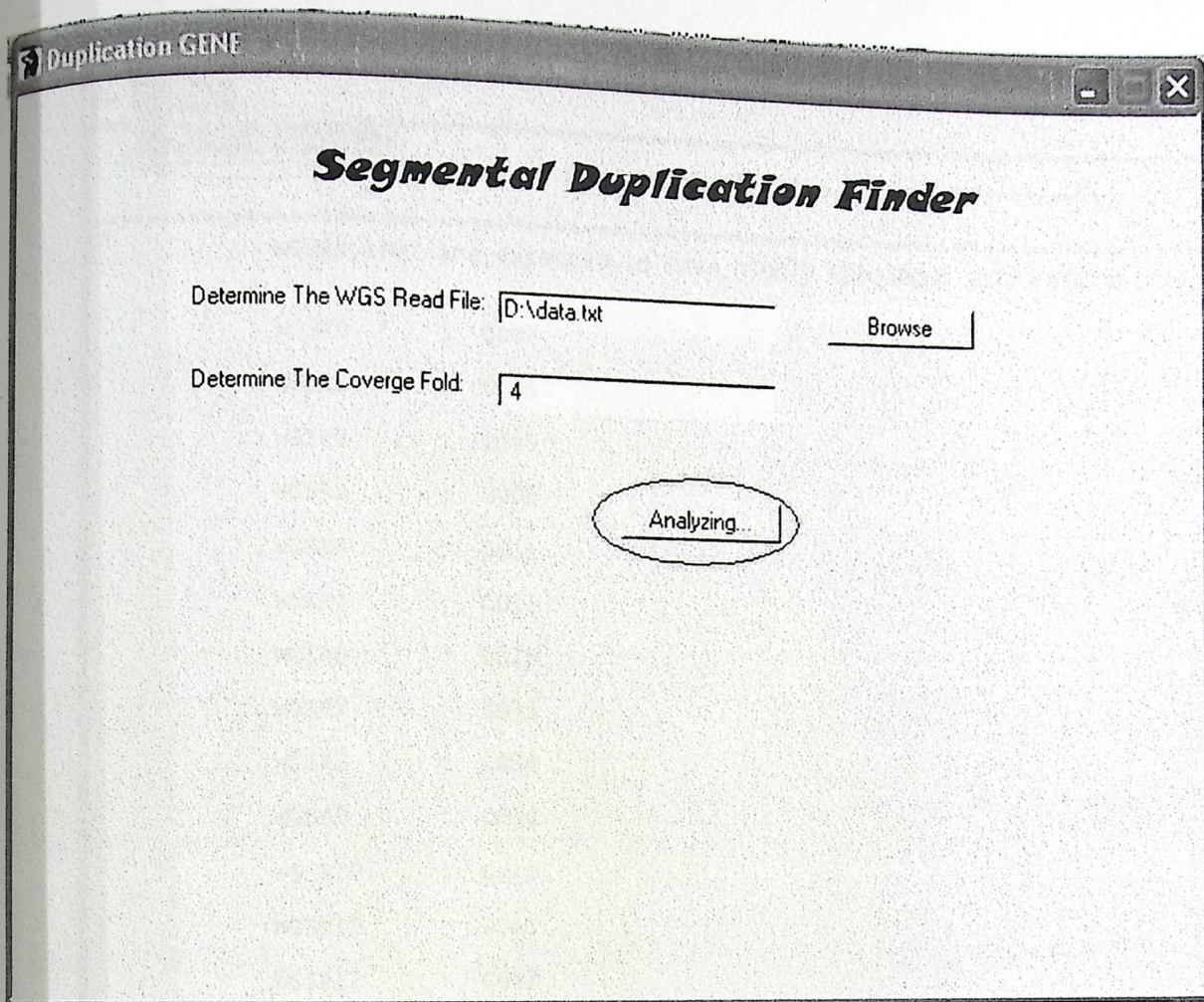


Figure [6.2] Segmental Duplication Finder Snapshot

6.6.2 Gene Duplication Result

After clicking on Analyzing button the program will give the result of duplication gene as the next figure.

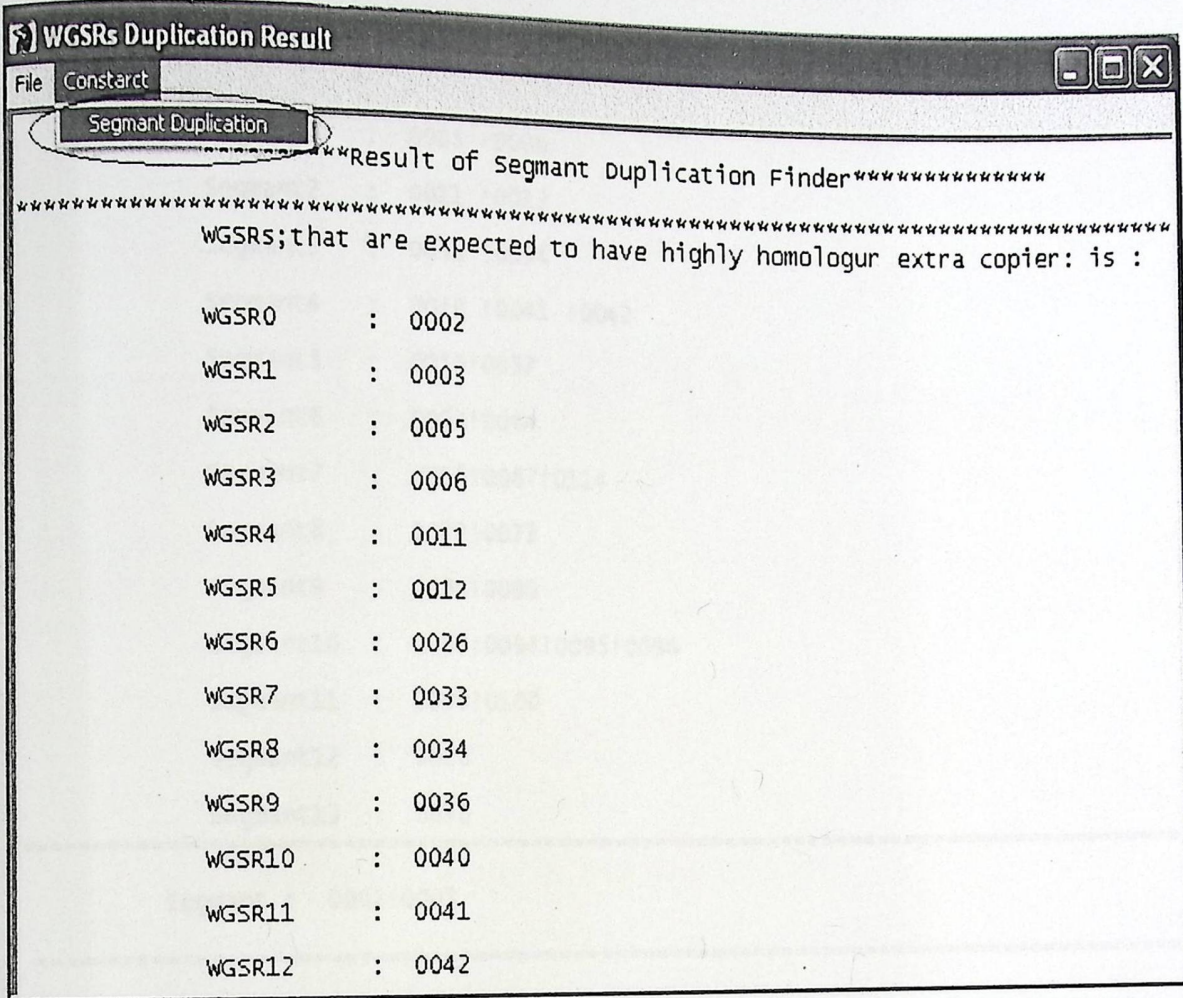
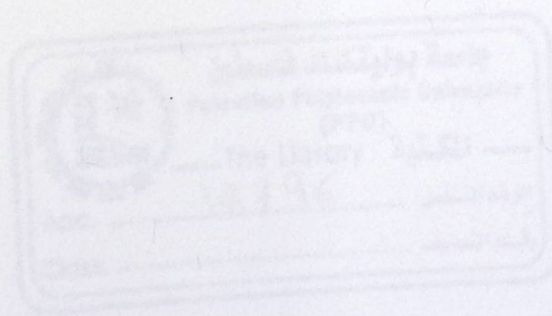


