University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Clique-finding tool for Detecting Gene Clusters

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Bianca Camille L. Silmaro

June 2018

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Clique-finding tool for Detecting Gene Clusters" prepared and submitted by Bianca Camille L. Silmaro in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D (*cand.*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*cand.*)	_____	_____
2. Avegail D. Carpio, M.S.	_____	_____
3. Richard Bryann L. Chua, Ph.D. (*cand.*)	_____	_____
4. Perlita E. Gasmen, M.S. (*cand.*)	_____	_____
5. Marvin John C. Ignacio, M.S. (*cand.*)	_____	_____
6. Ma. Sheila A. Magboo, M.S.	_____	_____
7. Vincent Peter C. Magboo, M.D., M.S.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.S.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

Defining relationships between species is a fundamental problem in bioinformatics. One of the ways to define relationships is to detect gene clusters, and can be formulated as a combinatorial problem called Approximate Gene Cluster Discovery Problem (AGCDP). Graph concepts have been applied to several genomic studies. AGCDP can be reduced to optimization problems in graph, specifically the Minimum Weight t-Partite Clique Problem (MWtCP). The goal of MWtCP is to create a t-partite graph and to find a t-star with minimum weight, which is used to approximate a t-clique. Clustar is a tool that applies an algorithm which solves the MWtCP for detecting gene clusters. It allows the user to detect gene clusters using three methods: approximate gene clustering, exact gene clustering (using GPU), and exact gene clustering (without using GPU). Clustar is able to produce candidate gene clusters and its alignment among the genomes, as well as the graph representation and the adjacency matrix produced from the generated graph. To verify the validity of the results produced by Clustar, a dataset containing homologous genes from 30 γ-proteobacterial genomes was processed using Clustar and other algorithms such as The Row's Subset of Symmetric Matrix (RSSM) and hierarchical clustering. Several gene clusters were found common across these three algorithms using different gene cluster sizes. Another dataset was used containing genes from *E. coli* and *B. subtilis* where several gene-groups have been established already. Clustar was able to produce candidate gene clusters that matched these gene-groups, using different gene cluster sizes.

Keywords: Keywords gene, genome, gene clusters, clique, approximate gene cluster discovery problem, minimum weight t-partite clique problem

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Defining relationships between organisms is a fundamental problem in comparative genomics and bioinformatics. Comparative genomics is a field in biological research wherein genomic sequences are analyzed to understand the genetic elements defining the commonality and uniqueness among different organisms [1]. Phylogenetic trees are prime examples of the applications of comparative genomics. The establishment of a phylogenetic tree is necessary to understand the evolution of various species and the patterns that occur with it [2]. New viruses and diseases can also be mapped using phylogenetic trees to trace back its origins and to help prevent its spreading to other organisms. For example, the origin of HIV was unknown to researchers until they constructed a phylogenetic tree and found out that HIV evolved from Simian Immunodeficiency Virus (SIV) which affected primates [3].

One of the ways to define relationships between species is to detect gene clusters. A gene cluster is a set of closely-related genes which are arranged in close proximity with each other, even after genome sequences have evolved in multiple events such as gene duplication and gene loss. Genomic regions with relatively similar gene content is a result of duplication and divergence [4]. Gene clusters appear in two or more genomes and perform related functions. Organisms containing similar gene clusters with other species tend to share a common ancestor.

Finding gene clusters between species is essential on mapping the relationships between different species. Several models have been presented on gene clustering such as gene teams [5] or max-gap clusters [6] and common intervals [6]. Genes are also represented as integers and can be modelled either as permutations or strings. Common intervals disallow gaps, while max-gap clusters allow gaps based on the cost of additional and missing genes. Gene cluster models as permutations do not

allow gene duplication while gene clusters as strings relax this constraint and allow repetition of genes [7].

Graph theory has been successfully applied to biological network topology, clusters, and biomolecules [8]. Complex networks in the biological standpoint have utilized concepts from graph theory and are discussed in [8]. Many problems are reducible to optimization problems in graph theory. One of these problems is the Approximate Gene Cluster Discovery Problem (AGCDP), a formulation proposed by Rahmann and Klau [7] on finding gene clusters. AGCDP was represented as a minimization problem on graph. The problem was reduced to finding a star S_n with the minimum weight given the transformed graph [9]. A star S_n of order n is a tree of n vertices where one vertex has a degree of $n-1$ and the other vertices with degree 1 [8].

Clique-finding graph problems have been studied extensively before. A t-clique can be defined as a subgraph of t vertices of an undirected graph such that every two distinct vertices are adjacent [8]. An illustration of a clique is seen on Figure 1. Finding the maximum- or minimum-weighted t-cliques has been useful to many different biological applications [10]. Many approximation algorithms have also been presented to solve clique-finding problems.

In [10], AGCDP was formalized as a clique-problem in graph, particularly the Minimum Weight t-Partite Clique Problem (MWtCP). A t-partite graph is an undirected graph such that its vertices V can be partitioned into t sets and no two vertices in the same set are adjacent [8]. An illustration of a t-partite is seen on Figure 2. The concept of the t-partite graph is an extension of the bipartite graph where a graph can be partitioned into two sets. A 2-approximation algorithm for the Metric Case of Minimum Weight t-partite Clique Problem was introduced in [10] to approximate the MWtCP.

Since complete sequences of genomes are now readily available in online databases,

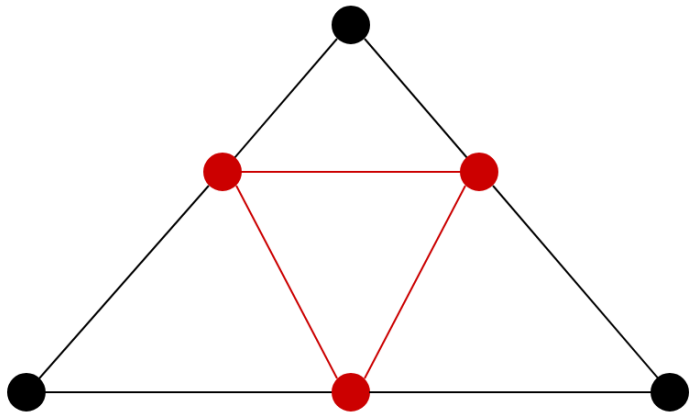


Figure 1: A 3-clique which is a subgraph of 3 vertices.

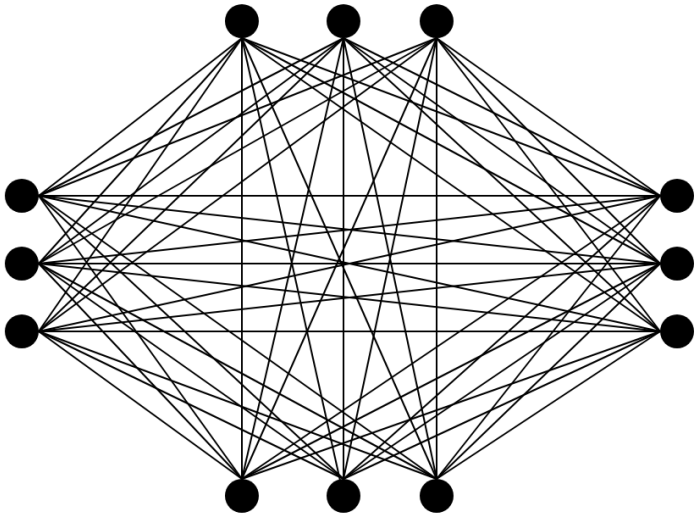


Figure 2: A complete 4-partite graph with 3 vertices on each partition.

it is important for researchers to be able to find sets of genes that may contribute to a particular trait or disease in very large datasets. But the problem of identifying and comparing genes, also called as the *Gene Cluster Discovery Problem*, is one of the key challenges faced by researchers. The Approximate Gene Cluster Discovery Problem was proven to be NP-hard in [11]. A problem H can be classified as NP-hard when

every problem L in NP can be reduced in polynomial time to H. By representing AGCDP as a clique-problem, these approximation algorithms can be used to produce candidate gene clusters.

In this paper, the detection of gene clusters is based from the transformation of the Approximate Gene Cluster Discovery Problem to a Minimum Weight t-Partite Clique Problem. The AGCDP is a combinatorial problem which detects genes that appear in similar segments across genomes [9]. The MWtCP will be used to generate the set of genes that are arranged close to each other across different genomes by finding the minimum weighted t-clique.

B. Statement of the Problem

Gene clusters can be used to map organisms to their ancestors and associate different species to one another. The discovery of gene clusters would help explain the complexities of the expression of traits [9] and would contribute greatly on genome studies [12]. Various techniques have already been presented in modelling gene clusters.

There are already many existing studies in the discovery of gene clusters. With various models of gene clusters available, such as common intervals and max-gap clusters classified either as permutations or strings, there are already existing tools utilizing these models to find gene clusters using either approximate or exact methods. But there is no existing tool in detecting gene clusters applying graph theory concepts, particularly using the clique-finding technique.

C. Objectives of the Study

The purpose of this study is to develop a software that detects gene clusters in a given dataset by computing for the minimum-weight t-partite clique of the constructed graph. The dataset of genomes is parsed as a CSV file through the software. The software converts the input to its integer representation. With this in-

put, a t-partite graph is constructed, where t is the number of genomes entered by the user. If the researcher has chosen to find approximate gene clusters using the 2-approximation algorithm, then the minimum weighted $(t-1)$-star that would approximate the minimum-weight t-partite clique in the graph is determined. If the researcher opted to find gene clusters using the exact method, then each combination of the vertices in the t-partite graph is identified, computing for the t-clique with the minimum weight. The researcher then specifies the necessary input parameters to be used. The software outputs the list of candidate gene clusters and the alignment of genes contained in each gene cluster across genomes. The functional specifications of the software are listed:

1. Allows the researcher to
    (a) Upload a dataset of genomes in CSV format which includes per row
        i. the species name
        ii. the genes of the given species given by its gene symbol
    (b) Specify the range for the size of the gene contents to be generated
    (c) Choose between finding the approximate gene clusters
        i. using the 2-approximation method
        ii. using the exact method without GPU
        iii. using the exact method with GPU
    (d) Specify the size of the gene cluster and the integer weights for the cost of additional and missing genes, if the researcher prefers to find the approximate gene clusters
        i. with the default value 1, or
        ii. with the user input value
    (e) Specify the number of candidate gene clusters to be displayed

2. The software is able to

   (a) Convert the genes from the dataset to its integer representation

   (b) Notify the researcher if a GPU device is installed

   (c) Display the organisms the researcher has uploaded

   (d) Display the integer representation of the input entered

   (e) Transform the dataset in integer form to its t-partite graph representation, where t is the number of genomes entered

   (f) Compute for the weights between each node in the graph

   (g) Find the minimum weighted t-clique from the constructed graph

   (h) Display the adjacency matrix generated

   (i) Display the graph representation generated

   (j) Display the list of candidate gene clusters and its alignment of genes in the gene cluster per genome

   (k) Export the summary of results, list of candidate gene clusters, and its alignment in PDF format

D. Significance of the Project

The development of this software can help speed up the process of identifying gene clusters to widen our knowledge on genetic information. The software also offers an optional method of finding gene clusters tailored for GPUs. The software would be using GPU resources to detect gene clusters in a faster rate. With this, the tool may serve as an aid to researchers on their discovery of gene clusters across genomes. By using the tool, the researchers are given candidate gene clusters in which they can use to start their more detailed analysis in the laboratory.

The detection of gene clusters is an essential process to better understand the relationship between genomes which is significant in describing the functional genetic information of species and its similarities with other organisms. The findings produced from these studies will further improve knowledge about gene functions, its contribution to certain traits and gene expressions, and its effects to diseases [13].

E. Scope and Limitations

The software used in this study have the following constraints:

1. The input format is limited to the *scientific name of the specie*, *gene symbol 1*, *gene symbol 2*, *gene symbol 3*, ..., *gene symbol n*, where n is the number of genes in the genome of the species.

2. The tool does not check if the user has entered a correctly annotated dataset.

3. The input values on the size of the gene cluster and the integer weights is correct and is subject to the researcher.

4. The 2-approximation algorithm used in the software for finding approximate gene clusters only applies for the metric case of the minimum weight t-partite clique problem. The approximability of a non-metric dataset cannot be determined.

5. The exact algorithm can only accomodate up to two genomes.

6. The software uses the max-gap clusters in permutations, thus duplicated genes in the genome are not analyzed.

7. The interpretation on the gene clusters is not provided by the tool.

II. Review of Related Literature

Gene clusters are regions of DNA sequences descending from a common ancestor across related genomes. Gene clusters are useful in constructing phylogenetic and ancestral trees [14]. The detection of gene clusters also helps identify gene functions and establish relationships between species [15].

Various techniques in modelling gene clusters are available. Bergeron et al. [6] listed models along with its algorithms in detecting gene clusters. Gene clusters can be modelled either as permutations, where gene clusters are assumed to have the same gene content for each genome, or strings, where gene duplication and repetitions are allowed [10]. The model of common intervals focus on finding "exact" gene clusters. Max-gap clusters "approximate" gene clusters and allow gaps within clusters [7].

