CB-COMMANDER: A PLUGIN BASED SOFTWARE SUITE FOR HIGH THROUGHPUT SEQUENCING ALGORITHMS

by

Reza Shahidi-Nejad

B.Sc. in Software Engineering, University of Tehran, 2008

a Project submitted in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in the School of
Computing Science

© Reza Shahidi-Nejad 2011
SIMON FRASER UNIVERSITY
Summer 2011

All rights reserved. However, in accordance with the Copyright Act of Canada, this work may be reproduced without authorization under the conditions for Fair Dealing. Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Reza Shahidi-Nejad
Degree: Master of Science
Title of Project: CB-Commander: A Plugin Based Software Suite for High Throughput Sequencing Algorithms

Examining Committee: Mr. Greg Baker
Chair

Dr. Cenk Sahinalp, Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Ke Wang, Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Torsten Moller
Associate Professor, Computing Science
Simon Fraser University
Examiner

Date Approved: July 27, 2011
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011
Abstract

The field of genomics has gone through dramatic changes in the past decade, and has increasingly become dependent on computational tools and algorithms to facilitate the researchers in their endeavor. With the advent of high throughput sequencing technologies in last few years, this field has been revolutionized and even more dependent on fast accurate algorithms. However, for a researcher, using new tools developed and learning their interfaces can be challenging by itself, and usually time consuming.

In this project, we design and implement a tool which is capable to act as a systematic interface for researchers to use tools developed for high throughput sequencing technologies with ease. This is a plug-in based tool which will give users the ability to add their algorithm to the system easily and will give them a user friendly interface for different tasks from creating the experiments to running it and visualizing the results.
To my parents, Maryam and Mahdi
Donkey: “Are we there yet?”, Shrek: “NOT YET!”

— Shrek 2, Dream Works, 2004
Acknowledgments

First, I offer my sincere appreciation to my senior supervisor, Cenk Sahinalp, for all of his support and encouragements. Without his supervision this project would have not been possible.

My grateful thanks to Nahid Karimaghalou, my beloved fiance, for being the best supporting friend who would help me whenever she could. Love you, for being there for me, no matter when or what.

My great appreciation and gratitude to my dear friend, Hasti Seifi for her friendship and for being the one whom I can always trust and rely on. Thanks a lot for helping me get through the difficult times and for all of your great advices.

My special thanks go to Fereydoun Hormozdiari for his wise ideas, helpful comments and guidelines through all stages of this project.

I would also like to express my sincere appreciation to my family for their love and support. Every milestone that I have passed is the result of your dedication and encouragements.

Last but not least, I would like to thank my friends at Simon Fraser university, specially Farhad Hormozdiari and Shabnam Mirshokraei who were my surrogate family in Canada and who made my student life more joyful.

Learned from Hasti, I will put this acknowledgment note on my homepage to be able to update it and add the forgotten names in the future.
Contents

Approval ii
Abstract iii
Dedication iv
Quotation v
Acknowledgments vi
Contents vii
List of Tables x
List of Figures xi
List of Programs xii
1 Introduction 1
  1.1 Genome ................................................................. 1
  1.2 High Throughput Sequencing ......................................... 1
  1.3 Genomic Challenges to be Addressed ................................. 3
    1.3.1 Mapping .......................................................... 4
    1.3.2 Structural Variation Discovery .................................... 4
    1.3.3 Gene Fusion Discovery ............................................. 5
    1.3.4 Novel Sequence Insertions Discovery ............................ 6
  1.4 The 1000 Genomes Project ............................................ 6
1.5 Motivation ................................................. 6
1.6 Some of Currently Available Software ....................... 7
1.7 Our Problem .............................................. 8

2 CB-Commander ............................................. 9
  2.1 About CB-Commander ................................... 9
  2.2 Use Cases .............................................. 10
    2.2.1 Install New Plugin ................................. 10
    2.2.2 Change Workspace ................................. 12
    2.2.3 Install Components on Server .................. 12
    2.2.4 Transfer File to Server ......................... 13
    2.2.5 Create Experiment ............................... 13
    2.2.6 Save/Load Experiment ......................... 15
    2.2.7 Validate Experiment ......................... 15
    2.2.8 Run Experiment ................................. 16
    2.2.9 Check Status of Experiment .................. 16
    2.2.10 Download Output Files ....................... 16
    2.2.11 Visualize Results ......................... 17
  2.3 Unique Features ..................................... 17
    2.3.1 Pluggability ................................... 18
    2.3.2 Automatic GUI Form Generation ............... 18
    2.3.3 Pipelining ................................... 19
    2.3.4 SSH and Remote ............................... 21
    2.3.5 User Friendly GUI ........................... 21
  2.4 Platform ............................................. 21

3 Using CB-Commander ....................................... 23
  3.1 Preliminaries ......................................... 23
    3.1.1 Java Bean ................................... 23
    3.1.2 Java Annotation ................................. 24
  3.2 How to Create Your Plugin ............................ 24
    3.2.1 How to Introduce the Parameters ............. 25
    3.2.2 How to Add Metadata to Parameters .......... 26
    3.2.3 What Properties Should the Algorithms Have .... 27
List of Tables

1.1 Human chromosomes ......................................................... 2
1.2 A comparison of next-generation sequencing platforms ............... 3
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Three structural variation - deletion, insertion and inversion</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>Gene fusion due to deletion</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Main form of CB-Commander - Logo by Bryan Ballinger: <a href="http://www.breadwig.com">www.breadwig.com</a></td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>High level use case diagram</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Switch workspace form</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Create pipeline form - Sample experiment designed in CB-Commander</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Structural variation visualizer</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>Sample automatically generated input form</td>
<td>19</td>
</tr>
<tr>
<td>2.7</td>
<td>Edit Connections</td>
<td>20</td>
</tr>
</tbody>
</table>
List of Programs

3.1 A simple Java bean with two member variables ......................... 24
3.2 A sample Java annotation for an integer field ......................... 24
3.3 A sample Java bean .................................................. 25
3.4 Sample configuration file for a component .......................... 30
3.5 Part of a CommanderBean class .................................... 31
3.6 A sample prepareExperiment implementation ....................... 32
Chapter 1

Introduction

1.1 Genome

A Genome is the complete set of hereditary information of an organism which is encoded in DNA; and DNA comes in the form of a double helix string consisting of four nucleic acids: Adenine (abbreviated A), Cytosine (C), Guanine (G) and Thymine (T). Genomes of all organisms are classified into different chromosomes. For example, the human genome has approximately three billion nucleic acid pairs that are arranged in 46 chromosomes of which 22 pairs are autosomal and the last pair are sex chromosomes. Table 1.1, extracted from the Vertebrate Genome Annotation (VEGA) database [2], compiles statistics on the different chromosomes of the human genome. The complete genome of a human was sequenced and published in 2003 and it is still being updated. This published genome is referred to as the reference genome and is used as a basis for solving many problems in the genomics fields.