Figure [6.3] Gene Duplication Result Snapshot

6.6.3 Segmental Gene Duplication Result

After clicking on Segment Duplication button the program will give the result of segmental duplication as the next figure.



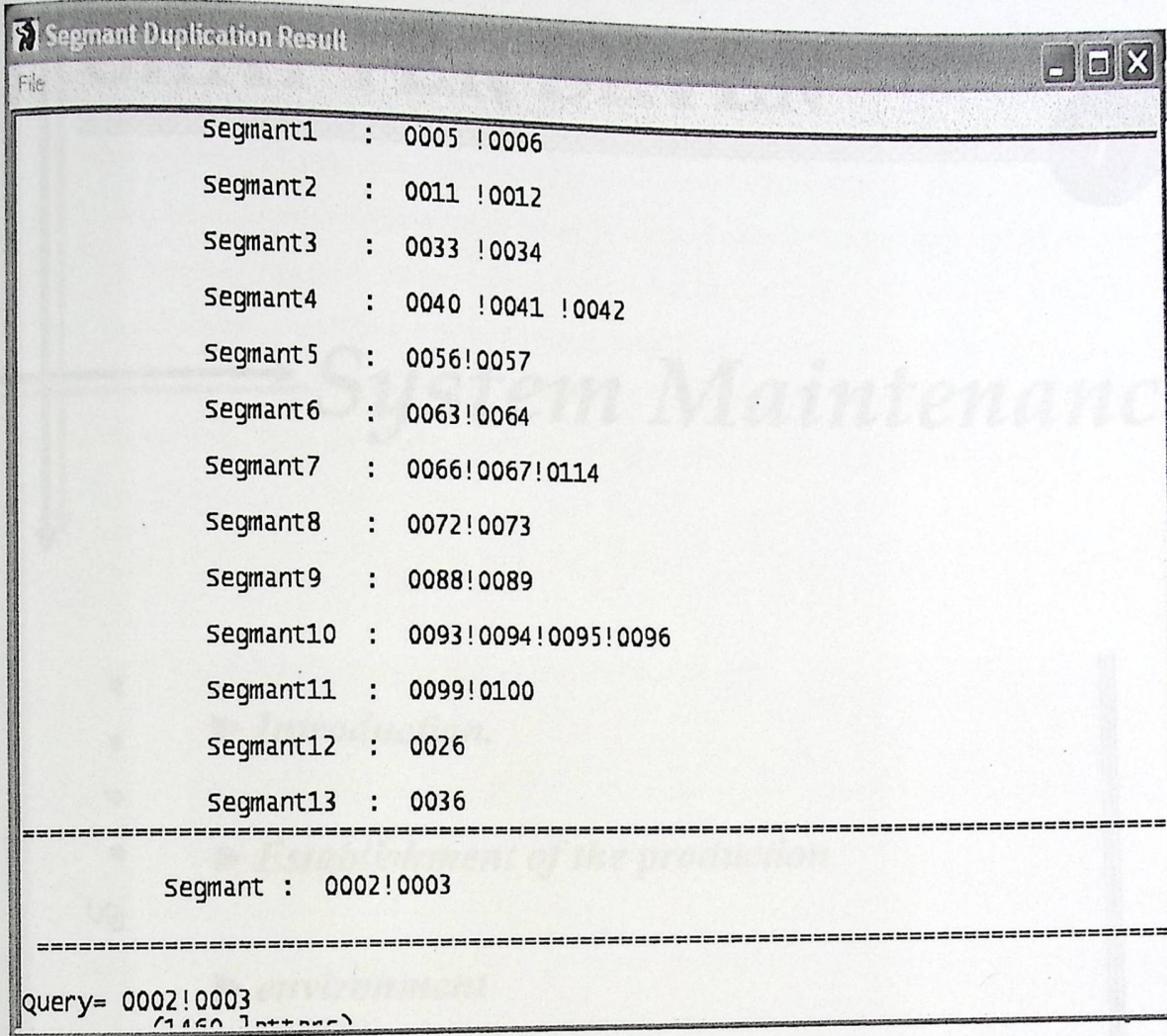
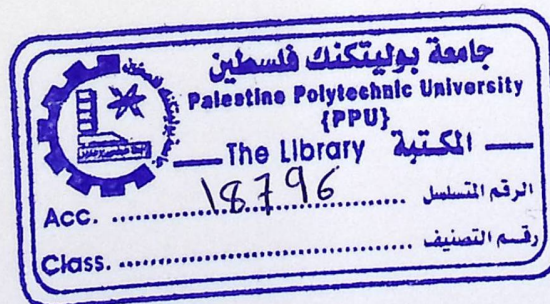


Figure [6.3] Segmental Duplication Result Snapshot



CHAPTER SEVEN

7

System Maintenance

Chapter Seven

▶ *Introduction.*

▶ *Establishment of the production*

▶ *environment*

▶ *Migration and Deployment Plan*

▶ *Maintenance Plan*

7.1 Introduction

In this chapter we will talk about the working environment. It is silly to think that the user is the developer of the system because he is just a user who lacks the knowledge of this system in that case, we have to provide the user with the sufficient information about the system, how it works, and how it could be maintained.