But finding a universal model in defining gene clusters has been a challenge for many years [6]. Many algorithms do not represent the real biological events and its analysis can be computationally difficult [7]. Gene clusters are products of multiple biological mechanisms, making the formulation of models to capture the biological reality even harder. Groups of genes remain in close proximity with each other due to functional pressure or theme along the genomes [16]. Others suggest that these genes are closely related as they are part of a larger biological network. Smon and Duret [17] proposed that genes, more specifically, housekeeping genes, are clustered together due to co-expression.

As complete sequences of genomes become more readily available, various algorithms and systems have been developed to address the gene cluster detection problem. Most of the applications developed were used to compare two species only but newer systems can process multiple genomes [18].

Gecko 3 is an open-source software which uses an exact method in detecting gene clusters given the specified parameters. Gecko 3 models genes as numbers where homologous genes across different species are represented as the same number. Given

the *cluster size threshold* s or the minimum number of genes in the gene cluster, *distance threshold* δ or the maximum number of additional and missing genes in the cluster, and the *quorum parameter* k' or the minimum number of genomes which contains the cluster, the algorithm used finds the reference gene clusters in all genomes. The reference gene cluster is defined as a set of genes C with $|C| \geq s$ such that C has an exact occurrence in one of the genomes and C has a distance of at most δ to in at least $k' - 1$ other genomes. The p-value is calculated for each reference gene cluster is computed to determine its significance [18].

The program searched for gene clusters in the dataset of 678 bacterial genomes with Synechocystis sp. PCC 6308 as a reference genome. It was used to compare against 677 other bacteria acquired from the STRING database. Gecko 3 was able to predict 587 gene clusters [18].

CYNTENATOR is another progressive gene order alignment software used to detect regions in the genome with conserved synteny across large, diverging set of organisms. Conserved synteny occurs when genes located in the same chromosome have preserved their gene content and gene order in two or more related species [19].

The chromosomes are represented as sequences of genes. Homologous genes are determined from the bitscores results from all vs. all BLASTP searches. The software runs at $O(n^3)$ time complexity and $O(n^2)$ space complexity, where n is the number of genes in a genome. CYNTENATOR tested their simulated dataset of vertebrate-sized gene orders using pairwise gene-order comparison [19].

The Max-gap Clusters by Multiple Sequence Comparison (MCMuSec) was developed using the "gene teams" model from [5]. Genomes are represented as sequences of genes and gene teams are sets of genes in close proximity with each other, allowing them to appear in different order [20]. Gene teams are also called as *max-gap clusters in permutation.*

The algorithm used in the software performs the *decompose* and *filter* procedures.

For example, given genomes G_1 and G_2, genes from both genomes are matched, regardless of their order. Subsets are identified from the matched genes and non-homologous genes are removed. Then, the algorithm computes for the signature $SIG(S_m^{d_m}) = \{sig_1, sig_2, \ldots, sig_{NUM}\}$ for each subset identified, where each sig_i consists of t genes from $S_m^{d_m}$. Common signatures from the combinations of subsests are determined. The *decompose-filter* procedure is applied recursively until all subsets produced are less than the specified *minsize* [20].

The software was tested using a large dataset of 133 bacterial genomes [20]. Homologous genes are determined based on COG (Clusters of Orthologous Groups). 32, 825 gene clusters were found; 370 of them are statistically significant under the p-value < 0.05) and occurred at $E.coliK12$.

III. Theoretical Framework

A. Gene

A gene is the basic unit of heredity. It contains a set of instructions for coding a particular protein. In this paper, a gene is represented by a positive integer g [9]. The set of all unique genes is called the *gene universe* and is denoted by $\mathcal{U} = \{0, 1, 2, ..., N\}$ for some integer $N \geq 1$ [7].

Genes may be related to each other by descent from a common ancestor. Homologs are genes that have evolved from a common ancestral DNA sequence and generally still perform the same function. Orthologous genes are genes that have evolved from a common ancestor across genomes through speciation. Paralogous genes are results of gene duplication within genomes. Genes with non existing homologs are represented by the integer 0, as they are not of interest in the problem [9].

B. Gene Symbols

Gene terminology, which includes both gene names and gene symbols, is used to identify genes based on certain standards. Genus-specific research groups have followed their own standards for naming genes, such as *Saccharomyces cerevisiae*, *Drosophilia melanogaster*, and *Mus muculus*. The nomenclature for human genes adapt the standards of the HUGO Gene Nomenclature Committee. Designating gene symbols and gene names is essential for unifying gene representation across various genomes.

C. Genome

A genome is the complete set of genetic material of an organism. It is represented as a sequence of integers $g^i = (g^i_1, g^i_2, \ldots, g^i_{n_i})$ where n_i is the length of the i^{th} genome, and g^i_j is the jth gene in the ith genome [9].

A *linear interval* J^i in a genome g^i is an index set which can either be empty $J = 0$ or $J = \{j, j+1, ..., k\}$, written as $J^i_{j_i,k_i}$, where $1 \leq j_i \leq k_i \leq n$. The *gene content* $G_J = \{g_j, \ldots, g_k\}$ of a linear interval $J^i_{j,k}$ in genome g^i is the set of unique genes contained in that interval [9].

D. Gene Cluster

A gene cluster is a set of closely-related genes which are arranged in close proximity with each other, even after genome sequences have evolved in multiple events such as gene duplication and gene loss. Generally, these groups of genes perform related functions and encode for similar proteins across different genomes. The size of a gene cluster refers to the number of genes in the cluster and it varies across different species [21].

E. Formal Models of Gene Clustering

Various models have already been presented for gene clusters. Heber and Stoye [22] modelled gene clusters as common intervals in k permutations. The concept of gene teams and max-gap clusters were discussed in [5] and [23].

In some models, a fixed set S of genes can be represented into two types: *permutations* and *strings*. Permutations assume exactly the same gene content [7]. Hence, given a genome g^i, the genes of set S must have an occurrence in genome g^i, without repetition or other genes. For example, given a genome $g^1 = \{1, 2, 3, 4, 5\}$, its permutations include $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 5, 3\}$, and $S_3 = \{3, 4, 1, 2\}$.

Unlike permutations, strings allow gene duplication and the existence of other genes in the set S [7]. From the given genome g^1 defined above, some examples of strings are $S_4 = \{1, 4, 4, 2\}$, $S_5 = \{1, 2, 2, 3, 4, 6\}$, and $S_6 = \{2, 2, 8, 8\}$.

The discussion of some gene clustering models are presented below.

1. Conserved Segments

Bergeron et al. [6] defined conserved segments as a simple model such that gene clusters must be sets of genes with the same order and orientation across various genomes. Given two sets of genes $S_1 = \{g_1, g_2, g_3, \ldots, g_k\}$ and $S_2 = \{g_1, g_2, g_3, \ldots, g_k\}$ with length k, S_1 and S_2 are equal if (1) $g_i = h_i$ or (2) $g_i = -h_{k_i+1}$, $\forall\ i$ [6].

Definition 1. *Let G be a set of signed permutations on the set of genes S. A subset of S is a conserved segment if it has an occurrence in each permutation of G, without gaps, and all occurrences are equal.*

Conserved segments acknowledge the notion that an inverted sequence is exactly the same as the original. The model also limits gene clusters to have an occurrence in different genomes in the same order and orientation. For example, given two genomes $g^1 = \{-1, 2, 3, 4, -5\}$ and $g^2 = \{5, -4, 1, 2, 3\}$, the sets $S_1 = \{2, 3\}$ and $S_2 = \{4, 5\}$ are some instances of conserved segments. Singletons are considered trivial conserved segments [6].

2. Common Intervals

A common interval is a model of clustering genes where the genes in the set are consecutive, regardless of its order. It is a first generalization of conserved segments where the order and the orientation of genes are relaxed. Thus, the signs of the genes are disregarded. The model of common intervals can be represented either as *permutations*, or *strings* [6].

Common intervals in permutations can be formally defined as:

Definition 2. *Let G be a set of permutations on the set of genes S. A subset of S is a common interval if it has an occurrence in each permutation of G, without gaps.*

Common intervals in strings can be formally defined as:

Definition 3. *Let G be a set of strings on the set of genes S. A subset of S is a common interval if it has an occurrence in each string of G without gaps. Common intervals in strings allow gene duplications.*

Unlike common intervals in permutations, common intervals in strings allow repetition [10] and are used in analyzing genomes with duplicates [6]. However, gaps are not allowed in both models.

3. Max-gap Clusters

Max-gap clusters is another model for detecting gene clusters. It is a model of approximate gene clusters [7] and is defined as a set of genes where the gap between the genes in each genome is less than or equal to the given threshold, δ. If $\delta = 0$, then gaps are not allowed and the model is reduced to a common interval [23]. Max-gap clusters can also be represented as permutations or strings.

Max-gap clusters in permutations, also referred to as *gene teams* [5], can be formally defined as:

Definition 4. *Let G be a set of strings on the set of genes $S \cup \{*\}$, such that each string is a permutation of S when the symbols $\{*\}$ are removed. Let $\delta \geq 0$ be a fixed integer. A subset of S is a gene team if it has an occurrence with maximum gap size δ in each permutation of G, and has no extension.*

Max-gap clusters in strings can be formally defined as:

Definition 5. *Let G be a set of strings on the set of genes S, and $\delta \geq 0$ be a fixed integer. A subset of S is a max-gap cluster if it has an occurrence with maximum gap size δ in at least one string of G, and has no extension.*

The model of max-gap clusters in strings generalizes the models of gene clusters mentioned above. It allows gaps, repetition and duplication of genes. However, the

running time of finding gene clusters in large datasets of genomes using this model can be very long and inefficient [6].

4. r-windows

The r-windows model is a gene cluster model where r is defined as the length of a region in the genomes under consideration wherein they share at least k genes [23].

For example, let genomes g^1 and g^2 be

$$g^1 = \{1, *, 2, *, 3, 4, *, *, 5, 6, 7, 8, 9\}$$

$$g^2 = \{*, 3, *, 1, 4, *, 2, 5, 6, 7, *, 9, 8\}$$

Plotting a dot plot using the r-windows model with $r = 5$ and $k = 4$, the clusters $\{5, 6, 7, 9\}$ (shown as the dotted box) and $\{6, 7, 8, 9\}$ (shown as the solid box) can be attained [23].

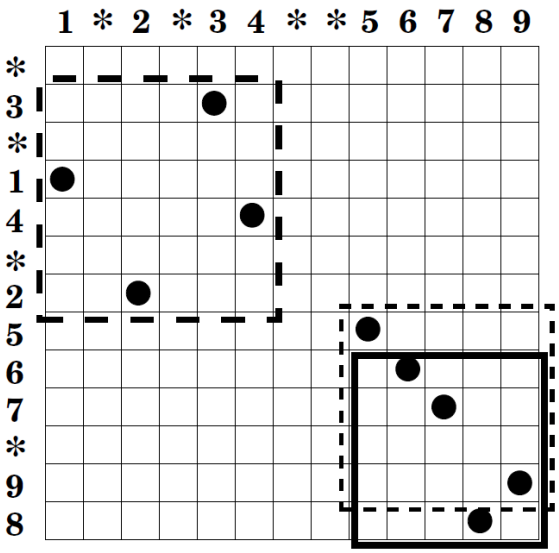


Figure 3: A dot plot with G_1 as columns and G_2 as rows. The filled dots represent homologous genes.

F. Graph Theory

A *clique* is a subgraph G' from an undirected graph G, such that every two distinct vertices are adjacent. The induced subgraph G' is a complete graph. The size of the clique G' is the number of vertices in G' [8].

A *bipartite graph* is an undirected graph $G = (V, E)$ such that V can be partitioned into two sets V_1 and V_2 where either $u \in V_1$ and $v \in V_2$, or $u \in V_2$ and $v \in V_1$. No two vertices within the same set V_1 or V_2 are adjacent [8].

The bipartite graph can be extended to having t partitions. A *t-partite graph* is an undirected graph such that V can be partitioned into t sets and there are no adjacent vertices within the same set. A *complete t-partite graph* is an undirected graph such that every pair of graph vertices in the t sets are adjacent.

A *star graph* S_n of order n is a tree of n vertices such that one vertex has a degree of $n - 1$ and the other vertices with degree 1 [24].

G. Approximate Gene Cluster Discovery Problem

Rahmann and Klau [7] formulated the problem of finding gene clusters as the *Approximate Gene Cluster Discovery Problem (AGCDP)*. AGCDP is presented as a combinatorial problem in detecting genes that are in close proximity with each other. Its formal definition is stated below.

Definition 6. *Given the gene universe $U = \{0, 1, \ldots, N\}$, set of genomes $G = \{g^1, g^2, \ldots, g^t\}$, a size range $[D^-, D^+]$ or positive constant D for the reference gene set, and integer weights w^- for the cost of missing genes and w^+ for the cost of additional genes, find $X \subset U$ with $0 \notin X$, $D^- \leq |X| \leq D^+$ or $|X| = D$, and a set of linear intervals $J = \{J^i_{j_i,k_i}\}$, $\forall i$ with the minimum cost function*

$$cost(X, J) = \sum_{i=1}^{t} [(w^- \cdot |X \backslash G^i_{j_i,k_i}) + (w^+ \cdot |G^i_{j_i,k_i} \backslash X)]$$

AGCDP aims to find the a gene set $X \subset U$ without non existing homologs, represented as 0, and a linear interval J_i for each genome under consideration such that X is roughly equal to the gene content of J_i in the i^{th} genome. Integer weights w^- and w^+ are introduced as well to account for the cost of missing and additional genes, respectively.