1.2 High Throughput Sequencing

DNA sequencing is the process of determining the order of the nucleotide bases (A, C, T and G) of a specific DNA. DNA sequencers can not determine the order of all of the base pairs of the genome in one pass. However, they are capable of reading small parts at a time. Each of these small parts is called a read. The size of a read depends on the sequencing technology. Unfortunately, the starting point of a read is not known and thus, we can not easily assemble them together to form the complete genome sequence and therefore, we can not determine if all of the genome is yet sequenced or not. On the other hand, the reads may
The first DNA sequences were obtained and published in the early 1970s. Since then, different technologies have been employed to sequence the DNA with high coverage while carrying low cost. In the last decade, with the advent of high throughput sequencing methods, the field of genomics has been revolutionized. Especially in the last few years, the cost of genome sequencing has been reduced by a factor of ten and today we can decode a person’s DNA in one week with less than $10,000 worth of materials [12]. Having

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Genes</th>
<th>Total bases</th>
<th>Sequenced bases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,240</td>
<td>249,240,621</td>
<td>225,280,621</td>
</tr>
<tr>
<td>2</td>
<td>3,076</td>
<td>243,189,373</td>
<td>238,207,373</td>
</tr>
<tr>
<td>3</td>
<td>2,330</td>
<td>197,962,430</td>
<td>194,797,140</td>
</tr>
<tr>
<td>4</td>
<td>2,009</td>
<td>191,044,276</td>
<td>187,661,676</td>
</tr>
<tr>
<td>5</td>
<td>2,283</td>
<td>180,905,260</td>
<td>177,695,260</td>
</tr>
<tr>
<td>6</td>
<td>2,284</td>
<td>171,055,067</td>
<td>167,395,067</td>
</tr>
<tr>
<td>7</td>
<td>2,230</td>
<td>159,128,663</td>
<td>155,353,663</td>
</tr>
<tr>
<td>8</td>
<td>1,872</td>
<td>146,304,022</td>
<td>142,888,922</td>
</tr>
<tr>
<td>9</td>
<td>1,939</td>
<td>141,153,431</td>
<td>120,143,431</td>
</tr>
<tr>
<td>10</td>
<td>1,780</td>
<td>135,524,747</td>
<td>131,314,747</td>
</tr>
<tr>
<td>11</td>
<td>2,754</td>
<td>134,946,516</td>
<td>131,129,516</td>
</tr>
<tr>
<td>12</td>
<td>2,366</td>
<td>133,841,895</td>
<td>130,481,895</td>
</tr>
<tr>
<td>13</td>
<td>925</td>
<td>115,109,878</td>
<td>95,589,878</td>
</tr>
<tr>
<td>14</td>
<td>776</td>
<td>107,289,540</td>
<td>88,289,540</td>
</tr>
<tr>
<td>15</td>
<td>516</td>
<td>102,521,392</td>
<td>81,694,769</td>
</tr>
<tr>
<td>16</td>
<td>763</td>
<td>90,294,753</td>
<td>78,884,753</td>
</tr>
<tr>
<td>17</td>
<td>1,086</td>
<td>81,195,210</td>
<td>77,795,210</td>
</tr>
<tr>
<td>18</td>
<td>243</td>
<td>78,017,248</td>
<td>74,657,248</td>
</tr>
<tr>
<td>19</td>
<td>576</td>
<td>59,118,983</td>
<td>55,808,983</td>
</tr>
<tr>
<td>20</td>
<td>1,023</td>
<td>62,965,520</td>
<td>59,505,520</td>
</tr>
<tr>
<td>21</td>
<td>596</td>
<td>48,119,895</td>
<td>35,108,702</td>
</tr>
<tr>
<td>22</td>
<td>1,014</td>
<td>51,244,566</td>
<td>34,894,566</td>
</tr>
<tr>
<td>X (sex chromosome)</td>
<td>1,860</td>
<td>155,260,560</td>
<td>151,100,560</td>
</tr>
<tr>
<td>Y (sex chromosome)</td>
<td>454</td>
<td>59,363,566</td>
<td>25,653,566</td>
</tr>
</tbody>
</table>

not be accurate and may contain errors. To make sure that all of the genome is sequenced with reasonable accuracy, sequencers ensure to read each part of the genome at least ‘k’ times. This ‘k’ is called the Depth of Coverage of a DNA sequence. The higher the coverage, the more accurate the sequence is and the easier the task of assembling the whole genome becomes.
sequences with high coverage obtained by low cost methods, it is now possible to perform detailed studies on different individuals and to find the differences between two genomes. But we are still incapable of determining the complete genome sequence by assembling the corresponding reads. Table 1.2 shows a comparison of some of the next-generation sequencing technologies from [16].

### 1.3 Genomic Challenges to be Addressed

As mentioned earlier in this chapter, assembling different reads into a complete genome sequence is not feasible yet due to the size of the data and having error prone technologies. This turns many existing unsolved problems into difficult challenges in the genomics field, namely finding the complete set of differences between two sequences. The following are some of the problems related to the DNA sequencing.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Read length (base-pairs)</th>
<th>Run Time (day)</th>
<th>Gb per Run</th>
<th>Machine cost (USD)</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roche 454’s Genome Sequencer FLX Titanium</td>
<td>330</td>
<td>0.35</td>
<td>0.45</td>
<td>500,000</td>
<td>Longer reads improve mapping in repetitive regions; fast run times</td>
<td>High reagent cost; high error rates in homopolymer repeats</td>
</tr>
<tr>
<td>Illumina Solexa’s Genome Analyzer II</td>
<td>75 or 100</td>
<td>4, 9</td>
<td>18, 35</td>
<td>540,000</td>
<td>Currently the most widely used platform in the field</td>
<td>Low multiplexing capability of samples</td>
</tr>
<tr>
<td>LifeAPGs SOLiD 3</td>
<td>50</td>
<td>7, 14</td>
<td>30, 50</td>
<td>595,000</td>
<td>Two-base encoding provides inherent error correction</td>
<td>Long run times</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

1.3.1 Mapping

One of the main drawbacks of high throughput sequencing technologies is that they can only produce short sequence reads. For example, Illumina’s Solexa Genome Analyzer\(^1\) can produce reads of length between 50 base pairs up to 150 base pairs. Also, in order to increase the coverage rate, each part of the genome will be sequenced more than once; as a result, we will get a lot of short chunks of genome which overlap each other.