This chapter describes how to start working with the system; the establishment of the environment that the system will work in, what is the process of deployment, and the maintenance plan.

7.2 Establishment of the production environment:

To make it easily for the user to install this system certain environment must be provided.

In order to operate system on a PC with an operating system is required:

- Install the operating system.
- Install the Gene Duplication Finder program package.

The following steps are needed to install the program:

1. Inside GD Finder folder, Click on Setup
2. Choose where to install.

You must install it On C:\Gene Duplication\
Or D:\Gene Duplication\
Or any drive directly.

7.3 Migration and Deployment Plan

The deployment of the system must be built by certain steps so that to work properly within its environment; the production environment has to be established, configured, and a decision of operating on the new system must be taken considering all constrains and risks of the process of migration to the new system. Toward deploying and migrating to the new system we describe here the steps that must be done:

There are three steps required to move any windows application from the development environment to a production directory:

1. Build, or compile the Windows application, this compilation creates a .EXE file in the directory that contains the code for Gene Duplication.
2. Copy the necessary windows application files(after compilation Gene Duplication) in the development directory to the production directory, which are:
 - The .PL files.
 - Megablast.exe and Formatdb.exe
 - The Execution File
3. Package and build the production solution and run.

7.4 Maintenance Plan

When running system failures or errors may occur. In this case the programmer can follow the code and maintain it, a single error or multiple errors may occur will be solved. If an error handling occurs during the implementation, the error message and a description of that error will be displayed on the screen, then the customer must call the vendor and tell him about the error.

When the programmer solves the error, he should make unit testing and integration testing to ensure that the last qualifications will not influence the whole system performance. Finally, the corrected components must be developed and published.

The program purchaser can call the programmer using a special form , which contains the error information and description, and then the programmer must record each step in another form.

The program purchaser can call the programmer using a special form (Fig 6.1), which contains the error information and description, and then the programmer must record each step in another form (Fig 7.1).

Software Change Request Form

Software Change Request (SCR)

Requirement # _____
Date: _____

Type

- New Requirement System Problem Suggestion for Improvement
- Requirement Change User Interface Problem other: _____
- Design Change Documentation Correction

Description:

Please attach supporting documentation for the requested change
(Screen/report printouts, document pages affected, etc.)

Figure [7.1] Software Change Request Form

References

1. <http://www.perl.com/pub/q/documentation>.
2. <http://perl-win32-gui.sourceforge.net/cgi-bin/docs.cgi>.
3. <http://www.NCBI.nlm.nih.gov/blast>.
4. Jambeck, Per and Gibas, Cynthia. **Developing Bioinformatics Computer Skills**. 1ST Edition O'Reilly & Associates.2001.
5. Lesk, Arthur. **Introduction to Bioinformatics**. Oxford University Press. 1ST Edition 2002.
6. Eichler EE. **Recent duplication, domain accretion and the dynamic mutation of the human genome**. Trends Genet. 2001 Nov;17(11):661-9.
7. Bailey JA, Eichler EE. **Genome-wide detection and analysis of recent segmental duplications within mammalian organisms**. Cold Spring Harb Symp Quant Biol. 2003;68:115-24.
8. Bailey JA, Yavor AM, Massa HF, Trask BJ, Eichler EE. **Segmental duplications: organization and impact within the current human genome project assembly**. Genome Res. 2001 Jun;11(6):1005-17.
9. Bailey JA, Gu Z, Clark RA, Reinert K, Samonte RV, Schwartz S, Adams MD, Myers EW, Li PW, Eichler EE. **Recent segmental duplications in the human genome**. Science. 2002 Aug 9;297(5583):1003-7.
10. Brown, T. A. **Genomes**. BIOS Scientific publisher. Oxford, UK 2nd Edition 2002.
11. Sommerville, Ian, **Software Engineering**, 7th edition, Addison-Wesley, 2001.
12. Lesk, Arthur. **Introduction to Bioinformatics**. Oxford University Press. 1ST Edition 2002.
13. Jambeck, Per and Gibas, Cynthia. **Developing Bioinformatics Computer Skills**. First edition O'Reilly & Associates.2001. United state of America.
14. Tisdall, James. **Beginning Perl for Bioinformatics**. First edition O'Reilly & Associates.2001. United state of America.

Appendix

```

#!/usr/bin/perl
$coverge = ();
@Data_gene = ();
@Data_gene_segmant = ();
@final = ();
$item = ();
@rep = ();
$h = 0;
$f = (); $l = 8;
use Win32::GUI;      ## engage the Win32::gui module
use win32;          ## engage the win32 module
my ($DOS) = Win32::GUI::GetPerlWindow(); Win32::GUI::Hide($DOS);

```

```

=====
##Create the Segmental Duplication Finder wiindows.

```

```

    $font = Win32::GUI::Font->new(
        -name => "Lucida Console",
        -size => 10, );

```

```

    $file_menu2 = new Win32::GUI::Menu
        (
            "File" => "file",
            ">Help" => "help" ,
            "> -" => 0,
            "> Exit" => "Exit" ,
        );

```

```

    $main = new Win32::GUI::Window
        (
            -name => "main" ,
            -text => "GENE Duplication Finder " ,
            -menu => $file_menu2,
            -sizable => 0,
            -controlparent => 1,
            -maximizebox => 0,
            -addexstyle => WS_EX_CLIENTEDGE | WS_EX_STATICEDGE,
            -top => 100,
            -left => 50,
            -width => 640,
            -height => 480,
        );

```

```

    $font_label2 = Win32::GUI::Font->new(
        -name => "Snap ITC",
        -size => 16,
        -italic => 1,
    );