In [9], the AGCDP was transformed into a minimization problem of graph G_{AGCDP}. The transformation of the AGCDP to a graph problem and its minimization is presented below.

1. Graph Construction

Given gene universe $U = \{0, 1, \ldots, N\}$, a set of genomes $G = \{g^1, g^2, \ldots, g^t\}$, a size range $[D^-, D^+]$ or positive constant D for the reference gene set, and integer weights w^- for the cost of missing genes and w^+ for the cost of additional genes, the proposed transformation of AGCDP input parameters into a t-partite edge-weighted undirected graph representation is defined below [9].

Definition 7. *Define a t-partite edge-weighted undirected graph $G_{AGCDP} = (V, E)$, where*

$$V = \bigcup_{i=1}^{t} V_i$$

1. *A part $V_i \subset V$ represents a genome $G^i \in G$ where $1 \leq i \leq t$.*

2. *A function $v(\cdot)$ is a bijection $v : G^i \rightarrow V_i$ from the set of all gene contents $G^i = \{G(J^i_{j_i,k_i})\}$ in genome g_i to the set of all vertices in $V_i \forall i, 1 \leq i \leq t$. An evaluation of $v(G(J^i_{j_i,k_i}))$ is a vertex in graph G_{AGCDP} and an evaluation of $v(x_i)$ is a gene content of a linear interval in genome g_i.*

3. *An edge $e \in E$ is incident to vertices $x \in V_x$ and $y \in V_y$ if an only if $x \neq y$.*

4. *The weight $w_{x,y}$ assigned to an edge incident to vertices x and y is equal to*

$$w_{x,y} = w^- \cdot |v(x) \backslash v(y)| + w^+ \cdot |v(y) \backslash v(x)|$$

where the "\" is the set diffference operator.

The set difference between two distinct gene contents $G(J^i_{j,k})$ and $G(J^p_{j,k})$, $1 \leq i \leq t$, $1 \leq p \leq t$ and $i \neq p$, is defined as

$$G(J^i_{j_i,k_i}) \backslash G(J^p_{j_p,k_p}) = \{g | g \in G(J^i_{j_i,k_i}) \text{ and } g \notin G(J^p_{j_p,k_p})\}$$

Definition 8. *Given a vertex x in G_{AGCDP}, let star(x) be a set of vertices of size (t-1) such that x_i is rooted at $x \in V_u$, where*

$$x_i = argmin_{x_i} w_{x_i,x} \forall x_i \in V_i, i \neq u$$

To compute for the weight of the star, simply add the weights of the vertices included in the $star(x)$ as shown below

$$weight(star(x)) = \sum_{i=1}^{t} w_{x_i,x}, i \neq u$$

2. Graph Minimization

After transforming the input parameters of AGCDP into its graph representation, the minimization criteria for G_{AGCDP} is presented below [9].

Definition 9. *Given a graph $G_{AGCDP} = (V, E)$ and a size range $[D^-, D^+]$ or positive constant D, find a vertex $x \in V_u$ and star(x) such that $|v(x)| = D$ or $|v(x)|$ is in the range $[D^-, D^+]$ and*

$$weight(star(x)) = \sum_{i=1}^{t} w_{x_i,x}, i \neq u$$

is ***minimum.***

The goal is to find a vertex $x \in V_u$ such that the length of its gene content is equal to the specified constant D or is inside the range $[D^-, D^+]$ and star with its base vertex X such that the total weight of the vertices in the star is the minimum [9].

H. Minimum Weight t-Partite Clique Problem

Let $G = (V, E, w)$ be an edge-weighted, t-partite complete graph. Each part in V is denoted by U_i, such that $|U_i| = m$ for all $1 \leq i \leq t$. Since graph G is a complete t-partite graph, each pair of vertices from two distinct parts has an edge with a non-negative weight [10].

From the definition of graph G, the total number of vertices is $n = \sum_{i=1}^{t} m_i$, where m_i is the number of vertices in the i^{th} partition.

Definition 10. *Given a complete edge-weighted t-partite graph, the Minimum Weighted t-partite Clique Problem (MWtCP) is the problem of finding a t-clique with minimum weight.*

If all edge weights satisfy the conditions of the triangle inequality, then the MWtCP is classified in the *metric* case.

The 2-Approximation Algorithm for the Metric Case of Minimum Weight t-Partite Clique problem approximates the Minimum Weight t-Partite Clique problem by finding the $(t-1)$-star S_{t-1} with the minimum weight.

The input is a complete weighted t-partite graph $G' = (V'\{v_j^i | 1 \leq j \leq t, 1 \leq 1 \leq m\}, E', w)$ where $\{V_1', ..., V_t'\}$ are the t partitions of V' and $V_j' = \{v_j^i | 1 \leq i \leq m\}$, $1 \leq j \leq t$ where m is the number of vertices per partition. The final output is a t-clique with the minimum weight [10].

Figure 4 shows a complete weighted t-partite graph where $t = 4$ and $m = 5$. The algorithm runs for every vertex in each partition, finding the vertex with the minimum weight. In Figure 5, the algorithm was able to find the minimum weight at v_{22} in partition 2. The algorithm runs until it is able to form a $(t-1)$-star, with each vertex having the minimum weight representing a particular partition. Figure 6 shows that the algorithm was able to generate a star rooted at v_{11} in partition 1 with v_{22} as the minimum in partition 2, v_{35} as the minimum in partition 3, and v_{42} having the least weight in partition 4.

The algorithm runs until all $\sum_{i=1}^{t} = m_i$ vertices have been considered. After getting all star graphs, the star graph S' with the minimum weight is considered. The weight of the star is computed by adding the weights of all the vertices included in the graph. From the star graph with the minimum weight, the t-clique is derived from all the vertices of S'. The pseudocode for the 2-approximation algorithm is shown below.

The running time of the algorithm is $O(n^2)$ for a complete weighted t-partite graph of n vertices [10]. The proof that the algorithm is a 2-approximation algorithm for the metric case is shown in [10]. A 2-approximation algorithm limits the solution to a cost of at most twice the optimal.

The proof that MWtCP returned approximate gene clusters was proven in [10]. The algorithm will work for gene clusters modelled either as common intervals or max-gap clusters. The following characteristics of the graph were observed from the 4 cases [10]:

1. Common Intervals in Permutations:
    - nodes have the same length
    - nodes are unique per partition
2. Common Intervals in Strings:

Figure 4: A complete weighted 4-partite graph.

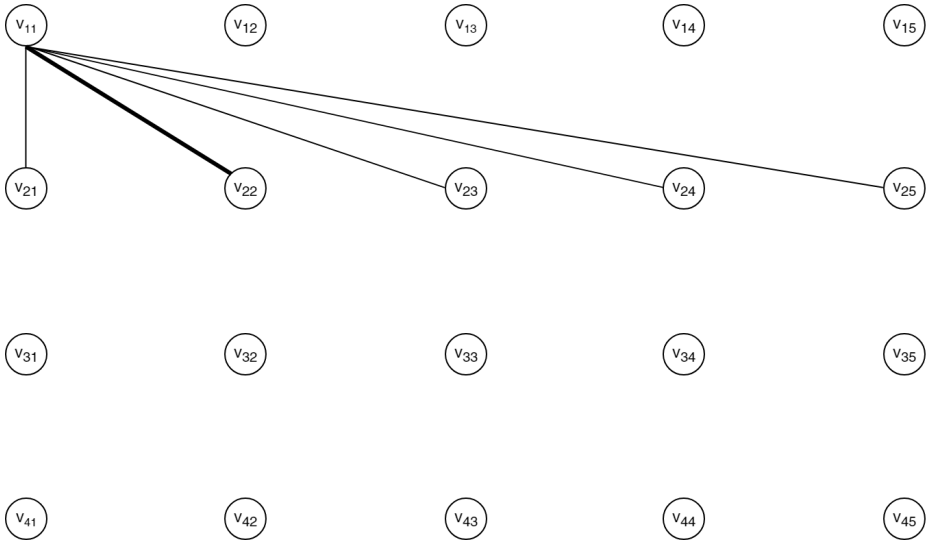


Figure 5: The first vertex selects the vertex in partition 2 with the minimum weight.

- nodes do not have the same length
- nodes are not unique per partition

3. Max-gaps in Permutations:

   - nodes do not have the same length
   - nodes are unique per partition

4. Max-gaps in Strings:

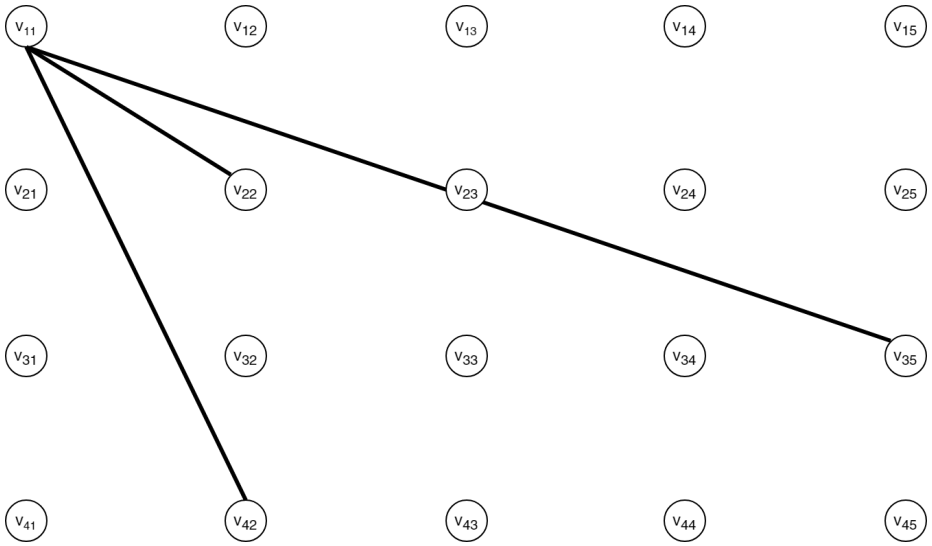


Figure 6: The algorithm forms a $(t-1)$-star from the vertices with minimum weight per partition rooted at the first vertex.

Algorithm 1 Minimum Weighted t-1 Star Algorithm for MINIMUM WEIGHT t-PARTITE CLIQUE problem

$n \leftarrow |V'|$
$m \leftarrow |V'_1|$
$min_{star} \leftarrow \emptyset$
$min_{weight} \leftarrow \infty$
for each vertex $v_j^i \in V'$ **do**
    $tmp_{star} \leftarrow \emptyset$
    $tmp_{weight} \leftarrow 0$
    for each $1 \leq x \neq j \leq t$ **do**
        $min_{edge-weight} \leftarrow \infty$
        $min_{edge-neighbor} \leftarrow v_j^i$
        for each vertex $v_x^y \in V'$ where $1 \leq y \neq \leq n$ **do**
            if $w(v_j^i, v_x^y) < min_{edge-weight}$ **then**
                $min_{edge-neighbor} \leftarrow v_x^y$
                $min_{edge-weight} \leftarrow w(v_j^i, v_x^y)$
        $tmp_{star} \leftarrow tmp_{star} \cup \{min_{edge-neighbor}\}$
        $tmp_{weight} \leftarrow tmp_{weight} + min_{edge-weight}$
    if $tmp_{weight} < min_{weight}$ **then**
        $min_{star} \leftarrow tmp_{star}$
return min_{star}

- nodes do not have the same length
- nodes are not unique per partition

IV. Design and Implementation

A. Data Specifications

The user inputs the dataset of genes per genome by entering a file that strictly follows the specified format:

1. The format of the file is .csv.
2. A row in the file corresponds to one genome.
3. The first column specifies the scientific name of the organism.
4. The rest of the columns indicate the genes in the genome.
5. Homologous genes are given the same name throughout the given genomes.
6. It does not contain any missing data.

B. System Design

The software is implemented as defined in the diagrams presented below. In general, the software's input and output are presented in the context diagram on Figure 7.

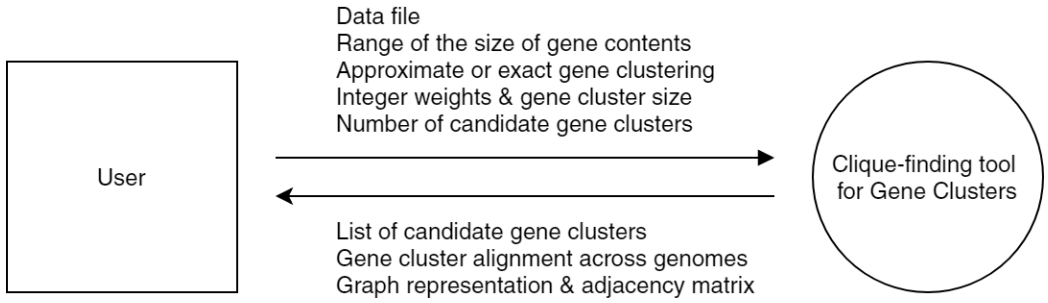


Figure 7: Context diagram

The tool accepts genomic data from the researcher through the input file and prompts the researcher to input the range of the size of gene contents. The researcher

can choose between finding approximate gene clusters or exact gene clusters. The researcher then specifies the input parameters needed for gene clustering, the size of the gene cluster and the integer weights for the cost of additional and missing genes. The researcher also specifies the number of candidate genes to be displayed. The software outputs the list of candidate gene clusters and compares the alignment of genes, including missing and additional genes, per species. The software also enables the researcher to export the alignment of genes in a PDF file. The tool allows the user to view the graph representation and adjacency matrix of the generated graph.