In the mapping problem, based on the fact that the genome sequences for two individuals of the same species are very similar to each other, we try to map the short reads from the target genome (called donor genome) to the reference genome sequence to find the entire sequence. In other words, a mapper finds the part of reference genome that is most similar to a given short read (i.e. it tries to locate a part of the reference genome with the minimum difference with the current read). MrFAST [1] and mrsFAST [6] are two algorithms available for mapping.

1.3.2 Structural Variation Discovery

Another important problem in the genomics field is structural variation discovery. Although genome sequences of two individuals of the same species bear considerably great similarity with each other, what differentiates the two is originated from the differences between their corresponding genomes. These differences are called variations and can be the cause of different eye colors, diseases, etc. Variations come in one of the following forms: (a) Single-Nucleotide Variation (SNV) which refers to the change of one single nucleic acid or (b) Small INDELS which are the insertions or deletions of some base pairs and (c) Structural Variation (SV). SV is the variation in the structure of a chromosome of an organism which is possible between two individuals of the same kind and is often of a size larger than 50 base pairs. In figure 1.1 you can see deletion, insertion and inversion SVs.

It has been found that these variations can be the underlying cause of some diseases in an affected individual. Diseases like psoriasis [9], HIV susceptibility [5], epilepsy [15] [8], renal disease [15], diabetes [15] and autism [3] are some examples of these genetic disorders. VariationHunter [10] is one of the available algorithms capable of finding insertion, deletion and inversion SVs.

\(^1\)http://www.illumina.com/technology/solexa_technology.ilmn
1.3.3 Gene Fusion Discovery

Sometimes, a structural variation happens that affects two genes simultaneously. For example, a part of the sequence is deleted in the conjunction of two genes. As a result, a fusion gene is created. A fusion gene is a gene that is formed from two previously separated genes. These genes are often oncogene, meaning that they have the potential to cause cancer as the newly fused gene may have a different functionality than both of its parents. Figure 1.2 shows a sample fusion gene that is generated as a result of a deletion. Finding the fusion genes of a sequence is another genomic problem that is addressed in Gene Fusion Discovery. Defuse [14] is one of the software packages implemented for tackling these problems.

Figure 1.2: Gene fusion due to deletion
1.3.4 Novel Sequence Insertions Discovery

It is estimated that 19-40Mb of the human genome sequence is missing in the human genome reference [7]. Finding the parts of the human genome that were not assembled in the reference genome is another problem known as Novel Sequence Insertion Discovery. To put differently, finding parts of the human genome that are available in the donor’s sequence, but are missing in the reference genome is called novel sequence insertion discovery.

Many of the structural variation algorithms focus on finding short SVs like short insertions, deletions, duplications, etc., but less effort has been dedicated to finding the longer sequence insertions. This is due to the fact that the detectable sequence length with read pair analysis is limited to the insert size. In other words, you can not find insertions of size larger than the read size. The insert size is small in the new technologies. Thus, the longer insertions can not be found with the SV discovery methods. Novelseq [7] is an algorithm which tries to find these sequences.

1.4 The 1000 Genomes Project

An example of a genomic project which is currently in progress is The 1000 Genomes Project. This project was launched in January 2008 as an international project to sequence 1000 human genomes from different ethnic groups all over the world. One of the goals of this project is to create a complete catalogue of human genetic variations. Genetic variations are the genetic differences between populations which happen due to the differences between their genomes. The 1000 genome project can also help improving our currently available human genome reference. In order to reach this goal, we need to solve the problems mentioned in section 1.3. It is also worth mentioning that, as of today, the pilot phase of this project has been finished and the project is currently in its production phase.

1.5 Motivation

Learning how to run new algorithms on a dataset is not always easy since each new algorithm might be developed in a different programming language, has its unique set of parameters and may work with different input and output file types. On the other hand, not every experiment can be run in a single step. There are multi-level experiments that should be executed one step at a time. Also, you do not always run an experiment on a single machine;
you may want to test it on your local machine before running the experiment on your server. Even if you are willing to do all these steps manually, working on an SSH console for running experiments or copying files through SCP commands on a text terminal, are tasks that a lot of users may not be comfortable to tackle. All these difficulties can make conducting an experiment hard and slow.

Moreover, you may want to run a multi-level experiment on different datasets. The only way of avoiding repetitive tasks in each step is writing shell scripts which may not be easy for all the users. Also, from the developers’ point of view, when you are releasing an algorithm, you have to write it in a user friendly way, while most of the times, you don’t provide a GUI for your package.

These issues suggest the need to develop a software tool for researchers which reduces the difficulties of creating software packages and running genomic experiments.

1.6 Some of Currently Available Software

Most of the works around developing software tools for genomics data have been dedicated to either visualizing the genome sequence - such as the UCSC Genome Browser [13] (online) and Savant [4] (desktop application) - or creating tools that contain a single - such as mrFAST [1] and defuse [14] - or a collection of bioinformatics algorithms of the same category - like Genome Tools\(^2\) and Genome Threader\(^3\).

When it comes to integrated software packages, most of the packages are focused on the assembly and analysis of next generation sequencing data, namely Genomatix\(^4\) and CLC Genomics Workbench\(^5\). We could not find any software application that integrates different tools of different categories. None of these tools can help the user in designing multi-step experiments or running them over the network.

The following is a brief introduction to two available software packages, the UCSC Genome Browser and the Savant Genome Browser. We have used the former in one of our components and the latter is the most similar available tool to ours.