```

```

    $label2 = $main->AddLabel
        (
            -name => "label2" ,
            -text => "Segmental Duplication Finder" ,
            -left => ($main->Width()-380)/2,
            -top => 30,
            -font => $font_label2,
            -width => 380,
            -height => 40,
        );

```

```
);

$label1 = $main->AddLabel
(
  -name=> "label1" ,
- text=> "Determine The WGS Read File:" ,
  -left=>100,
  -top=> 100,
  -width=>200,
  -height=>20,
);

$label3 = $main->AddLabel
(
  -name=> "label3" ,
- text=> "Determine The Coverage Fold:" ,
  -left=>100,
  -top=> 140,
  -width=>200,
  -height=>20,
);

$text1 = $main->AddTextfield
(
  -name=> "text1" ,
  -text=> " " ,
  -left=> 255,
  -top=> 100,
  -width=>150,
  -height=>20,
);

$text2 = $main->AddTextfield
(
  -name=> "text2" ,
  -text=> " " ,
  -number=> 1,
  -left=> 255,
  -top=> 140,
  -width=>150,
  -height=>20,
);

$button1 = $main->AddButton
(
  -name => "button1" ,
  -text => "Browse" ,
  -left =>430,
  -top =>100,
  -width => 80,
  -height => 20,
);

$button2 = $main->AddButton
(
  -name => "button2" ,
```

Appendix

```
-text => " Analyzing... " ,
-left => (($main->Width()-85)/2),
-top =>200,
-width => 85,
-height => 20,
);
```

```
#####
##Create the WGSRs Duplication Result windows.
```

```
$file_menu = new Win32::GUI::Menu
(
    "File" => "File" ,
    "> Save AS" => "Save" ,
    ">BACK" => "BACK",
    "> -" => 0,
    "> Exit" => "Exit" ,
    "Constarct"=>"make",
    ">Segmant Duplication"=>"Segmant" ,
);
$main1= new Win32::GUI::Window
(
    -name => "main1" ,
    -text => "WGSRs Duplication Result " ,
    -top => 50,
    -sizable => 0,
    -menu=>$file_menu,
    -controlparent => 1,
    -addestyle =>WS_EX_CLIENTEDGE | WS_EX_STATICEDGE,
    -left=> 50,
    -width => 670,
    -height =>480,
);
$rich = $main1->AddRichEdit
(
    -name => "rich" ,
    -text=> " " ,
    -left =>0,
    -multiline => 1,
    -top =>0,
    -vscroll =>1,
    -hscroll => 1,
    -readonly => 1,
    -font=>$font,
    -width => $main1->Width()-10,
    -height =>$main1->Height()-55,
);
```

```
#####
##Create the WGSRs Duplication Result windows.
```

```
$file_menu1 = new Win32::GUI::Menu
(
    "File" => "File" ,
    "> Save AS" => "Save1" ,
    ">BACK" => "BACK1" ,
    "> -" => 0,
    "> Exit" => "Exit" ,
```


Appendix

```
-text => " Analyzing... " ,  
-left => (($main->Width()-85)/2),  
-top =>200,  
-width => 85,  
-height => 20,  
);
```

```
#####  
##Create the WGSRs Duplication Result windows.
```

```
$file_menu = new Win32::GUI::Menu  
(  
    "File" => "File" ,  
    "> Save AS" => "Save" ,  
    ">BACK" => "BACK",  
    "> -" => 0,  
    "> Exit" => "Exit" ,  
    "Constarct"=>"make",  
    ">Segmant Duplication"=>"Segmant" ,  
);  
$main1= new Win32::GUI::Window  
(  
    -name => "main1" ,  
    -text => "WGSRs Duplication Result " ,  
    -top => 50,  
    -sizable => 0,  
    -menu=>$file_menu,  
    -controlparent => 1,  
    -addestyle =>WS_EX_CLIENTEDGE | WS_EX_STATICEDGE,  
    -left=> 50,  
    -width => 670,  
    -height =>480,  
);  
$rich = $main1->AddRichEdit  
(  
    -name => "rich" ,  
    -text=> " " ,  
    -left =>0,  
    -multiline => 1,  
    -top =>0,  
    -vscroll =>1,  
    -hscroll => 1,  
    -readonly => 1,  
    -font=>$font,  
    -width => $main1->Width()-10,  
    -height =>$main1->Height()-55,  
);
```

```
#####  
##Create the WGSRs Duplication Result windows.
```

```
$file_menu1 = new Win32::GUI::Menu  
(  
    "File" => "File" ,  
    "> Save AS" => "Save1" ,  
    ">BACK" => "BACK1" ,  
    "> -" => 0,  
    "> Exit" => "Exit" ,
```

```
);
$main2= new Win32::GUI::Window
(
    -name => "main2" ,
    -text => "Segment Duplication Result " ,
    -top => 50,
    -sizable => 0,
    -menu=>$file_menu1,
    -controlparent => 1,
    -addexstyle =>WS_EX_CLIENTEDGE | WS_EX_STATICEDGE,
    -left=> 50,
    -width => 670,
    -height =>480,
);
$rich1 = $main2->AddRichEdit
(
    -name => "rich1" ,
    -text=> " " ,
    -left =>0,
    -multiline => 1,
    -top =>0,
    -vscroll =>1,
    -hscroll => 1,
    -readonly => 1,
    -font=>$font,
    -width => $main2->Width()-10,
    -height =>$main2->Height()-55,
);
```

```
=====
##Show main window
$main->Show();

##Message Loop
Win32::GUI::Dialog();
```

```
=====
sub button1_Click
{
    $folder = Win32::GUI::BrowseForFolder(
        -root => (0x0011),
        -owner=> $main,
        -editbox => 1,
        -includefiles => 1,
    );

    $text1->Text($folder);
    # $main->Hide();
    # $main1->Show();
}
```

```
);
$main2= new Win32::GUI::Window
(
    -name => "main2" ,
    -text => "Segment Duplication Result " ,
    -top => 50,
    -sizable => 0,
    -menu=>$file_menu1,
    -controlparent => 1,
    -addexstyle =>WS_EX_CLIENTEDGE | WS_EX_STATICEDGE,
    -left=> 50,
    -width => 670,
    -height =>480,
);
$rich1 = $main2->AddRichEdit
(
    -name => "rich1" ,
    -text=> " " ,
    -left =>0,
    -multiline => 1,
    -top =>0,
    -vscroll =>1,
    -hscroll => 1,
    -readonly => 1,
    -font=>$font,
    -width => $main2->Width()-10,
    -height =>$main2->Height()-55,
);
```

```
#####
##Show main window
$main->Show();

##Message Loop
Win32::GUI::Dialog();
```

```
#####

sub button1_Click
{
    $folder = Win32::GUI::BrowseForFolder(
        -root => (0x0011),
        -owner=> $main,
        -editbox => 1,
        -includefiles => 1,
    );

    $text1->Text($folder);
    # $main->Hide();
    # $main1->Show();
}
```