The use case diagram presents the functionalities in the tool that the researcher can use. The researcher enters the dataset of genes as input and the size interval for the gene contents. Then, the researcher can view the list of input genomes in its integer form. The researcher chooses between finding approximate gene clusters or exact gene clusters. The researcher enters the specified input parameters and the number of candidate gene clusters to be displayed. After processing the data and computing for the gene clusters, the researcher can view the list of candidate gene clusters. The graph representation, adjacency matrix, and alignment of genes present in the gene cluster across different genomes are also available to the researcher. The researcher can also export the list of candidate gene clusters and their alignment of genes in PDF format.

The flowchart diagram displays the processes involved in the tool. The researcher starts by providing the dataset of genes and the size interval for the gene contents. The dataset consists of genomic data such that each row pertains to the genes of one genome and the first column specifies the scientific name of the organism. After entering the dataset, the tool parses the CSV file and creates a matrix of genes. The contents of the matrix is then converted to its integer representation such that genes with the same name are given the same integer as well. After the process of integer representation, the researcher is shown a preview of the list of organisms being

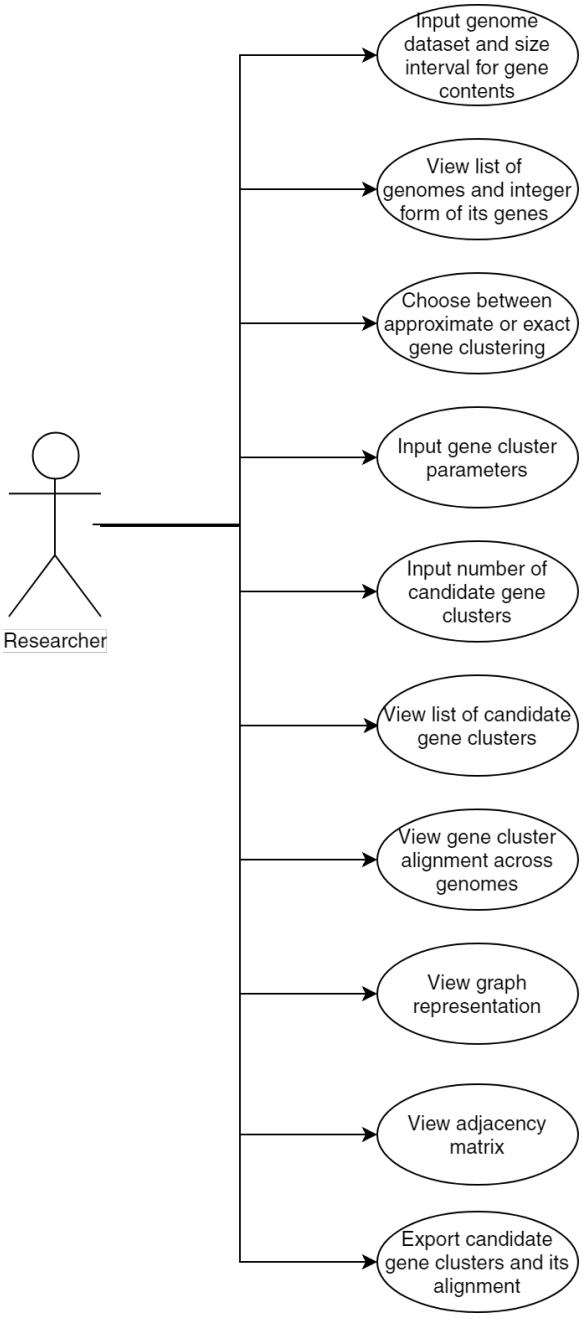


Figure 8: Use-case diagram

processed and its integers.

Then, the researcher chooses if approximate gene clusters or exact gene clusters are going to be determined. If the user chooses to find approximate gene clusters, the user is asked for the integer weights accounting for missing and additional genes and the size of the gene cluster. The researcher also specifies the number of candidate

gene clusters to be displayed. Then, the software constructs the graph and computes for the minimum weighted t-clique. The researcher then views the list of candidate genes. The graph representation, adjacency matrix of weights, and alignment of genes in the gene cluster across the different genomes are also presented.

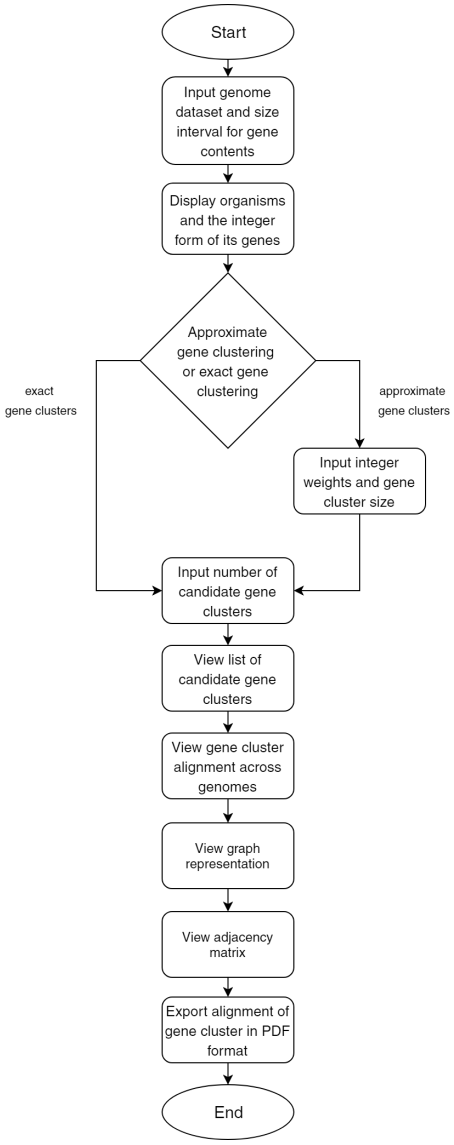


Figure 9: Flowchart diagram

C. System Architecture

The gene cluster approximation software is implemented using Java and will run on a machine that has the Java Runtime Environment installed. If the researcher wishes to use the exact gene clustering method using GPU, the machine should have at least CUDA 8.0 installed.

D. Technical Architecture

For the tool to run smoothly, it is recommended that the researcher runs the tool in a machine with 8 GB RAM. If the researcher intends to find gene clusters using the exact gene clustering method, a machine with a GeForce GTX 1070 graphics card is recommended.

V. Results

The home window appears after running the application. As shown in Figure 10 below, the home window contains the start and help buttons to guide the user on how and where to start detecting gene clusters.

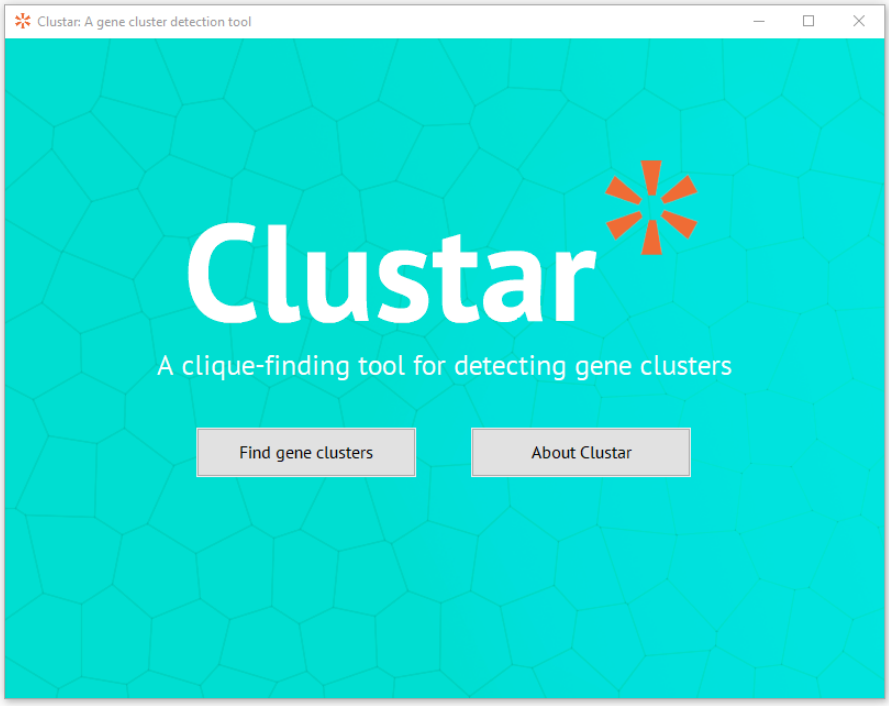


Figure 10: The window for the main menu

Clicking on the *About Clustar* button will show a user guide on the necessary input and output of the tool, which is shown in Figure 11. The user guide shows the correct format of the dataset to be uploaded. The user guide also shows a brief description of each input instance and shows the various ways of viewing the results. If the user has understood the correct input and output format, the user can go back to the main window to start the procedure of detecting gene clusters.

Clicking the *Find gene clusters* button will show a window which allows the user to choose and upload a dataset. The user is also prompted to specify the range for the size of the gene contents. Its corresponding window is shown on Figure 12.

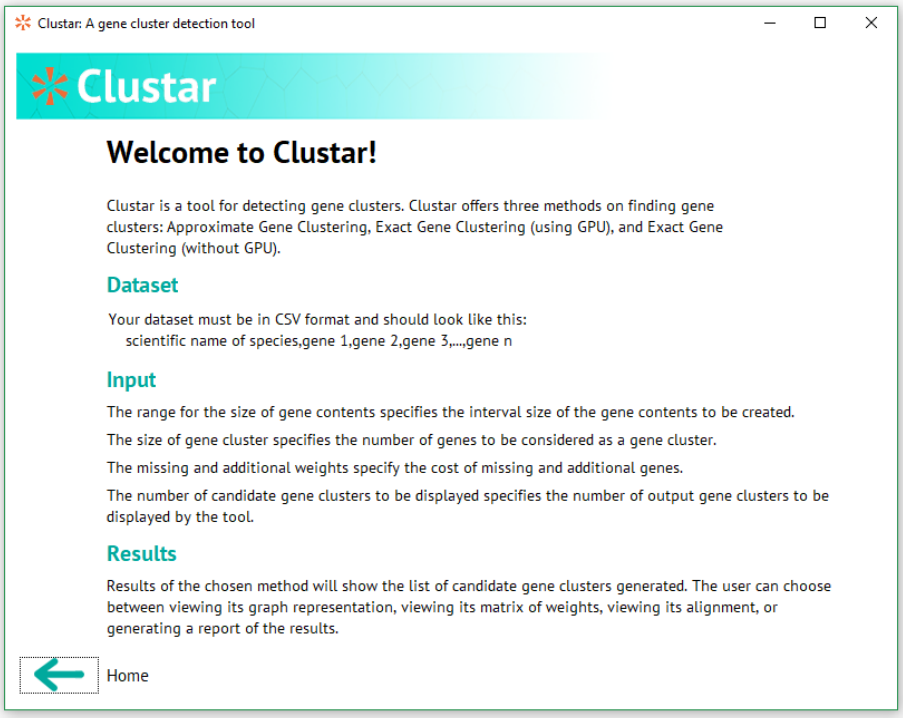


Figure 11: The window for the user guide

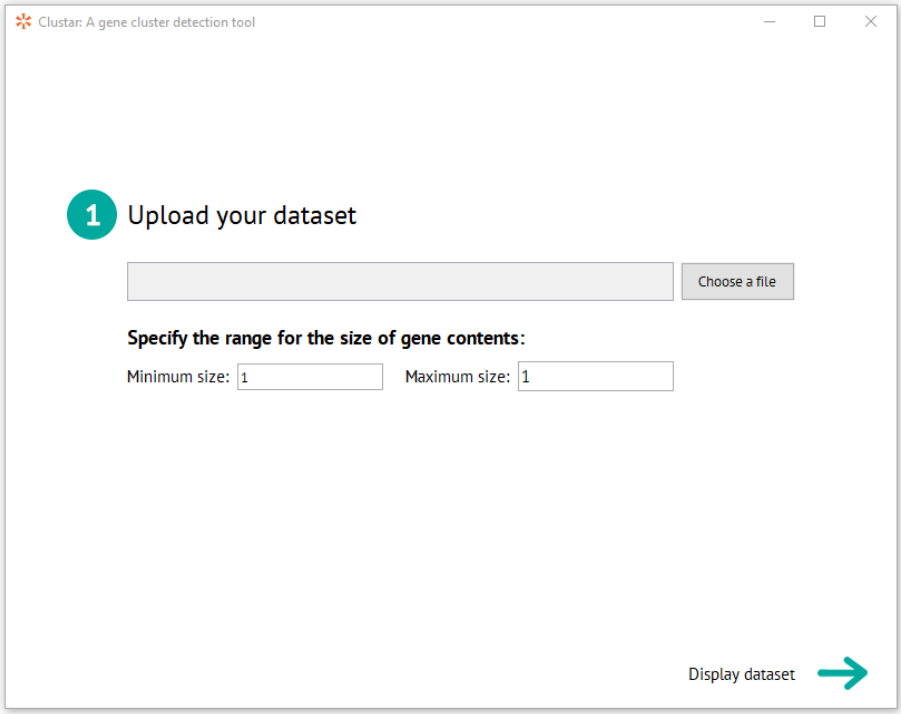


Figure 12: The window for the uploading a dataset

A file chooser will appear once the user clicks on the *Upload* button as shown in Figure 13. Then, the user can click on the arrow on the bottom right section of the window to proceed to the next step.