UCSC Genome Browser : Having a genome sequence in a string format that consists of

\(^2\)www.genometools.org
\(^3\)www.genomethreader.org
\(^4\)www.genomatix.de
\(^5\)www.clcbio.com
millions of characters is not easy to interpret and thus, not really useful. The UCSC genome browser [13] is an online tool developed at the University of California at Santa Cruz that helps visualizing genome sequences. With the help of this tool, users can browse and display any part of the genome and select among a set of aligned annotation tracks that are available in the browser. These tracks contain information such as location of genes, assembly gaps, chromosomal bands, etc. You can also add your own custom track to the browser. Therefore, the UCSC genome browser is a great tool for visualizing genome sequence.

**Savant Genome Browser** : Savant [4] is another visualization tool for genomic data created at the University of Toronto. The difference between the UCSC genome browser and Savant is that Savant is a desktop application which can provide you with more features and more user friendly interfaces and plugins. It also provides the user with a set of analysis tools. Furthermore, Savant is a pluggable software tool which allows developers to create their own analyzer components.

### 1.7 Our Problem

To the best of our knowledge, there is no tool available that integrates the different algorithms of different genomics problems. Many bioinformatics research groups publish their software tools as separate packages on their websites while application developers’ attention is mostly devoted to creating tools to ease the genome visualization in the genomics community. Furthermore, using new algorithms may be difficult as discussed in section 1.5.

These issues motivated us to create a software tool that integrates the different genomic algorithms and makes running experiments a much easier task. Using our tool, researchers can focus on their experiments and their results rather than the technical issues they may face while executing experiments. They can also benefit from a powerful software tool that integrates different algorithms and enables them to create multi-level experiments with a simple user friendly visual interface.
Chapter 2

CB-Commander

In this chapter we start with a brief introduction to CB-Commander in section 2.1, followed by the complete list of use case descriptions and their functionalities in section 2.2. Section 2.3 describes the key features which differentiate CB-Commander from similar softwares in the field of bioinformatics. Finally, in section 2.4, we present the development platform used.

2.1 About CB-Commander

CB-Commander is a software tool that we developed at the Computational Biology Lab\(^1\) of Simon Fraser University. This tool is designed to assist biologists and computational biology researchers during the design and execution processes of their experiments. CB-Commander enables users to integrate different components of an experiment, e.g. algorithms and converters, into one graphically interfaced application that is very easy to use when working on remote servers as well as local computers. The graphical user interface of CB-Commander facilitates a visual design of experiments by using a block diagram to represent the components (algorithms, converters, etc.) of an experiment as a pipeline. The users can easily modify this pipeline.

The SSH support in CB-Commander is entirely transparent to users and makes the file transfer between the remote server and local computer effortless. Providing a simple Java

\(^1\)Website: http://compbio.sfu.ca
Bean\textsuperscript{2} class for inputs and outputs of any new algorithm, CB-Commander can adaptively interface that algorithm. Based on these distinctive features of pluggability, remote access, and visual design, CB-Commander can provide a powerful solution for both academic and bioinformatics research uses.

![Figure 2.1: Main form of CB-Commander - Logo by Bryan Ballinger: www.breadwig.com](image)

2.2 Use Cases

Figure 2.2 shows the use case diagram for CB-Commander. The description for each of these use cases are provided in the following subsections. Note that the descriptions only include the main flow of the corresponding scenario and any possible error or exception (like connection loss, error in inputs, etc.) is disregarded in these descriptions.

2.2.1 Install New Plugin

This use case shows how a new plugin (a new algorithm or converter) is added to the system. We assume that the user provides a valid component as a plugin, meaning that the

\textsuperscript{2}More information available in section 3.1.1
Figure 2.2: High level use case diagram
component includes the complete source code, configuration file and the input beans\(^3\).

Once the user downloads the package from the publisher’s website, they should copy it into an appropriate folder of the program. By restarting the tool, the system automatically finds the new component and appends it to the list of available components in the system.

### 2.2.2 Change Workspace

Users can run their experiments locally (on their local machine) or remotely on a server through SSH connections. The Users’ workspace is the folder on the server’s hard disk (either local or remote) that is used for deploying the experiments. Source codes of the components, experiment files and their inputs and outputs are some of the files stored in this workspace. Before being able to design and create experiments, the user must specify a workspace.

**Scenario:** When the user requests modifying a workspace, the system first asks whether the workspace is local or remote. If the workspace is local, then the user can browse the file system and select a workspace folder locally on the current machine. If they want to run the experiment remotely, the system asks for server name, port number, user name and password to connect to the server. Having these information, it tries to connect to the remote server using the specified addresses and login credentials. After the connection is made successfully, the user can browse the server’s file system and choose a folder as their workspace. After the workspace is set, the system creates the necessary folder structure and files for running the experiments. From this point forward, experiments will be deployed in the folder specified.

### 2.2.3 Install Components on Server

The user can not use a component (algorithm, convertor, etc.) unless it is installed on the server (local or remote). Installing a component on the server is the process of copying the necessary files to the workspace and compiling them on the server.

**Scenario:** The list of currently installed algorithms in the system is shown to the user. The user can select the algorithm(s) to be installed on the server. Once the desired

\(^3\)For detailed information on how to create a new plugin, refer to section 3.2.
algorithms are selected, the tool creates the essential folders on the server, copies related files, compiles the copied files and the algorithm is now ready to be used on that server. It should be noted that these algorithms will be installed on the user’s current workspace.

2.2.4 Transfer File to Server

With this feature, users can upload their input files to the server using CB-Commander, so they don’t need to manually connect to their servers using an FTP client or any similar programs to copy their files.

Scenario: The system asks for the path of the files to be copied to the server. After the files are selected, the system asks for the target folder on the remote machine for transferring the files. Once both source and destination are specified, the system copies the files to the target folder through the SCP$^4$ protocol.

2.2.5 Create Experiment

One of the main capabilities of CB-Commander is assisting the user in the design of their experiment. An experiment in CB-Commander consists of a set of tasks each handled with a different component. These components can be pipelined, meaning that the output of one component can be piped into the other component as its input. The user can install

$^4$Secure Copy is a protocol used for transferring files over SSH
different components such as algorithms, visualizers, comparators and converters, and then use them as any part of the pipeline to get the desired output from the experiment.