}

```
#####  
#-----  
sub Save_Click  
{ $File_save = Win32::GUI::GetSaveFileName  
  (-owner=> $main1,  
  -filter => (*.TXT;*.DOC;*.BAK),  
  -createprompt => 1,  
  -defaulttextextension => "txt",  
  -includefiles => 1,  
  );  
  $rich->Save("$File_save",0x0004);  
}
```

```
#####  
#-----  
sub Savel_Click  
{ $File_save = Win32::GUI::GetSaveFileName  
  (-owner=> $main2,  
  -filter => (*.TXT;*.DOC;*.BAK),  
  -createprompt => 1,  
  -defaulttextextension => "txt",  
  -includefiles => 1,  
  );  
  $rich1->Save("$File_save",0x0004);  
}
```

```
#####  
#-----  
sub Segmant_Click  
{  
  &progress_Bar1();  
  &Collection_WGSR($item);  
}
```

```
#####  
#-----  
sub main1_Maximize  
{  
  $main1->Maximize();  
  $rich->Maximize();  
}
```

```
#####  
#-----  
sub main2_Maximize  
{  
  $main2->Maximize();  
  $rich1->Maximize();  
}
```

```
    }  
    sub BACK1_Click  
    {  
    &main2_Terminate;  
    }  
=====
```

```
    sub BACK_Click  
    {  
    &main1_Terminate;  
    }  
    sub button2_Click  
    {  
    if(($text2->Text() eq 0)|($text1->Text()eq " "))  
    {  
    &massge_box(" Please Fill the Textbox ...?");  
    }  
    else  
    { $tex = $text1->Text();  
    if (open (data,"$tex"))  
    { &progress_Bar();  
    $coverge = $text2->Text();  
    &Reads_WGSR();  
    }  
    # $main->Hide();  
    # $main1->Show();  
    }  
    else  
    {  
    &massge_box(" The File is not correct...?" ) ;  
    }  
    }  
    }  
=====
```

```
    sub Exit_Click  
    {  
    &main_Terminate;  
    }  
    sub main1_Terminate  
    {  
=====
```

```
    $main1->Hide();  
    $main->Show();  
    }  
=====
```

```
    sub main2_Terminate  
    {  
=====
```

```
    $main2->Hide();  
    $main1->Show();  
    }  
=====
```

```

=====
sub main_Terminate
{
=====
    exit;
    return -1;

}
sub help_Click
{
    & Help_box;
}
=====

```

```

=====
sub Help_box
{
=====
    $main->Disable();
    $help1= new Win32::GUI::Window
        ( -owner=>$main,
          -name => "help1" ,
          -sizable => 0,
          -text => "Help...??" ,
          -top => $main->Top()+50,
          -left=>$main->Left()+50,
          -parent => $main,
          -width => 500,
          -height =>400,
          -maximizeBox => 0,
          -systemMenu =>1
        );
    $button5 = $help1->AddButton
        (
          -name => "button5" ,
          -text => " <= Back <=" ,
          -left =>($help1->Width()-60)/2,
          -top =>210,
          -width => 60,
          -height => 20,
        );
    $help1->Show();

    $label5 = $help1->AddLabel
        (
          -name=> "label5" ,
          -text=> " Gene Duplication Finder" ,
          -left=> ($help1->Width - 150) /2,
          -top=> 15,
          -width=>150,
          -height=>20,
        );
}
=====

```

```

@help_text = ("Gene Duplication Finder (GD Finder) is a program
designed to analyze databases that ".
"contain high number of shotgun reads that are
produced by genome projects. Gene ".
"Duplication Finder can easily and accurately

```

Appendix

delineate the genomic segments that extra ".
"very similar copies somewhere else in the same
genome. The program rely a Blast-based ".
"approach to analyze the whole genome shotgun reads
in order to detect the shotgun reads " .
"that have similar extra copies. After detecting the
duplicated shotgun reads, GD Finder " .
"will construct and determine physical limits of the
duplicated segments. " .