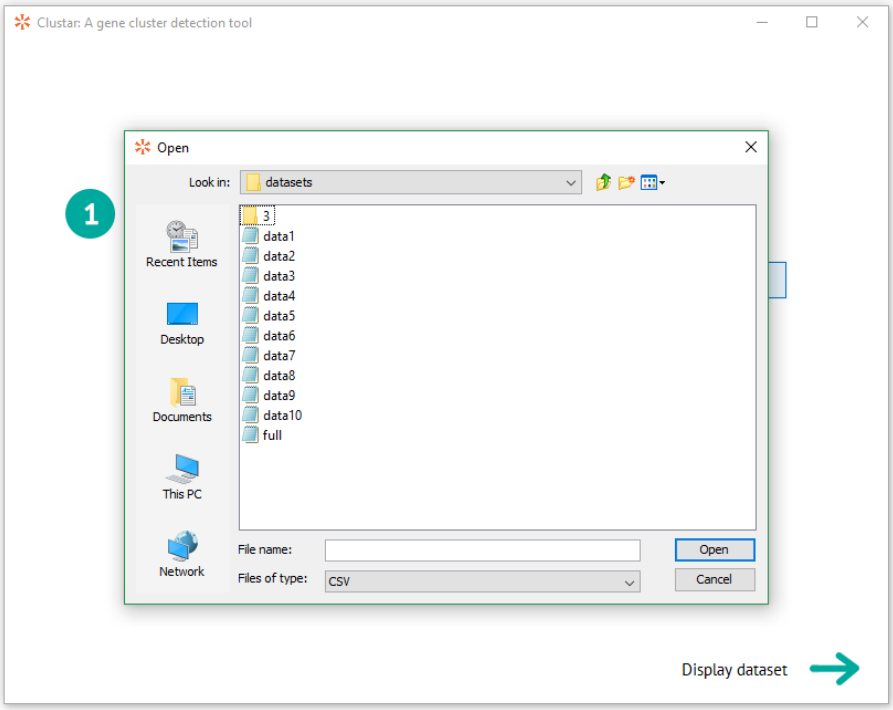


Figure 13: A file chooser pops up to guide the user on what dataset to upload

A progress bar will appear to show that the dataset the user has uploaded is being processed. Figure 14 shows this.

Once the dataset has been preprocessed, a window displaying the genomes uploaded and the integer representation of each gene will appear, as shown in Figure 15. The user can select between genomes on the drop down box to view its contents. The user can click on the arrow on the bottom right section of the window to proceed to the next step.

After displaying the data, the user is now able to choose the type of gene clustering method to use in detecting gene clusters. A note is also shown to inform the user if the machine currently used has a GPU device connected. If a GPU device is detected, the user can choose between all three options: Approximate Gene Clustering, Exact

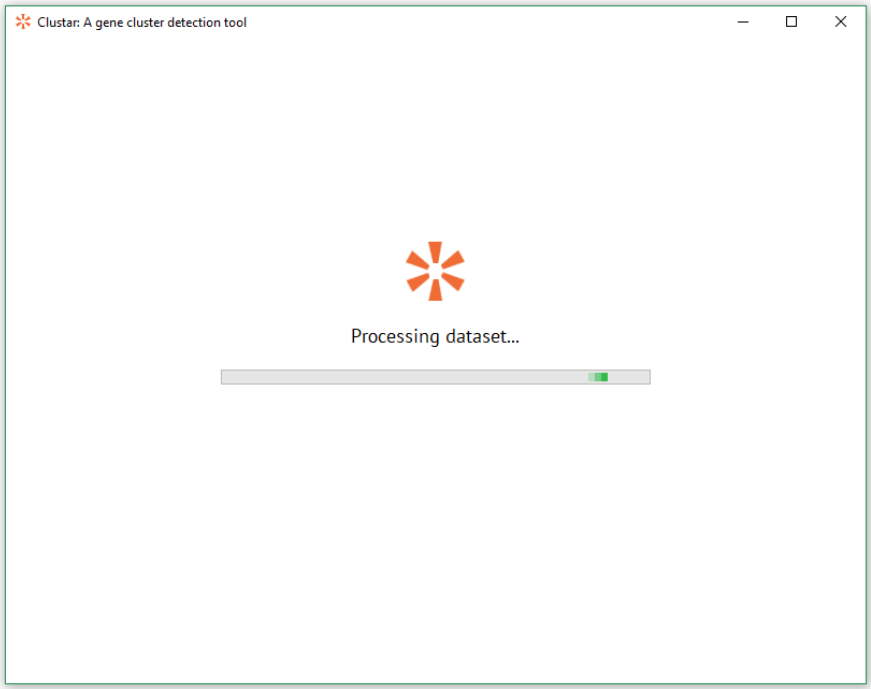


Figure 14: The progress bar for uploading a dataset

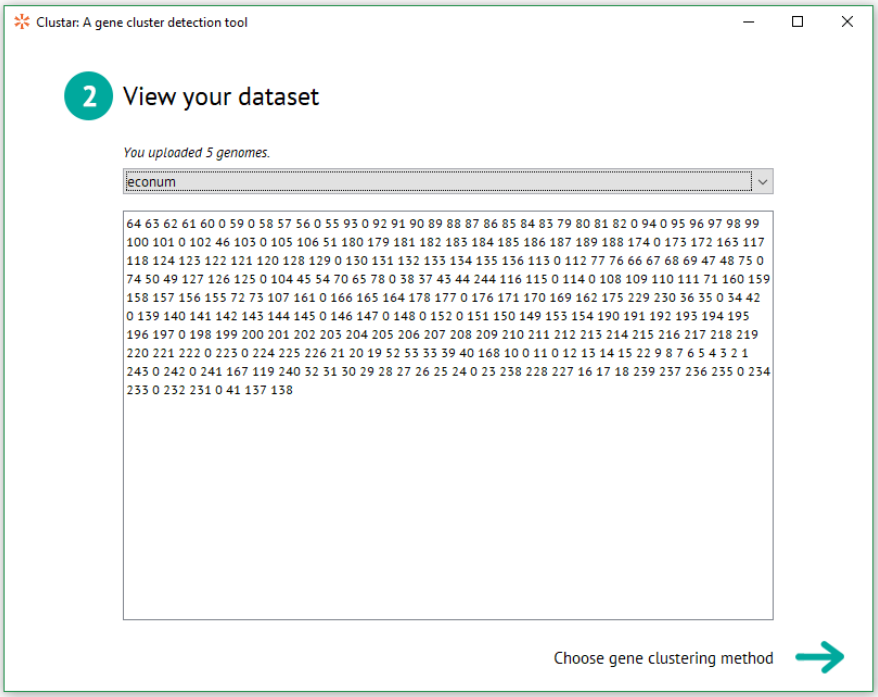


Figure 15: The window for displaying the integer representation of the dataset

Gene Clustering (using GPU), and Exact Gene Clustering (without GPU). If the application was not able to detect a GPU device, the option for Exact Gene Clustering (using GPU) will be disabled. These features are shown in Figure 16.

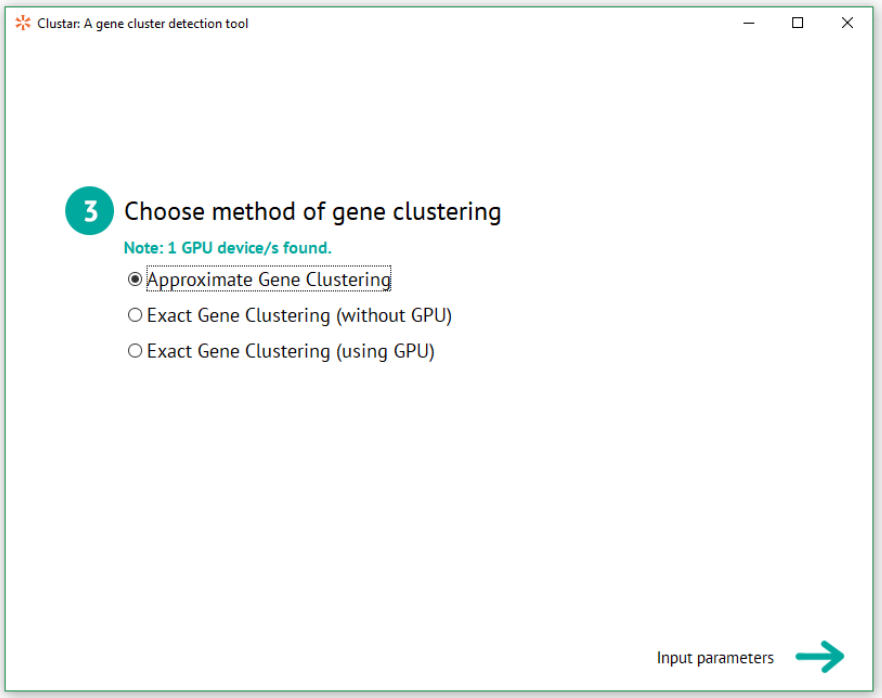


Figure 16: The window for choosing the method to be used

Proceeding to the next step by clicking the arrow below, a window displaying the input parameters needed will appear as shown in Figure 17. Choosing the option *approximate gene clustering* will yield text boxes where the user can input their preferred values. These input boxes include the size of the gene cluster, weights of missing and additional genes, and the number of candidate gene clusters to be displayed. Choosing any of the *exact gene clustering* options will only show one text box expecting values for the number of candidate gene clusters to be displayed as shown in Figure 18.

Clicking on the arrow below, a progress bar will again appear to show the user that the tool is now building the graph representation, calculating the weights of each star, and finding the cliques with the least weight. Figure 19 shows the window of

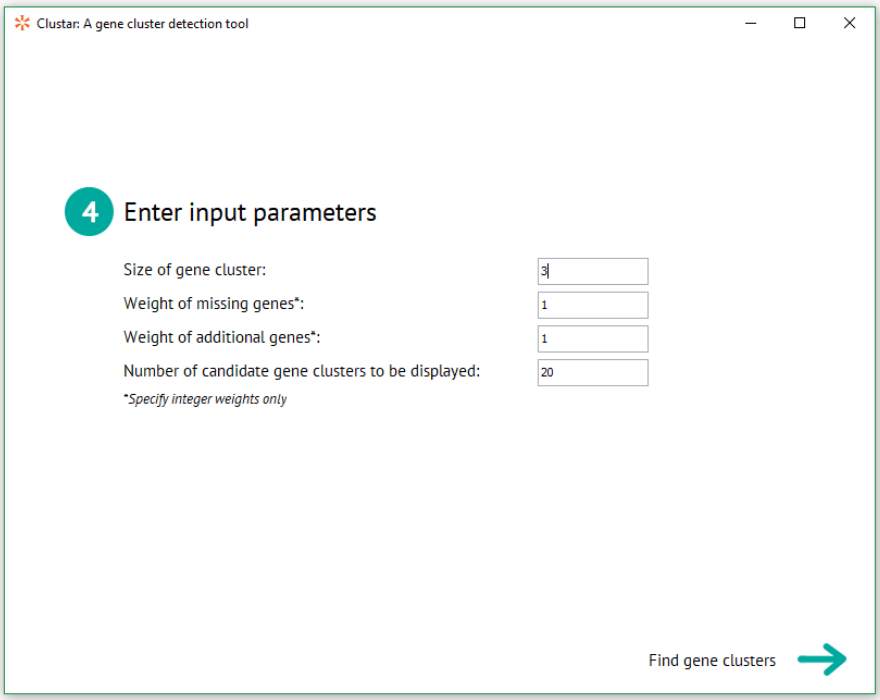


Figure 17: The window for entering input parameters for approximate gene clustering

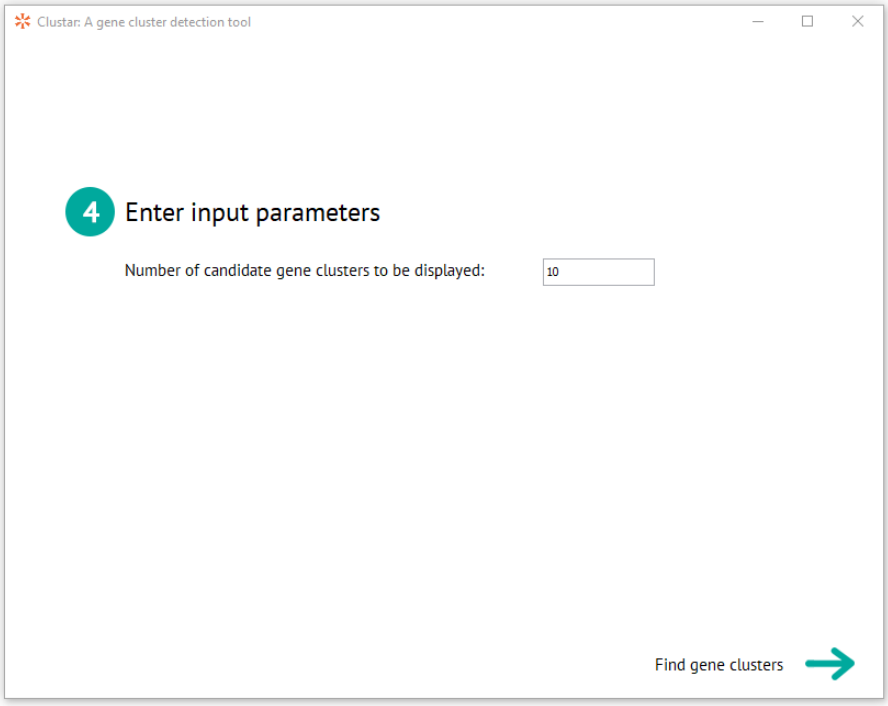


Figure 18: The window for entering input parameters for exact gene clustering

the progress bar.