Figure 2.4 shows a sample experiment in CB-Commander. This is a simple experiment design for running the VariationHunter algorithm. In this example, we can see four components: Start, DivetReader, SetCover and Maximal Cluster. The edges between the nodes determine the dependencies of components. For example, in this figure, SetCover is dependent to DivetReader and MaximalCluster, as it uses some of their outputs as its input. Every pipeline in CB-Commander has a Start node which shows the starting point of the experiment. The components immediately after the Start node are the first components which will be executed, provided that all of their inputs are ready. In this digram, DivetReader is the first component that is executed. SetCover and MaximalCluster cannot yet start, as they are dependent on DivetReader. Once DivetReader is terminated, MaximalCluster can start its execution. After the termination of MaximalCluster, SetCover can start as both of its dependents are now executed and their outputs are ready.

**Scenario:** The system shows the list of currently installed plugins to the user. The user can now drag and drop the components on the screen and connect them through directed links. A directed link indicates that the target component uses the outputs of the source component (totally or partially) as its input and should be executed after the source one. Once two components are connected, the user should specify which outputs of the source should be directed to which inputs of the target. Then the user can set the parameters of each component instance as well as the parameters of the links between the two connected components.
Figure 2.4: Create pipeline form - Sample experiment designed in CB-Commander

2.2.6 Save/Load Experiment

CB-Commander allows the users to save the current designed experiment, in order to be able to load it for later runs with different settings. By saving an experiment, the system creates a file based on the experiment’s graph of components. This file contains all the parameters required for loading the experiment structure and settings later.

2.2.7 Validate Experiment

After designing an experiment, the user can request validation of the designed experiment. During the validation process, the system automatically searches for some common problems such as data type mismatch, parameter values out of valid range, loops in the component graph, nodes (algorithms or converters) with no input edge, etc. Then, the system informs the user of any issues found. Users can not run experiments before validating them.
2.2.8 Run Experiment

When a run is initiated, CB-Commander automatically generates an executable script file, copies that file to the server and executes it. This way the user can still use CB-Commander while the experiment is running and the execution is not terminated if the user quits CB-Commander. The user can also check the status of their running experiments at any time during execution (section 2.2.9). After a run is completed - terminated either successfully or due to an exception - the user can download and view the generated outputs.

Scenario: When the user starts running an experiment (either local or remote), the system runs the Validate Experiment (section 2.2.7) use case. If the design is valid, then the system asks for saving the experiment in case it is not currently saved. Once the experiment is validated and saved, the main run script will be generated. Then the necessary folders and files will be created and copied to the workspace. Finally, CB-Commander runs the main script. A proper message is shown to the user in case of a successful run.

2.2.9 Check Status of Experiment

Users can check the status of their experiments at any time. The status report provides the messages from the error and output consoles as well as messages about the current state of the experiment (e.g. ‘Running’, ‘Completed’). The report also shows which steps of the experiments have been completed.

Scenario: The list of available experiments in the current workspace is shown to the user. They can choose an experiment to view its status. The system shows a report to the user, containing the current status of the experiment and messages from the output and error consoles.

2.2.10 Download Output Files

In order to be able to edit the output files manually or to use a visualizer for viewing the results, the user should copy the outputs to their local machine. Using CB-Commander’s interface, users can browse different folders in the workspace and ask the system to download their selected files from the server to some folder on the local machine. All of the network related tasks like SSH and SCP are transparent from the user’s view.
2.2.11 Visualize Results

Scenario: To visualize the outputs, the system shows the set of available visualizers to the user. The user chooses the appropriate visualizer and the output file. User can then work with the visualizer through its panel.

![Figure 2.5: Structural variation visualizer](image)

2.3 Unique Features

In this section, we will present some of the features of CB-Commander that facilitate conducting experiments.
2.3.1 Pluggability

CB-Commander is plugin based; that is, writing and adding a new component to this tool is easy. The followings are what you need for creating a CB-Commander plugin from your algorithm: (a) your algorithm’s source code, (b) a makefile for compiling the code and (3) a list of the parameters of your algorithms. The only constraint on the code is that you should be able to run your code from the command line\textsuperscript{5}.

Having these requirements ready, you need to write a simple configuration file for your component and introduce the parameters of algorithms to CB-Commander via a Java bean file. That’s all you should prepare to add your algorithm to CB-Commander. The details of how to create a new plugin for CB-Commander are discussed in section 3.2.

After preparing these files, if you copy them in the plugin folder of CB-Commander and restart the tool, you will be able to use your new component.

2.3.2 Automatic GUI Form Generation

As a plugin developer for CB-Commander, you do not need to be concerned about the graphical user interface (GUI) of your components. CB-Commander automatically creates the input forms for your components. It also adds some simple range check and type validations on the inputs. The known input types for CB-Commander are \textit{integer}, \textit{float}, \textit{double}, \textit{string}, \textit{file} and \textit{user defined enumerations}. Not only does CB-Commander support these types, but it also supports \textit{collections} and \textit{complex collections}. More information on the known input and output types is available in section 3.2.1.

Figure 2.6 shows a sample form that is automatically created. This form is generated for \textit{SetCover} algorithm’s parameters. This is an example of a form which contains parameters of types \textit{String} (Library Name), \textit{Integer} (Number of Classes), user defined enumerations (Library Type), \textit{File} (Reads Sorted) and \textit{List} of user defined types (Libraries). You can see that for the libraries parameter which is of type list, the user interface has automatically added a table for managing the content of the list. You can also see that some of the fields in this figure are disabled. If a field is disabled, it shows that its value is set through the pipeline through a connection to another component’s output.

\textsuperscript{5}This can be easily done through a launcher class
2.3.3 Pipelining

Redirecting the outputs of an algorithm to one or more other components as their input is not always a straightforward task to do. Sometimes, you also have to run a converter on the output file to adapt it to fit the inputs for the next algorithm. This means that you have to wait for the first component to terminate, create the converted file using the output and then, manually run the second component.

With CB-Commander, you can design a pipeline of different components. It automatically takes care of the different component calls. When one component terminates, CB-Commander prepares the input for the next component and executes it. Thus, you
don’t have to wait for different steps of the experiment. You design the experiment and CB-Commander handles the entire execution process.

Figure 2.7 shows the connection editing form of CB-Commander. When you connect two components together (adding connection) or when you double click on an existing edge (editing connection), a form similar to this figure is shown. In this form, the source panel includes the list of outputs of the source component and the target panel contains the list of output parameters of the target component. After connecting the parameters, they will be added to the connection list which is the list located at the bottom panel. e.g. in this figure, parameter ClusterOut of the source node which is an instance of MaximalCluster component is piped into parameter Cluster of a SetCover instance.