"\n\nThe program was designed in the Information
Technology Department, Palestine " .
"Polytechnic University, Hebron, and Palestine. ");

```
$label6 = $help1->AddLabel  
( -foreground => , [150, 29, 255],  
  -name=> "label6" ,  
  -text=> @help_text ,  
  -left=>($help1->Width()-450)/2,  
  -top=> 50,  
  -width=>450,  
  -height=>150,  
  );
```

```
    }  
    sub help1_Terminate  
    {
```

```
#####  
        $help1->Hide();  
        $main->Enable();  
        $main->Show();
```

```
    }
```

```
    sub massge_box  
    {
```

```
#####  
        $main->Disable();  
        $massege1 = new Win32::GUI::Window  
        ( -owner=>$main,  
          -name => "massege" ,  
          -sizable => 0,  
          -text => "Error!!!!!" ,  
          -top => $main->Top()+50,  
          -left=>$main->Left()+50,  
          -parent => $main,  
          -width => 200,  
          -height =>100,  
          -maximizeBox => 0,  
          -systemMenu => 0  
        );
```

```
$button3 = $massege1->AddButton  
(  
  -name => "button3" ,  
  -text => " OK " ,  
  -left =>70,  
  -top =>40,  
  -width => 40,  
  -height => 20,
```

```
);
$massege1->Show();

$label4 = $massege1->AddLabel
(
    -name=> "label4" ,
    -text=> $_[0] ,
    -left=>30,
    -top=> 15,
    -width=>150,
    -height=>20,
);
}

=====
sub progress_Bar
{
    $main->Disable();
    $probarform= new Win32::GUI::Window
        ( -owner=> $main,
        -name => "probarform" ,
        -text =>" Process Analyzing" ,
        -sizable => 0,
        -top => 200,
        -left=>100,
        -parent =>$main,
        -width => 500,
        -height =>150,
        -maximizeBox => 0,
        -systemMenu => 0
        );
    $probar = $probarform->AddProgressBar
        (
            -name=>"probar",
            -left =>40,
            -top =>50,
            -width =>400,
            -height => 30,

        );
    $label5 = $probarform->AddLabel
        (
            -name=> "label1" ,
            -text=>"Please Wait...",
            -left=>50,
            -top=>20,
            -width=>100,
            -height=>20,
        );
    $probar->SetStep(1);
    $probar->SetBkColor([255,255,255]);
    $probar->SetPos(10);
    $probarform->Show();
}
}
```



```
#####  
#-----  
sub progress_Bar1  
{ $main1->Disable();  
$probarform1 = new Win32::GUI::Window  
( -owner=> $main,  
-name => "probarform1" ,  
-text =>" Process Analyzing" ,  
-sizable => 0,  
-top => 200,  
-left=>100,  
-parent =>$main,  
-width => 500,  
-height =>150,  
-maximizeBox => 0,  
-systemMenu => 0  
);  
$probar1 = $probarform1->AddProgressBar  
(  
-name=>"probar",  
-left =>40,  
-top =>50,  
-width =>400,  
-height => 30,  
  
);  
$label6 = $probarform1->AddLabel  
(  
-name=> "label1" ,  
-text=>"Please Wait...",  
-left=>50,  
-top=>20,  
-width=>100,  
-height=>20,  
);  
$probar1->SetStep(1);  
$probar1->SetBkColor([255,255,255]);  
$probar1->SetPos(10);  
$probarform1->Show();  
}
```

```
#####  
#-----  
sub button3_Click  
{  
$massegel->Hide();  
$main->Enable();  
$main->SetActiveWindow();  
}
```

```
#####  
#-----  
sub button5_Click  
{  
$help1->Hide();  
$main->Enable();  
$main->SetActiveWindow();  
}
```

```
#####  
#-----  
sub Reads_WGSR
```

```

{   local($NUMWGSR);

$NUMWGSR = &Run_MegaBlast($tex);
for ( $f=1 ; $f< $NUMWGSR;$f++)
{
    $counter = 0;
    $expect = 0;
    $query=();
    $subjct=();
    @Data_gene=();
    $len=0;
    $condion =0;
    $con=0;
    $i=0; $j=0; $t=0;  $l=8; $e=0;
    unless (open (data,"TEMP/file$f.out"))
        { print "count open the file!";}

    while ( <data> ) #start loop
        {
            chop($_);
            if ($condion) # length of the query
            { $len = substr($_,(index($_,"")+1) ,
            (rindex($_," ")-(index($_,"")+1)) );
              $condion=0;
            }
            if ($_ =~ /Query= /) # query name
            {
                ($delet , $Data_gene[$i][0]) = split(/Query= /,"$_");
                $Data_gene[$i][0]= ($Data_gene[$i][0]);
                $condion=1;
                next;
            }

if (($_ =~ /alignments:/) or $con ) # number of subject
    result
        { $con=1;
          $expect++;
        }

if ($_ =~ />/ ) # name of the subject
    { $counter++;
      if($con)
        { $e=0;
          ($delet, $Data_gene[$i][1]) =
split(/>/,"$_");
          chop($Data_gene[$i][1]);
          $con = 0;
          $expect -=4;
        }
      else
        { $e=0;
          &extract_similarity_no($query,$i);
          &extract_similarity_Subject($subjct,$i);
          $query=();
          $subjct=();
          $i++;
          $Data_gene[$i][0] = $Data_gene[0][0] ;
        }
    }
}

```



```

                                next;
                                }

                                } #end loop

                                if($j==$counter)
                                {
                                    #unlink(TEMP/file.out);
                                    $T = 0 ;
                                    }

                                close (data); # close file
                                    if($T)
                                {
                                    &extract_similarity_no($query,$i);
                                    &extract_similarity_Subject($subjct,$i);
                                    for($i1=0;$i1 <= $i ; $i1++)
                                        {
                                            for($j1=0;$j1 <= $i;$j1++)
                                                {
                                                    $Data_gene_segmant[$item][$j1]=
                                                    $Data_gene[$i1][$j1];

                                                }
                                            $item++;
                                        }

                                    &filter_file($counter) ;
                                }

                                $probar->StepIt();

                                }#end loop
                                &sort_array_Datasegment($item);
                                open (input," file12.out" );
                                open (output1,">WGSr_duplication.txt");
                                print output1 "\n \t\t *****Result of Segment
                                Duplication Finder***** \n\n";
                                print output1 "*" x 100;
                                print output1 "\n \t\t WGSRs;that are expected to have
                                highly homologur extra copier: is : \n\n";
                                $f1= $rich->Text();
                                $rich->Text("$f1\r\n \t\t The WGSr Dublication is : ");
                                for($i=0;$i<$h;$i++)
                                    {
                                        print output1 "\n\t\t WGSr$i\t: $rep[$i] \n";
                                        # $f1= $rich->Text();
                                        $rich->Text("$f1\r\n\t\t WGSr$i\t: $rep[$i] ");
                                    }

                                $probar->StepIt();
                                print output1 "*" x 100;
                                print output1 "\n\n";
                                while (<input>)
                                    {
                                        print output1 $_;
                                        # $f1= $rich->Text();
                                        $rich->Text("$f1\r\n $_ ");
                                    }

```

```

@RT = <output1>;
close(input);

close(output1);
unlink ( "file12.out");

$rich->Load("WGSR_duplication.txt", 0x0001);
$main->Enable();
$main->Hide();
$probar->SetPos($probar->GetMax());
$probarform->Hide();
$main1->Show();
} # *****End function
Reads_WGSR*****

```

```

=====
sub print_array
{
    local ($i,$j,$len);
    $len = $_[0];
    open(out,">>yyy.txt");
    for ($i=0;$i<=$len;$i++)
{ if ($i==0){ print out "\n Query      subject length  start
End Ident\n"; }
    for ($j=0;$j<=$len;$j++)
    {
        print out "    $Data_gene[$i][$j]";
    }
    print out "\n";
}

    close(out);
}

```

```

=====
sub extract_similarity_no
{
    local ( $string , $cut , $y=(), $i);
    $string = $_[0] ;
    $i = $_[1];
    $Data_gene[$i][3]= int($string);
    while (1)
    { $cut = chop($string);
      if($cut =~ / /)
        {last ;}
      $y=$y.$cut;
    }
    $Data_gene[$i][4]=int(reverse($y)) ;
} # extract_similarity_no

```

```

=====
sub extract_similarity_Subject
{
    local ( $string , $cut , $y=(), $i);