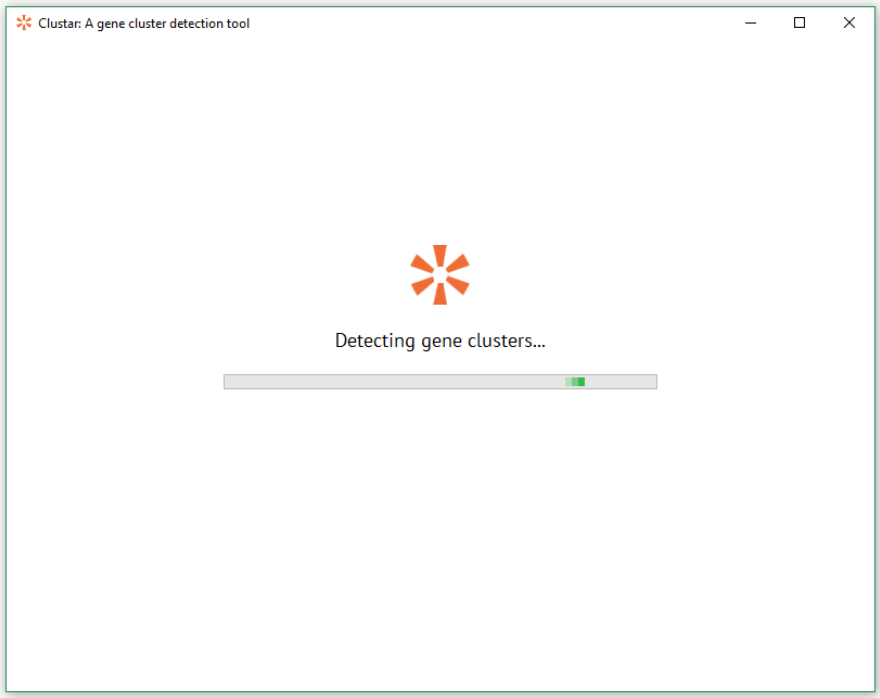


Figure 19: The progress bar for running the algorithm and detecting gene clusters

Once the tool is finished, a results window will appear containing the list of candidate gene clusters generated as shown in Figure 20. The user can also choose between viewing its graph representation, viewing its adjacency matrix, checking the alignment of genes for each clique or gene cluster, or generate a PDF file for a report of the results generated.

Clicking the *View graph* button will allow the user to view a visualization of the graph generated by the tool. The user can interact with the graph and move vertices to view the graph better. Figure 21 shows the window displaying the graph visualization. A directory of vertices will pop out as well to help the user determine the gene cluster corresponding each vertex. After viewing, the user can go back to the results window.

If the user wants to check the adjacency matrix generated, the user can simply click the *View adjacency matrix* button to direct them to a new panel showing the

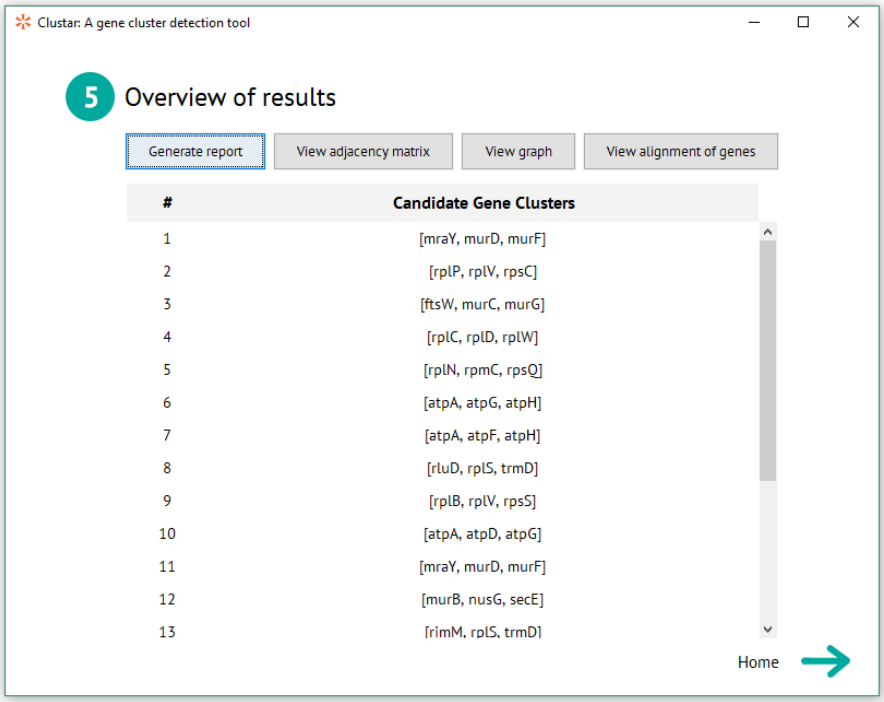


Figure 20: The progress bar for displaying the list of candidate gene clusters

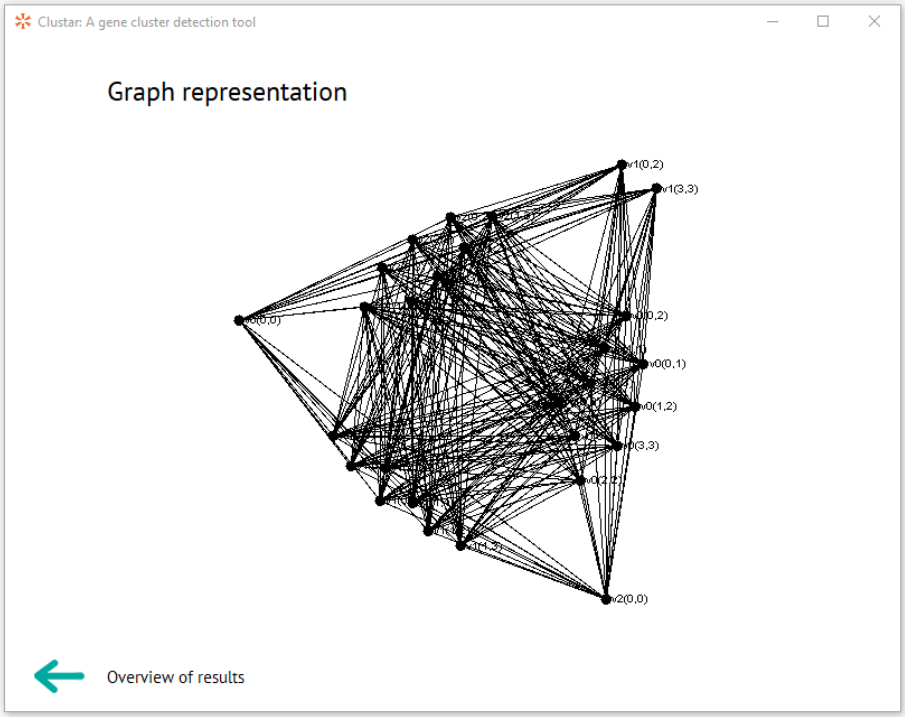
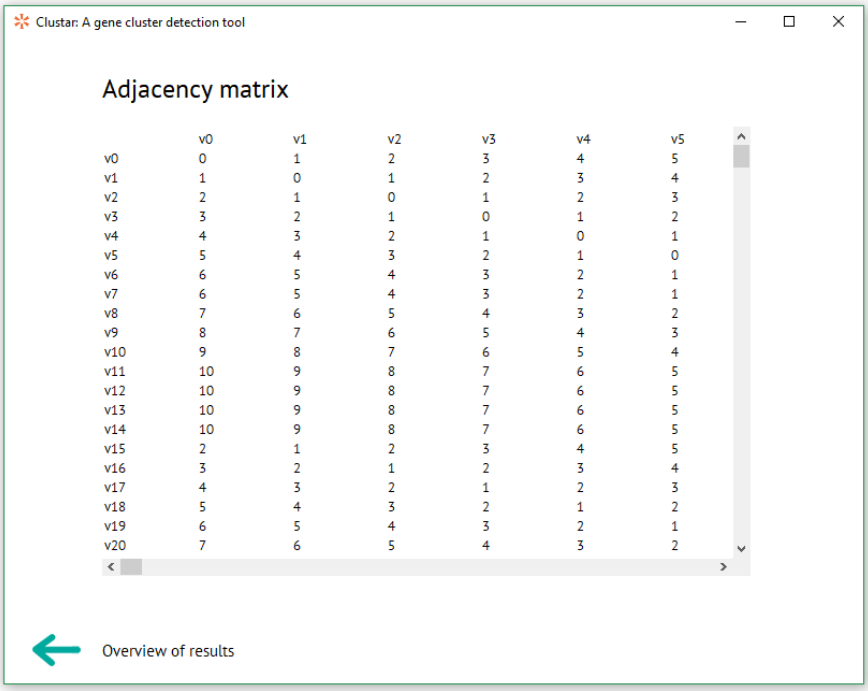


Figure 21: The window for displaying the graph representation

weights of each vertex. Figure 22 shows the window displaying the adjacency matrix. A window for the directory of vertices will pop out to show the corresponding gene cluster for each vertex. After checking the matrix, the user can now go back to the results window.



	v0	v1	v2	v3	v4	v5
v0	0	1	2	3	4	5
v1	1	0	1	2	3	4
v2	2	1	0	1	2	3
v3	3	2	1	0	1	2
v4	4	3	2	1	0	1
v5	5	4	3	2	1	0
v6	6	5	4	3	2	1
v7	6	5	4	3	2	1
v8	7	6	5	4	3	2
v9	8	7	6	5	4	3
v10	9	8	7	6	5	4
v11	10	9	8	7	6	5
v12	10	9	8	7	6	5
v13	10	9	8	7	6	5
v14	10	9	8	7	6	5
v15	2	1	2	3	4	5
v16	3	2	1	2	3	4
v17	4	3	2	1	2	3
v18	5	4	3	2	1	2
v19	6	5	4	3	2	1
v20	7	6	5	4	3	2

Figure 22: The window for displaying the adjacency matrix

The user can also view the alignment of genes based on the gene cluster and cliques generated by the tool, as shown in Figure 23. Each gene is aligned based on their location in the genome and is highlighted in red if it appears across all genomes. The user will be able to check the alignment of each gene cluster or clique by choosing on the drop down box above.

Lastly, the user can generate a PDF file for a report of the results generated. A file navigator appears to guide the user on where to store the report. The PDF file includes the list of genomes uploaded, the method of gene clustering being used, the input parameters specified, and the candidate gene clusters generated. The report also includes the alignment of genes per candidate gene cluster or clique.

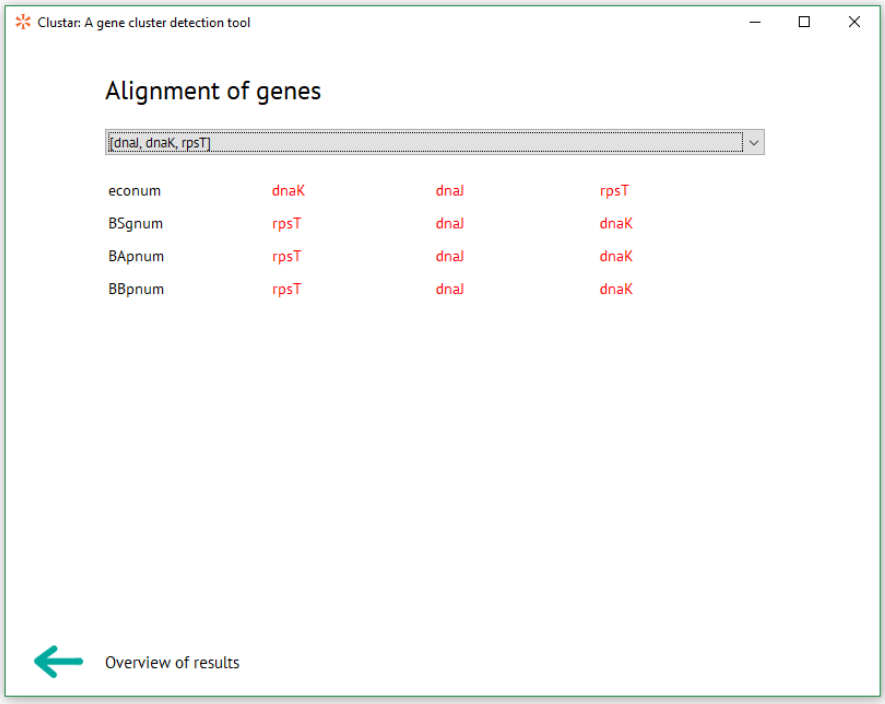


Figure 23: The window for displaying the alignment of genes

After viewing the results, the user can go back to the home window where the user can again search for gene clusters.