![Figure 2.7: Edit Connections](image)
2.3.4 SSH and Remote

Running experiments through SSH terminals, where everything is in textual format, is not a user friendly task. With the help of CB-Commander, these commands will be executed in a way completely transparent to the user. The only thing the users need to do is to work with the graphical user interface and then CB-Commander translates everything into the appropriate command line commands and runs the experiments on the servers.

2.3.5 User Friendly GUI

CB-Commander’s graphical user interface (GUI) is capable of fulfilling many common tasks needed for running the experiments. Handling tasks such as connecting to a server, browsing the local and remote folders, uploading and downloading files, running experiments and checking the status of runs are all possible through the GUI of CB-Commander. Dragging and dropping components for designing an experiment is a much easier task than writing command line scripts. Also, through this GUI, you have the option of saving and loading your experiments to be able to run them on different machines and datasets. It should also be noted that closing CB-Commander will not terminate the experiments; you can always come back to check the status and the outputs of previous executions from the interface.

2.4 Platform

CB-Commander is developed in Java 1.6. The implementation is mostly done in JetBrains’ Java development environment, IntelliJ IDEA 10.0 community edition. It provides the developers with a lot of features like fast and smart code completion, refactoring tools and version control support. The community edition of this IDE\(^6\) is freely available for download from their website \(^7\).

During the implementation phase, we used Subversion as our source control system. Our Subversion hosting was provided by Codesion\(^8\) which is a provider of different source control systems like Git, CVS and Subversion.

The graphical user interface of the system is implemented with Java’s Swing library

\(^6\)Integrated Development Environment
\(^7\)http://www.jetbrains.com/idea
\(^8\)http://www.codesion.com
which is the primary Java GUI widget toolkit. However, for layout management, we used the MigLayout library. Miglayout\(^9\) is a free and open source Java layout management library that is powerful and easy to use. Most of the forms, including the automatic generated forms of CB-Commander are designed using this library.

*JGraph*\(^10\) is the graphical library which we used for graph visualization. This free and open source library is written in Java and can be used for drawing graphs and visualizing them. It also comes with a set of layouts to automatically position your diagrams.

For handling the remote tasks we used the *Java Secure Channel (JSch)*\(^11\) library. Everything from browsing the server’s directory to executing the codes and file transfers between machines are implemented over this library. JSch is purely implemented in Java and enables you to connect to an SSH server, use port forwarding, file transfers, etc. We implemented a set of wrapper classes around JSch as well as an SSH file browser component which allows users to browse the file system of the server.

The CB-Commander logging system is implemented over *Apache’s*\(^12\) logging component of the *Apache Commons Project*\(^13\) and *log4j*\(^14\). These libraries enable developers to include high performance low cost logging systems in their tools.

\(^9\)http://www.miglayout.com/
\(^10\)http://www.jgraph.com
\(^11\)http://www.jcraft.com/jsch/
\(^12\)www.apache.org
\(^13\)http://commons.apache.org/
\(^14\)http://logging.apache.org/log4j
Chapter 3

Using CB-Commander

In this chapter, we explain the procedure of adding your own plugin to CB-Commander in
detail. First, we begin with defining a few Java programming concepts and then, in section
3.2, we continue with the procedure of creating a CB-Commander plugin from an existing
implemented algorithm.

3.1 Preliminaries

3.1.1 Java Bean

In Java programming, a class is called a *Bean*, if (1) that class has a default constructor\(^1\)
and (2) for each of its member variables, there exists a *getter* and a *setter* method. Setter
and getter methods are used for setting and retrieving the values of these variables respec-
tively. The getter and setter methods should exactly follow this naming convention: For
a variable \(xyz\) of type \(T\), the signature\(^2\) of *getter* and *setter* methods must be \(T \text{ getXyz}()\)
and \(\text{void setXyz}(T \ t)\). In other words, we capitalize the first letter in the variable’s name
and concatenate it to ‘get’ for getter or ‘set’ for setter methods. Following this naming
convention is critical for having a correct automatic form generation. Program 3.1 shows a
simple Java bean with two member variables.

\(^1\)A default constructor is a constructor method which takes no parameter

\(^2\)A method signature includes the method name, and the number, types and order of its parameters
public class SampleBean {
    private Integer i;
    private String name;

    public SampleBean() {}
    public Integer getI() {return i;}
    public void setI(Integer arg) {i = arg;}
    public String getName() {return name;}
    public void setName(String argName) {name = argName;}
}

Program 3.1: A simple Java bean with two member variables

3.1.2 Java Annotation

Java annotation is a kind of metadata that can be added to different Java source code elements such as variables, classes, and methods. What makes annotation different from Javadocs or comments is that this metadata information can be extracted at run-time. Program 3.2 shows an integer field with annotation for its minimum and maximum allowed values. Having this information in the Java bean, we can check the value of the age field at run-time and see if it is within the valid range.

public class AnnotationTest {
    @AgeMetaData {minValue = "1", maxValue = "120"}
    private Integer age;

    ...
}

Program 3.2: A sample Java annotation for an integer field

3.2 How to Create Your Plugin

Introducing the concepts of Java bean and Java annotation, we can now explain how you can develop a plugin for CB-Commander. In this section assume that we are trying to create a component for a program that is executable from the command line; e.g.:
CHAPTER 3. USING CB-COMMANDER

./myAlg -i <number_of_iterations> -o <filename>. This program gets a number as input and generates a file as output. The name of the output file should be provided in the -o argument.

3.2.1 How to Introduce the Parameters

First step towards creating a new component is introducing its parameters to CB-Commander. CB-Commander should know what the different input and output parameters of an algorithm are, to be able to generate a form for it.