```

```

$string = $_[0] ;
$i = $_[1];
$Data_gene[$i][7]= int($string);
while (1)
{ $cut = chop($string);
  if($cut =~ / /)
    {last ;}
  $y=$y.$cut;
}
$Data_gene[$i][8]=int(reverse($y)) ;
} # extract_similarity_Subject

```

```

=====
sub sort_array
{ local($i,$j,$len,$temp,$l1);
  $len= $_[0] ;
  for ($i=0 ; $i<= $len ; $i++)
  {
    for($j=$i;$j<=$len; $j++)
    {
      if($Data_gene[$i][3]> $Data_gene[$j][3])
      {
        for ($l1=0 ;$l1<=$l;$l1++)
        {
          $temp= $Data_gene[$i][$l1] ;
          $Data_gene[$i][$l1]=$Data_gene[$j][$l1];
          $Data_gene[$j][$l1] = $temp;
        }
      }
    }
  }
}

```

```

=====
sub sort_array_Datasegment
{ local($i=0,$j=0,$len=0,$temp=0,$l1=0,$l=8);
  $len= $_[0] ;

  for ($i=0 ; $i<= $len ; $i++)
  {
    for($j=$i;$j<=$len; $j++)
    {
      if($Data_gene_segmant[$i][7] >
      $Data_gene_segmant[$j][7])
      { for ($l1=0 ;$l1<=$l+2;$l1++)
        {
          $temp= $Data_gene_segmant[$i][$l1] ;
          $Data_gene_segmant[$i][$l1]=$Data_gene_segmant[$j][$l1];
          $Data_gene_segmant[$j][$l1] = $temp;
        }
      }
    }
  }
}

```

```

        $probar->StepIt();
        open(out, ">>segmant1.txt");
        for ($i=0;$i<=$len;$i++)
    {# if ($i==0){ print out "\n Query      subject length
      start End Ident      -----\n"; }
        for ($j=0;$j<=$l;$j++)
        {
            print out "    $Data_gene_segmant[$i][$j]";
        }
        print out "\n";
    }

        close(out);
    }

```

```

=====
    sub filter_file
    {
        local($i,$j,$len1,$temp,$l1,$t=0,$c=0);
        $temp=0;
        $len1= $_[0] ;
        &sort_array($len1);
        for ($i=0 ; $i<= $len1 ; $i++)
        {
            if ($Data_gene[$i][6]==0)
            {
                $Data_gene[$i][6]=$c;
                $c++;
                if( $len-1 == abs($Data_gene[$i][3]-
                $Data_gene[$i][4]))
                {
                    next;
                    $temp++;
                } #end if
            }
            else
            {
                $l1=$Data_gene[$i][4];
                for($j=$i+1;$j<=$len1;$j++)
                {
                    if ($Data_gene[$j][6]==0)
                    {
                        if($Data_gene[$j][3] == $l1 +1)
                        {
                            $Data_gene[$j][6]=$i;
                            if( $len-1 ==
                            abs($Data_gene[$i][3]- $Data_gene[$j][4]))
                            {
                                $Data_gene[$j][6]=$i;
                                $temp++;
                                last ;
                            } # end if
                        }
                    }
                    else
                    {
                        $l1 = $Data_gene[$j][4];
                        $Data_gene[$j][6]=$i;
                    }
                }
            }
        }
    }

```

```

        }#end if
    }#end if
}# end loop
}

}#end if

}#end loop

    if($c <= $coverge-1)
{ #unlink(TEMP/file.out); #Delete file
    }
    else
{ &Display_WGSR_Duplication();
  $rep[$h]= $Data_gene[1][0];
  $h++;
  &print_array($counter);
  }

} # end procedure

```

```

=====
sub Run_MegaBlast
{
    local($location=());
    #engage the Cwd module
    use Cwd ;
    ##Get the current directory location
    $directory = cwd ;
    ##assign the variable $folder
    $folder = "$directory/TEMP" ;
    ##assign the attribute
    $attribute = 777;
    ## make a directory
    mkdir($folder,$attribute);
    $location = $_[0];
    system( "formatdb -i $location -p F -o T" );
    unless (open (DATABASE,"$location"))
    { print "count open the file!";}
    $i=1;
    open(file,">file.txt");
    while ( <DATABASE> )
    {
        chop($_);

        if ( $_ =~ />/ )
        {

            if($i!= 1 )
            { close(file);
              $j=$i-1;
            }
            system( "MegaBlast -d $location -i

```



```

file.txt -o temp/file$j.out -F \"m D;V;L \" \" );
                                my ($DOS) =
Win32::GUI::GetPerlWindow();Win32::GUI::Hide($DOS);
                                open(file,\">file.txt");
                                }
                                $i++;
                                }
                                print file "$_\n";
                                }
                                $j=$i-1;

                                system( "MegaBlast -d $location -i file.txt -o
temp/file$j.out -F \"m D;V;L \" \" );
                                my ($DOS) =
Win32::GUI::GetPerlWindow();Win32::GUI::Hide($DOS);
                                return $i;
                                }

=====
sub Display_WGSR_Duplication
{
    local ($i=0,$j,$len,$temp,$l1);

    unless (open (data,"TEMP/file$f.out"))
    { print "count open the file!"; }
    unless (open (input,">>file12.out"))
    { print "count open the file!"; }
    print input "=" x 100;
    print input "\n\n WGSR : $Data_gene[1][0] \n\n " ;
    print input "=" x 100;
    print input "\n\n";
    while ( <data> ) #start loop
    { chop($_);
      if($_ =~ /Query=/)
      {$i++; }
      if($_ =~ /Database:/)
      {$i=0; }
      if($i)
      { print input $_;
        print input "\n";
      }
    }

    close(data);
    print input "\n\n";

    print input "\n\n";
    close(input);
} # end of function Display

```



```

== $l1 +1)
if($Data_gene_segmant[$j][5]< 99)
$loc[$He].!"!$Data_gene_segmant[$j][0]";
$Data_gene_segmant[$j][8];
$Data_gene_segmant[$j][6]=$c;
{
  if($Data_gene_segmant[$j][7]
  {
    $loc[ $He ]=
    $l1 =
  }
}
} # end for 2

$He++;
} #end else
} # end if 2
} # end if 1
} # end for 1

$probar1->StepIt();
open(out,">>segmant1.txt");
for ($i=0;$i<=$len;$i++)
{ # if ($i==0){ print out "\n Query      subject length
start  End Ident      -----\n"; }
for ($j=0;$j<=$l;$j++)
{
  print out "      $Data_gene_segmant[$i][$j]";
}
print out "\n";
}

close(out);
$j=0;
for($i=0;$i<=$#loc;$i++)
{
  if($loc[$i]=~/!/)
  { $locf[$j][0]=$loc[$i];
    $locf[$j][1]=0;
    $j++;
  }
}
$probar1->StepIt();
@locd=(); $d=0;
for($i=0;$i<=$#locf;$i++)
{ if($locf[$i][1]==0)
{
  for($j=$i;$j<=$#locf;$j++)
  {
    if($j==$i) {$locd[$d][0]= $locf[$i][0]; }
    if($locf[$i][0] eq $locf[$j][0])
    { $locf[$j][1]=1;
      $locd[$d][1]++;
    }
  }
}
}