VI. Discussions

Clustar is a tool for detecting gene clusters by finding the minimum-weighted t-clique where t is the number of genomes uploaded. Using Clustar, the user is able to view the graph representation of the dataset, its adjacency matrix, and to view the alignment of genes based on its results. Because of this, biologists are able to analyze the relationship of genes between different species.

Clustar offers three methods on finding gene clusters: Approximate Gene Clustering, Exact Gene Clustering (using GPU), and Exact Gene Clustering (without GPU). The approximate gene clustering method detects gene clusters by approximating the minimum-weighted t-clique with the minimum-weighted star. The exact gene clustering method generates all possible combinations of t-cliques and finds the clique with the minimum weight.

Unlike other gene detection tools, Clustar utilizes graph concepts and its approximation algorithms in detecting gene clusters. This speeds up the process of finding clusters of genes. However, Clustar can only generate candidate gene clusters and these results are still subject to a more detailed analysis in the laboratory for biologists and bioinformaticians. Clustar also limits the number of genomes to up to 5 for approximate gene clustering, and for 2 genomes only on exact gene clustering to prevent memory problems.

The algorithm used in Clustar and in GECKO 3 present various similarities and differences. For both tools, the algorithm used ignores order and multiplicity of genes. This means that when a section of the genome contains $ABBAC$, the gene clusters generated will contain ABC only. Both algorithms relax the constraint of gene duplication and repetition [18].

However, the algorithm used in Gecko 3 uses other parameters as its input. It introduces the quorum parameter k' which specifies the minimum number of genomes which contains the cluster. This parameter relaxes the constraint of generating gene

clusters across all genomes. The model used for Clustar, on the other hand, do not allow this constraint, thus finding clusters of genes that are common to all genomes entered by the user [18].

Both tools also utilize exact algorithms in finding gene clusters. Although exact algorithms provide better results for finding gene clusters, an approximation algorithm is also offered in Clustar to account for faster processing time. Another feature Clustar provides for the exact algorithm is the option for utilizing a GPU for faster detection of gene clusters for large datasets [18].

The algorithm used in Gecko 3 also applies statistical evaluation to its results. A p-value is calculated for each generated reference gene cluster to test its significance. Gecko 3 allows filtering of results to show only gene clusters with the best p-value [18].

CYNTENATOR is another tool which uses an algorithm that detects regions in the genome with conserved synteny across large set of organisms. Its algorithm and the algorithm used in Clustar use sequences of genes as its input. However, CYNTENATOR determines homologous genes by computing its BLASTP searches. Clustar do not obtain homologous genes, instead, it accepts input with homologous genes correctly annotated already [19].

CYNTENATOR operates using a pairwise alignment method. This method compares two genomes at a time for detecting gene clusters. The algorithm used in Clustar, on the contrary, compares and handles multiple genomes [19].

Similar to Clustar, the Max-gap Clusters by Multiple Sequence Comparison (MCMuSec) uses the gene teams model in detecting gene clusters. The order of the genes is ignored with this model. MCMuSec uses the decompose and filter procedure. It removes genes from subsets which do not contain homologs in the other subsets. Then, it sorts genes based on its position on the chromosome. The subset is broken down into smaller subsets and removes subsets which are less than the *minsize*. The sig-

nature for each subset is computed, removing those with uncommon signatures. The decompose-filter procedure is applied recursively until all subsets produced are less than the specified *minsize* [20].

MCMuSec also applies statistical methods to include evolutionary relationships across genomes. Because of this, gene clusters under evolutionary constraints are identified even with reordering of genes [20].

A dataset adapted from [25] was used to check the results of the tool. The dataset contains 244 homologous genes found in 30 γ-proteobacterial genomes. The Row's Subset of Symmetric Matrix (RSSM), another approximation algorithm, was also used to process the dataset and verify the results produced by the tool. The RSSM approximates the minimum-weighted t-clique by finding the minimum-weighted t-star of the generated graph. A 2-approximation algorithm was introduced in [26] such that for each $j = 1, 2, \ldots, n$, we find a set B_j that consists of indices of t smallest entries in the jth row of an nxn symmetric matrix witn non-negative integers including the j itself. The symmetric matrix used in the algorithm was derived from the weights of the edges–the total distances between genes across multiple genomes. The results for both approximation algorithms are confirmed by performing hierarchical clustering and generating a dendrogram of clusters of genes.

Let the missing and additional weights be equal to 1. Specifying the size of gene cluster = 3, the RSSM algorithm and the approximation algorithm used by the tool generated a total of 9 common gene clusters:

1. murF, murE, ftsI
2. rimM, trmD, rpsP
3. rpsK, rpsD, rpsM
4. rplO, rpmD, secY

5. rpmD, rplO, rpsE

6. rpsE, rplR, rpmD

7. rplX, rplE, rplN

8. rplC, tpsJ, rplD

9. atpA, atpG, atpH

Verifying the results produced using hierarchical clustering, Figure 24 shows that the genes rplC, rpsJ, and rplD were found to be clustered together as well.

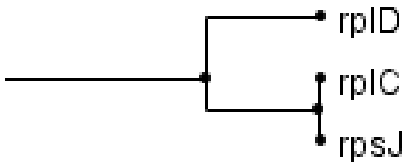


Figure 24: A dendrogram showing the genes with cluster size = 3.

The genes rplB, rpsS, and rplW were also found to be clustered together with an additional gene as shown in Figure 25.

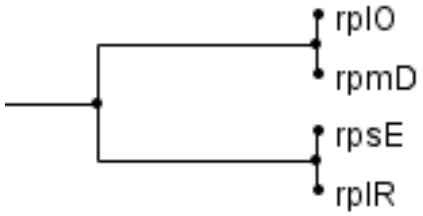


Figure 25: A dendrogram showing the genes with cluster size = 3 with an additional gene.

The genes rimM, trmD, and rpsP, together with 2 additional genes were found to be clustered together in the dendrogram showin in Figure 26

Running both algorithms with size of gene cluster= 4, both approximation algorithms generated 4 common gene clusters:

1. rimM, trmD, rpsP, rplS

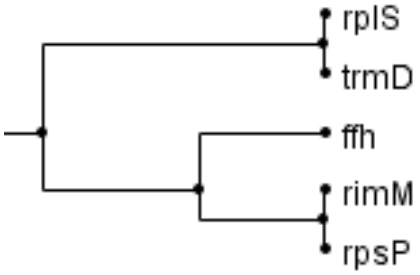


Figure 26: A dendrogram showing the genes with cluster size = 3 with two additional genes.

2. rpsK, rpsD, rpsM, rpoA

3. atpA, atpG, atpH, atpD

4. atpF, atpE, atpH, atpB

Displaying the dendrogram generated by hierarchical clustering, Figure 27 shows how the genes atpF, atpE, atpH, and atpB were found to be clustered together.

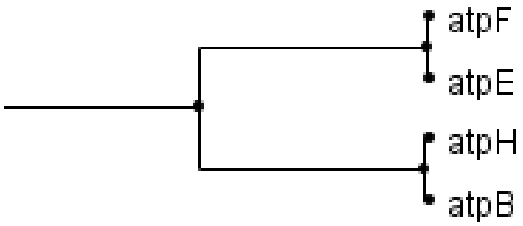


Figure 27: A dendrogram showing the genes with cluster size = 4.

Another set of genes rimM, trmD, rpsP, and rplS are also found together in a cluster with an addition of one gene as shown in Figure 28.

The approximation algorithm used by the tool and RSSM produced 2 common candidate gene clusters with size equal to 5:

1. rpsE, rplR, rpmD, rplO, rplF

2. rplV, rpsS, rpsC, rplB, rplP

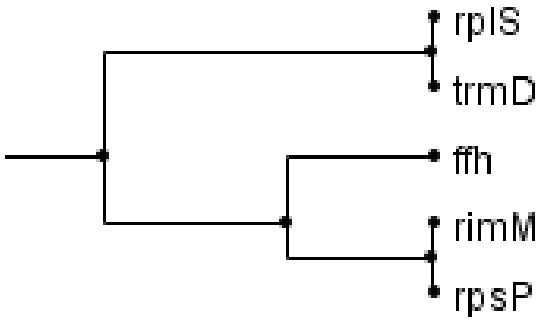


Figure 28: A dendrogram showing the genes with cluster size = 4 with an additional gene.

The dendrogram shown in Figure 29 shows the genes rplO, rpmD, rpsE, and rplR without the missing gene rplF.

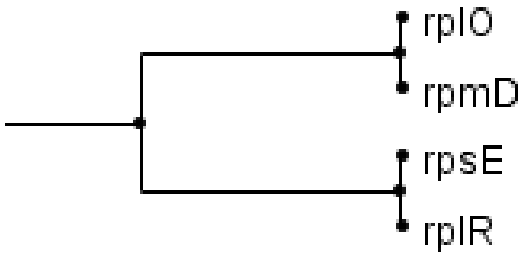


Figure 29: A dendrogram showing the genes with cluster size = 5 with a missing gene.

With the size of gene cluster equal to 6, three common candidate gene clusters were produced:

1. rpsE, rplR, rpmD, rplO, rplF, secY
2. rplV, rpsS, rpsC, rplB, rplP, rplW
3. atpA, atpG, atpH, atpD, atpF, atpC

The dendrogram in Figure 30 shows the genes rpsE, rplR, rpmD, rplO, rplF, and secY with two additional genes, rpsM and rplF.

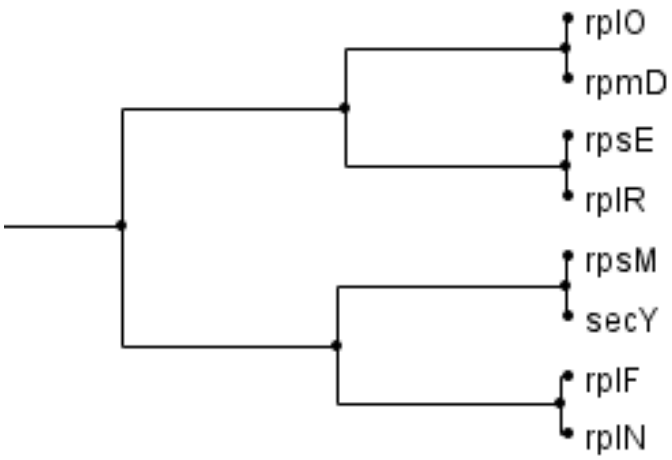


Figure 30: A dendrogram showing the genes with cluster size = 6 with two additional genes.

Using size of gene cluster = 7, both algorithms yielded the following gene clusters:

1. rpsE, rplR, rpmD, rplO, rplF, secY, rpsH
2. rpsH, rpsN, rplF, rplE, rplR, rplX, rpsE
3. rplV, rpsS, rpsC, rplB, rplP, rplW, rpmC
4. atpA, atpG, atpH, atpD, atpF, atpC, atpE

Running hierarchical clustering in the dataset, the dendrogram in Figure 31 shows that the genes rpsE, rplR, rpmD, rplO, rplF, and secY are clustered together with missing and additional genes.

Another set of genes atpF, atpE, atpH, atpG, atpA, atpC, and atpD were clustered together with 2 additional genes in Figure 32.

Setting the size of the gene cluster to 8 produced the following results:

1. rpsE, rplR, rpmD, rplO, rplF, secY, rpsH, rpsN
2. rplR, rpsE, rplF, rpmD, rpsH, rplO, rpsN, secY
3. rplF, rpsH, rplR, rpsN, rpsE, rplE, rpmD, rplX

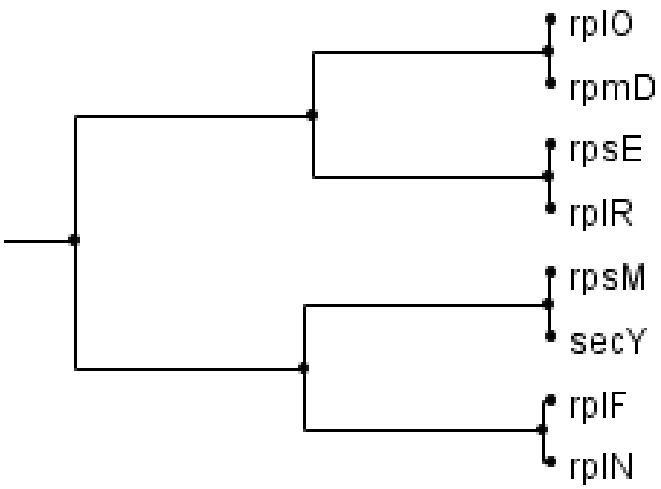


Figure 31: A dendrogram showing the genes with cluster size = 7 with missing and additional genes.