To introduce these parameters to CB-Commander, you have to use Java beans. All of the parameters of your algorithm, both input and output parameters, should be included in a Java bean which inherits from the CommanderBean class. For our example, we should prepare a Java bean like the one presented in program 3.3.

```java
public class SampleBean extends CommanderBean {
    private Integer numberOfIterations;
    private File outputFile;

    public SampleBean() {}  
    public Integer getNumberOfIterations() {return numberOfIterations;}
    public void setNumberOfIterations(Integer n) {numberOfIterations = n;}
    public File getOutputFile() {return outputFile;}
    public void setOutputFile(File arg) {outputFile = arg;}

}
```

Program 3.3: A sample Java bean

CB-Commander supports two categories of data types: simple and complex. The former includes basic types of Boolean, Integer, Float, Double, String and File, while the latter includes user defined types. Complex data types can be in the form of user defined enumerations, other CommanderBean classes or collections of either simple or complex data types. Program 3.5 shows the complete example of the Java bean defined for a CB-Commander plugin. Its corresponding generated form is shown in Figure 2.6.
3.2.2 How to Add Metadata to Parameters

To be able to execute an algorithm, we need more information than the types of its input and output parameters, but Java beans can only help us describe the types of different parameters. To provide more metadata for parameters, we have to use field annotations. Different annotations that you can use to define metadata for your parameters are listed below:

- **min** - optional - Specifies the minimum valid value of a numeric parameter.
- **max** - optional - Specifies the maximum valid value of a numeric parameter.
- **description** - required - Is the text explaining extra information about the parameter to the user. This description is shown to the user as a tooltip when they hover over the parameter.
- **defaultText** - optional - Is the default value for the parameter. When the program generates the form, it will set the value of the corresponding parameter to this default value.
- **parameter** - optional - shows the argument of this parameter on the command line. In our previous example, `-o` is the value of *parameter annotation* for the output file field.
- **required** - optional - is a boolean parameter which tells CB-Commander whether an input parameter is required or optional.
- **labelText** - required - is the text that is used as the label of the parameter in the generated input form.
- **className** - optional - is used for complex data types.
- **isOutput** - required - when set to true, it indicates that the parameter is an output variable, otherwise an input variable.
- **isConnectable** - optional - must be set to *true* if this input parameter is connectable and can be piped in from other components.
3.2.3 What Properties Should the Algorithms Have

Before talking about the properties of the algorithm, let us briefly explain the procedure of installation and execution of a component in CB-Commander.

When you install a new algorithm on a server, CB-Commander first copies the source code of that algorithm to the server and then tries to compile the code on the server. To be able to compile the algorithm’s code, CB-Commander requires the code to come with a valid makefile. Also, the algorithm should be compilable and executable on the server machine.

If the above-mentioned phases are completed successfully, the component will be added to the list of available components and the user can use it in their experiment designs. Next step is executing the algorithm. As CB-Commander uses SSH connections for running the experiments, it can only execute the programs which are executable in command line - i.e. the ones without any graphical user interface. Also, these algorithms should not need user interaction during execution. In other words, all of the input parameters should be passed to the algorithm as arguments before execution starts.

In summary, the algorithm should be compilable and executable on the server, have a valid makefile, should be executable in text based consoles and it should not have any run-time interactions with the user in order to be converted to a CB-Commander plugin.

3.2.4 How to Change the Default Procedures

CB-Commander has a default procedure for executing algorithms. However, there are algorithms that can not be executed through this default procedure. For these algorithms, CB-Commander allows you to change some of the phases of the execution process through overriding some methods of Wrapper classes. A wrapper class is the main class responsible for the execution of an algorithm. DefaultCommanderWrapper is the default wrapper available in CB-Commander’s package. Unless mentioned in the configuration file\(^3\) of an algorithm, the DefaultCommanderWrapper is assigned to an algorithm as its wrapper. In case you need to change this default procedure, you have to implement your own Wrapper class, inherit it from DefaultCommanderWrapper and override the necessary methods.

Before explaining the different methods that can be overridden in a Wrapper class, we will briefly present the different phases of executing an algorithm in CB-Commander. To

\(^3\)Refer to section 3.4 for more information on the configuration file.
execute an algorithm, first, CB-Commander tries to validate the experiment’s parameters. If the validation phase is passed successfully, then preparation routines are executed on the parameters\(^4\). After that, CB-Commander copies the content of the algorithm’s work directory\(^5\) to the workspace of the user on the server. Finally, the execution command of the component is generated by concatenating the executable file name of the algorithm, its different parameters and their corresponding arguments.

You can modify the default procedure of some of these phases through overriding the following methods of your Wrapper class.

**validateBean**  The default validation process only looks at the valid ranges (for numeric values) and null values (for the fields which are flagged as required). Any other validation that you might need should be added to this method.

**prepareExperiment**  There are some scenarios in which some preparations are necessary before the execution of an algorithm. For example, a developer may want to create a file from the list of integers that are set through the GUI and pass that file as an argument to the algorithm. Tasks like this should be implemented in this method. This method will be called after `validateBean` and before `deployFiles` method. Files created should be put in the experiment’s work folder - which is passed as a parameter to the `prepareExperiment` method. All of the content of the work directory will be copied to the workspace during the `deployFiles` step and therefore is accessible on the server. A sample implementation of this method is provided in program 3.6.

**createRunScript**  This method is responsible for creating the command line for executing the algorithm. The default implementation iterates on every parameter and concatenates the command line with the parameter and its proper argument (extracted from its annotations). If there are parameters which should not be added to the command line or if there are extra parameters that should be added, such as the newly created files in the previous steps, this is where you should add the handling code for them.

\(^4\)These steps are explained in detail in the following sections.

\(^5\)When an instance of a component is created in an experiment, a *work directory* is created and assigned to that instance. This directory can be used as a storage space for that instance.
3.2.5 How to Pass the Different Information to CB-Commander

After you are done preparing the code, creating the Java bean and implementing your wrapper (if necessary), you have to introduce them to CB-Commander. This is possible through one configuration file for each component. Each component is required to have a configuration file containing all of the following information:

**name** is the name of the component. e.g. “VCF to DIVET Converter”

**version** is the version of the code, useful when you have more than one version of an algorithm

**exec_file** the name of the executable file of the algorithm after it is compiled. It can also be the name of a script file created for running the algorithm.

**installationFilesPath** is the path for the source code of the algorithm. This folder should be in the folder of the plugin.

**bean** is the CommanderBean class that contains the information about the inputs and outputs of the algorithm.

**wrapper** is the class of the algorithm’s wrapper class - if any.

Program 3.4 shows a sample configuration file.

3.2.6 A Sample Plugin

In this part, we present the different parts of a sample plugin. Program 3.4 shows a sample configuration file; a piece of the Java bean for the plugin is shown in 3.5 and finally, program 3.6 is the overridden `prepareExperiment` method of the corresponding wrapper class.