```

```
    }
  }
  $d++;
}
}
$probar1->StepIt();
for($i=0;$i<=#locd;$i++)
{
  @fi=();
  @fi = split(/!/, $locd[$i][0]);
  if($locd[$i][1]!=0)
  {
    for($j=$i+1;$j<=#locd;$j++)
    {
      @se=();
      @se = split(/!/, $locd[$j][0]);
      for($k=0;$k<=#fi;$k++)
      {
        if($fi[$k] eq $se[$k])
        {
          if($locd[$i][1]>$locd[$j][1])
          {
            $locd[$j][1]=0;
            last;
          }
          else
          {
            $locd[$i][1]=0;
            last;
          }
        }
      }
    }
  }
}
# *****

for ($i=0 ; $i<= $len ; $i++)
{
  for($j=$i;$j<=#len; $j++)
  {
    if($locd[$i][0] gt $locd[$j][0])
    {
      for ($l1=0 ;$l1<=1;$l1++)
      {
        $temp= $locd[$i][$l1] ;
        $locd[$i][$l1]=$locd[$j][$l1];
        $locd[$j][$l1] = $temp;
      }
    }
  }
}
$probar1->StepIt();
@final=();
$d=0;
for($i=0;$i<=#locd;$i++)
{
  @fi=();
  @fi = split(/!/, $locd[$i][0]);
```

```

if($locd[$i][1]!=0)
{
    $rt=$fi[$#fi];
    $final[$d]=$locd[$i][0];
    for($j=0;$j<=$#locd;$j++)
    {
        @se=();
        @se = split(/!/, $locd[$j][0]);
        if($locd[$j][1]!= 0)
        {
            if($rt =~ /$se[0]/)
            {
                $rt = $se[$#se];
                $final[$d]=$final[$d].!"$rt";
                $locd[$j][1]=0;
                $locd[$i][1]=0;
            }
        }
    }
    $d++;
}
} $probar1->StepIt();
for($i=0;$i<=$#final;$i++)
{
    print " $final[$i]" ;
    print "\n";
}

for ($i=0 ; $i<= $#rep ; $i++)
{
    $k=1;
    for($j=0;$j<=$#final; $j++)
    {
        @fi=();
        @fi = split(/!/, $final[$j]);
        for ($l1=0 ;$l1<=$#fi;$l1++)
        {
            if ( $fi[$l1] eq $rep[$i] )
            {
                $k=0; last; }
        }
    }
    if($k)
    {
        push(@final, $rep[$i]); }
}
$probar1->StepIt();
&MakeFile_Segmant();

$probar1->StepIt();

} # END for procedure "Collection_WGSR "

```

```

=====
sub MakeFile_Segmant
{
    local ($j=0, $i=0, @fi=(), $k=0);

```

```
open(input, ">DATAS.txt");
for($i=0 ; $i<=$#final;$i++ )
{
    print input "\n >$final[$i]\n";
    $k=0;
    @fi=();
    @fi = split(/!/, $final[$i]);
    for($j=0;$j<=$#fi; $j++)
    {
        open (data, "$tex");
        while(<data>)
        { chop($_);

            if($k)
            {
                if($_ =~ />/)
                {$k=0; last;}
                print input "$_\n";
            }
            if($_ =~ />$fi[$j]/)
            { $k=1
            }

        }
        close(data);
    }

}

$probar1->StepIt();
close(input);
&Rur_megeblast_Segment();

}
```

```
#####

sub Rur_megeblast_Segment
{ local($i=0,$j=0,$k=0);
  #engage the Cwd module
  use Cwd ;
  ##Get the current directory location
  $directory = cwd ;
  ##assign the variable $folder
  $folder = "$directory/SEGMANT" ;
  ##assign the attribute
  $attribute = 777;
  ## make a directory
  mkdir($folder,$attribute);

  unless (open (DATABASE,"DATAS.txt"))
  { print "count open the file!";}
  $i=1;
  open(file,">file.txt");
  while ( <DATABASE> )
  {
    chop($_);
```

```

if ( $_ =~ />/ )
{
    if($i!= 1 )
    { close(file);
      $j=$i-1;
      system( "MegaBlast -d $tex -i
file.txt -o SEGMENT/file$j.out -F \"m D;V;L \" " );
      open(file,">file.txt");
    }
    $i++;
}
print file "$_\n";
}
$j=$i-1;
system( "MegaBlast -d $tex -i file.txt -o
SEGMENT/file$j.out -F \"m D;V;L \" " );

unless (open (input,">SegmentDuplication.txt"))
{print "count open the file!"; }
$probar1->StepIt();
print input "*" x 100;
print input "\n \t\t The WGSR Dublication is : \n\n";
for($i=0;$i<=$#final;$i++)
{
    print input "\n\t\t Segment$i\t: $final[$i] \n";
}
print output1 "*" x 100;
print output1 "\n\n";
$probar1->StepIt();
for($i=0;$i<=$#final;$i++)
{
    $k=$i+1;
    unless (open (data,"SEGMENT/file$k.out"))
    { print "count open the file!"; }
    print input "=" x 100;
    print input "\n\n      Segment : $final[$i] \n\n" ;
    print input "=" x 100;
    print input "\n\n";
    $j=0;
    while ( <data> ) #start loop
    { chop($_);
      if($_ =~ /Query=/)
      {$j++; }
      if($_ =~ /Database:/)
      {$j=0; }
      if($j)
      { print input $_;
        print input "\n";
      }
    }
}
close(data);

```

```
print input "\n\n";  
print input "\n\n";  
    }  
    close(input);  
$probar1->SetPos($probar1->GetMax());  
$rich1->Load("SegmantDuplication.txt",0x0001);  
$probarform1->Hide();  
$main1->Enable();  
$main1->Hide();  
$main2->Show();  
}  
#=====
```