3.3 Currently Added Components

In this section, we introduce the components that we have developed for some of the algorithms created at the computational biology lab of Simon Fraser University. These components are already implemented and added to CB-Commander. The following is the list of these components with their brief descriptions.
CHAPTER 3. USING CB-COMMANDER

#Configuration File for MaximaCluster Component
name = Maximal Cluster
version = 1.0
exec_file = VH
installationFilesPath = code/
bean = ca.sfu.compbio.vh.MaxClusterBean
wrapper = ca.sfu.compbio.vh.MaxClusterWrapper

Program 3.4: Sample configuration file for a component

Variation Hunter: Variation Hunter is an algorithm which was first introduced in [10] for finding the structural variations in genomes. The Maximal Cluster and Set Cover components are the two components developed for the two different phases of this algorithm. It should be noted that the latest version of this algorithm [11] has been used for development of components.

SV Visualizer: To be able to visualize the outputs of structural variation discovery tools, we developed a visualization component for CB-Commander. Given the output file of the algorithm, our visualizer loads the file and shows the different variations to the user. You can sort the entries by different columns and also filter them according to their variation type and starting and ending points. There is also a link next to each variation that takes you to the UCSD genome browser [13] website. Figure 2.5 is a screenshot of this component.

SV Output Comparator Given the output of two SV discovery algorithms, this component can compare the files and report the differences.

Sam2Divet Converter Some versions of several algorithms in our lab work with DIVET input files. However, some assemblers generate SAM files as their output. Sam2Divet Converter is the component that gets a SAM file as input and returns a DIVET file as output.

Novelseq Novelseq is the algorithm introduced in [7] which is developed for finding the novel insertion sequences in the genome.

Defuse This algorithm, introduced in [14] is for gene fusion discovery using RNA-Seq data.
public class VH2VCFBean extends CommanderBean {
    @CommanderAnnotation(
        labelText = "VH Calls",
        isOutput = false,
        parameter = "-i",
        isConnectable = true,
        required = true,
        description = "Input the calls file generated with VariationHunter"
    )
    private File vhCalls;

    @CommanderAnnotation(
        labelText = "Min Support",
        isOutput = false,
        parameter = "-min",
        required = false,
        defaultText = "2",
        description = "Minimum number of supporting read pairs. Default is 2"
    )
    private Integer minSupport;

    @CommanderAnnotation(
        labelText = "Prune Filter",
        isOutput = false,
        parameter = "-p",
        required = false,
        defaultText = "",
        description = "Threshold to prune based on concordant read depth."
    )
    private Boolean pruneFilter;

    public VH2VCFBean() {..}
    public File getVhCalls() {return vhCalls;}
    public void setVhCalls(File vhCalls) {this.vhCalls = vhCalls;}

    ...
}

Program 3.5: Part of a CommanderBean class
```
@Override
public void prepareExperiment(File workDir) {
    SetCoverBean bean = (SetCoverBean) algorithmInstance.getBean();

    PrintWriter writer = null;
    try {
        libFile = File.createTempFile("libtmp", null, workDir);
        libFile.deleteOnExit();

        writer = new PrintWriter(new FileWriter(libFile));
        writer.write(bean.getLibraries().size() + "\r\n");
        for (ReaderInputBean library : bean.getLibraries()) {
            StringBuilder builder = new StringBuilder();
            String path = library.getLibFile().getPath();
            path = path.replace(\\, "/");

            builder.append(library.getLibName()).append("\t")
                .append(library.getIndividualName()).append("\t")
                .append(path).append("\t")
                .append(library.getMinDelta()).append("\t")
                .append(library.getMaxDelta()).append("\t")
                .append(library.getReadLen()).append("\t")
                ;

            writer.write(builder.toString());
        }
    } catch (IOException ignored) {
    } finally {
        if (writer != null)
            writer.close();
    }
}
```

Program 3.6: A sample prepareExperiment implementation
Chapter 4

Conclusion

There are many technical problems that may happen to genomics researchers while designing or executing their experiments. At the beginning, learning how to execute a new algorithm may be a slow and difficult task for them. Later on, they might find it hard to execute multi-level experiments and converting the inter-process file types. In addition, they may not have any experience working with a remote server through a text based interface, transferring files using SFTP and SCP protocols. Such problems not only slow down the process of designing and running the experiments but also cause a strong feeling of resistance against using new algorithms. Therefore, having a tool that can take care of all these technical issues of conducting experiments can be extremely handy for the researchers who deal with multi-level experiments with more than one algorithm.

We designed and developed CB-Commander as a tool that aims at integrating different algorithms in the genomics field. This plugin based tool provides a systematic user friendly graphical interface for researchers and for the algorithm developers and provides them with the option of adding their algorithm to the system. In CB-Commander, the users are able to design their experiments in a graphical interface and set the parameters of the algorithms through the GUI\(^1\). Tasks from transferring files, experiment design, setting and changing the parameters’ values, execution of experiment and visualization of the results are all possible through the GUI of CB-Commander. All of the network tasks are transparent to the users view and running an already created experiment on a new server machine is as easy as logging in to the new server and pressing the execute button.

\(^1\)Graphical User Interface
We believe that CB-Commander can help researchers to focus more on their experiments, instead of the technical issues and can increase the speed and efficiency of conducting experiments. Furthermore, as this tool contains different algorithms for each problem, it can serve as a useful tool for academic purposes.
Chapter 5

Future Work

The following is the list of the features that we found useful to be implemented in later versions of CB-Commander.

- Developing an interface or a wizard for automatic creation of a configuration file and Java beans. Having this interface, the users will not need to know anything about Java beans and Java annotation concepts.

- Providing a facility for handling and controlling the processes that are being executed on the server machine. Currently, when an experiment is started, the users can not kill it through CB-Commander.

- Developing different wrappers other than the DefaultCommanderWrapper which are capable of executing other kinds of algorithms. By adding this feature, fewer developers should develop their own wrapper class in order to be able to execute their algorithm with CB-Commander.

- Developing more built-in comparators, report generators and visualizers will address more users.

- Providing support for multi-level SSH connections in order to be able to pass the servers which are behind a firewall.

- Providing a plugin directory website for storing the different plugins.

- Having better validations and exception handling on exceptions happening at the server side.
• Making the experiment design environment more intelligent. E.g. in connecting two components, we can determine which parameters should be connected to each other.
Bibliography


BIBLIOGRAPHY