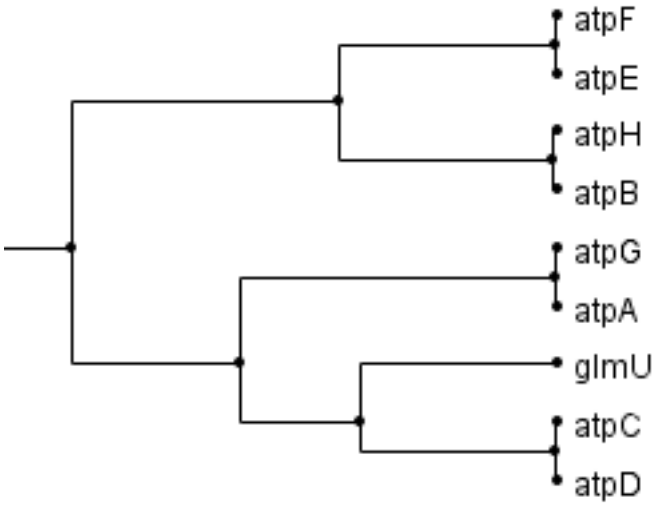


Figure 32: A dendrogram showing the genes with cluster size = 7 with 2 additional genes.

4. rpsH, rpsN, rplF, rplE, rplR, rplX, rpsE, rplN

5. rplV, rpsS, rpsC, rplB, rplP, rplW, rpmC, rplD

6. rplB, rpsS, rplW, rplV, rplD, rpsC, rplC, rplP

7. atpA, atpG, atpH, atpD, atpF, atpC, atpE, atpB

Figure 33 shows the genes rplO, rpmD, rpsE, rplR, rpsM, secY, rplF, rplN, rpsH,

rpsN, rplE, and rplX, including the additional genes, clustered together.

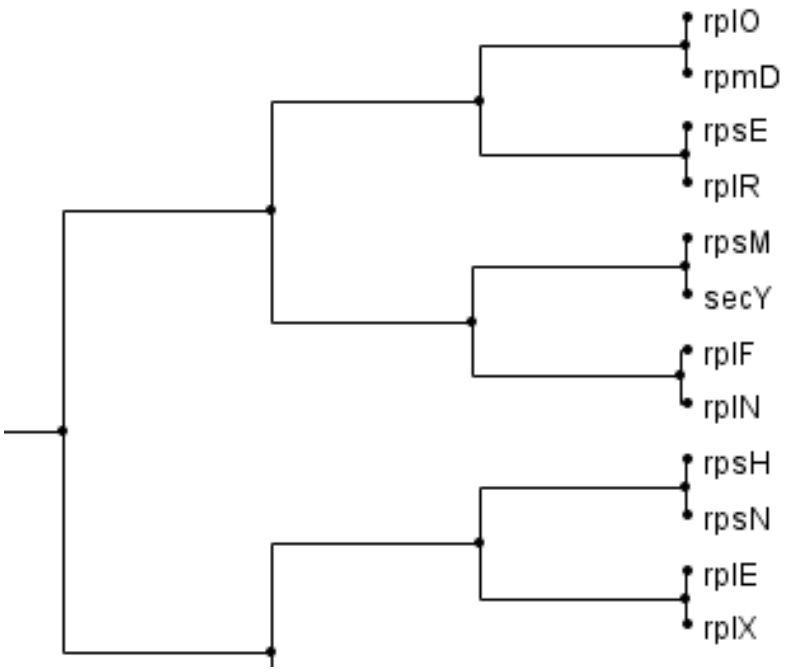


Figure 33: A dendrogram showing the genes with cluster size = 8 with additional genes.

Figure 34 shows the genes clustered together from the dendrogram generated. The genes atpA, atpG, atpH, atpD, atpF, atpC, atpE, atpB form a cluster and thus verifies the results of both approximate algorithms.

Another dataset was adapted to compare the gene clusters generated by the tool and the gene-groups produced from [27]. The dataset is composed of genes from *E. coli* and *B. subtilis* genomes. A representative subset of the gene-groups as pathway seeds are shown on Table 1.

First, the size of gene cluster was set to 2, with the value of the missing and additional weights equal to 1. The following gene clusters were produced by the tool which matched the gene-groups specified from the table by Bansal [27].

1. sucA, sucB

2. sucC, sucD

Table 1: Table of gene-groups as pathway seeds

Citrate cycle	(sdhA, sdhB), (sucA, sucB), (sucC, sucD)
Lysine degradation	(sucA, sucB)
Urea cycle and metabolism of amino groups	(proB, proA), (argC, argB)
Reductive carboxylate cycle	(sdhA, sdhB), (sucC, sucD), (yjcG, acs)
Phenylalanine, tyrosine, and tryptophan biosynthesis	(pheT, pheT)
Purine metabolism	(guaA, guaB), (purM, purN), (cysC, cysH), (prsA, ychB), (purD, purH)
Peptideglycan biosynthesis	(yabB, yabC, ftsI, murE, murD, murF, mraY, ftsW, murG, murC)
Flagellar assembly	(fliE, fliF, fliG, fliI, fliJ, fliM, fliN, flip, fliQ, fliR)
Pentose and glucuronate Interconversions	(araD, araA, araB), (lyxk, yiaS), (orf, ygcE)
Histidine metabolism	(hisG, hisD, hisB, hisH, hisA, hisF, hisI)
Fatty acid biosynthesis	(plsX, fabD, fabG)
Terpenoid biosynthesis	(orf, ispA, xseB)
Alanine and aspartate metabolism	(argH, oxyR)
Cs-branched dibasic acid metabolism	(sucC, sucC)
Phopholipid degradation	(glpQ, glpT)
Valine, leucine, and isoleucine biosynthesis	(leuC, leuB, leuA)
Folate biosynthesis	(folP, hflB)
Pentose phosphate cycle	(prsA, ychB), (deoC, deoA)
Glyciine, serine, and threonine metabolism	(gcvP, gcvT), (betB, orf), (ycaJ, serS), (thrA, thrB, thrC)
Glycolisis	(ascF, ascB)
Pyrimidine metabolism	(carA, carB), (rhlB, trxA), (deoC, deoA), (tmk, holB)
Glyoxylate and dicarboxylate metabolism	(def, fmt, fmu), (yddg, fdnG)
Glycerolipid metabolism	(glpQ, glpT), (glpK, glpF)
Riboflavin metabolism	(ribD, ribH)
Glutamate metabolism	(gltA, gltB)
Porphyrin and cholorophyll metabolism	(proB, proA)
Arginine and proline metabolism	(argH, oxyR)
Pantothenate and CoA biosynthesis	(panD, panC, panB)

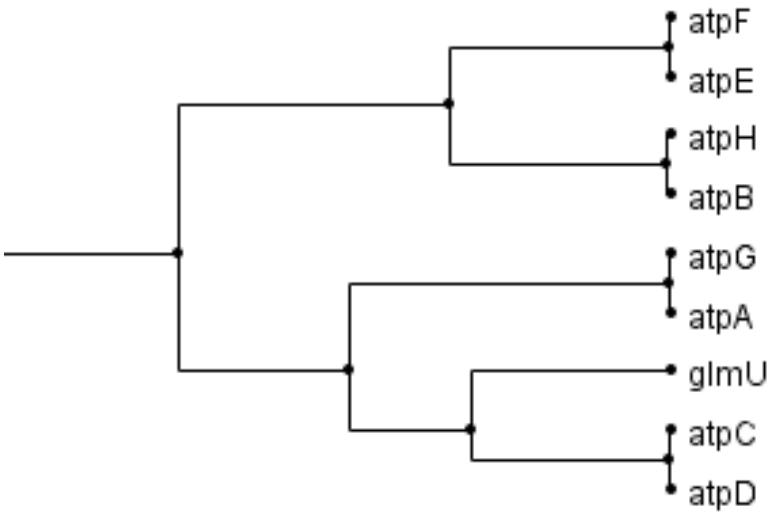


Figure 34: A dendrogram showing the genes with cluster size = 8.

Specifying the gene cluster size to 3, the following gene clusters were found common between the results of the tool and the list of gene-groups:

1. orf, ispA, xseB
2. leuC, leuB, leuA,
3. thrA, thrB, thrC

Increasing the size of gene cluster to 7, the gene-group *Histidine metabolism* with genes *hisA, hisB, hisD, hisF, hisG, hisH, hisI* was also produced by the tool.

Based from the findings above, Clustar was able to provide gene clusters that are also produced by other approximation algorithms such as RSSM and hierarchical clustering. With another dataset, the results of Clustar was also verified using the gene groups produced by Bansal in [27].

VII. Conclusions

Clustar is a tool that utilizes graph concepts and approximation algorithms in detecting gene clusters. The tool provides three methods in obtaining gene clusters: Approximate Gene Clustering, Exact Gene Clustering (using GPU), and Exact Gene Clustering (without GPU). The tool displays the list of candidate gene clusters or cliques generated, the graph generated, its adjacency matrix, and its alignment to view the common genes across genomes.

The tool produced can be used as a guide for biologists to use for finding relationships of genes within genomes and defining functions of clusters of genes. Clustar is only able to generate candidate gene clusters; researchers still need to verify in the laboratory using other tools to get a more detailed analysis on the relationship between genes.

VIII. Recommendations

Clustar is able to generate meaningful candidate gene clusters but there are still features that can be improved. One is to be able to check the dataset entered by the user and check if the homologs are correctly annotated.

The visualization of the graph can be also be further improved to accomodate larger datasets and to allow interaction between the user to fully understand the graph being generated by the algorithm.

Additional analysis on the candidate gene clusters can also be added to the tool to let biologists create conclusions on the relationships between genes and their resulting functions.

The running time of the algorithms used by the tool can also be further optimized to deal with shorter processing times for larger datasets.

IX. Bibliography

[1] J. R. Brown, *Comparative genomics: basic and applied research*. CRC Press, 2007.

[2] D. E. Soltis and P. S. Soltis, "The role of phylogenetics in comparative genetics," *Plant Physiology*, vol. 132, no. 4, pp. 1790–1800, 2003.

[3] B. F. Keele, F. Van Heuverswyn, Y. Li, E. Bailes, J. Takehisa, M. L. Santiago, F. Bibollet-Ruche, Y. Chen, L. V. Wain, F. Liegeois, *et al.*, "Chimpanzee reservoirs of pandemic and nonpandemic hiv-1," *Science*, vol. 313, no. 5786, pp. 523–526, 2006.

[4] J. Lawrence, "Selfish operons: the evolutionary impact of gene clustering in prokaryotes and eukaryotes," *Current opinion in genetics & development*, vol. 9, no. 6, pp. 642–648, 1999.

[5] A. Bergeron, S. Corteel, and M. Raffinot, "The algorithmic of gene teams," in *International Workshop on Algorithms in Bioinformatics*, pp. 464–476, Springer, 2002.

[6] A. Bergeron, C. Chauve, and Y. Gingras, "Formal models of gene clusters," *Bioinformatics algorithms: techniques and applications*, vol. 8, pp. 177–202, 2008.

[7] S. Rahmann and G. W. Klau, "Integer linear programming techniques for discovering approximate gene clusters," *Bioinformatics Algorithms: Techniques and Applications, Wiley Series on Bioinformatics: Computational Techniques and Engineering*, pp. 203–222, 2008.

[8] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos, "Using graph theory to analyze biological networks," *BioData mining*, vol. 4, no. 1, p. 10, 2011.

[9] J. A. Aborot, H. Adorna, J. B. Clemente, B. K. de Jesus, and G. Solano, "Search for a star: Approximate gene cluster discovery problem (agcdp) as minimization problem," *Philippine Computing Journal*, vol. 7, pp. 1–11, 2012.

[10] G. A. Solano, *Clique Finding Approach for the Approximate Gene Cluster Discovery Problem*. PhD thesis, University of the Philippines Diliman, 2017.

[11] G. S. Cabunducan, J. B. Clemente, H. N. Adorna, and R. T. Relator, "Probing the hardness of the approximate gene cluster discovery problem (agcdp)," in *Theory and Practice of Computation: Proceedings of Workshop on Computation: Theory and Practice WCTP2013*, pp. 35–47, 2013.

[12] G. Blin, D. Faye, and J. Stoye, "Finding nested common intervals efficiently," *Journal of Computational Biology*, vol. 17, no. 9, pp. 1183–1194, 2010.

[13] C. Moreno, J. Lazar, H. J. Jacob, and A. E. Kwitek, "Comparative genomics for detecting human disease genes," *Advances in genetics*, vol. 60, pp. 655–697, 2008.

[14] A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve, "Reconstructing ancestral gene orders using conserved intervals," *Algorithms in bioinformatics*, pp. 14–25, 2004.

[15] Q. Yang, G. Yi, F. Zhang, M. R. Thon, and S.-H. Sze, "Identifying gene clusters within localized regions in multiple genomes," *Journal of Computational Biology*, vol. 17, no. 5, pp. 657–668, 2010.

[16] I. B. Rogozin, K. S. Makarova, J. Murvai, E. Czabarka, Y. I. Wolf, R. L. Tatusov, L. A. Szekely, and E. V. Koonin, "Connected gene neighborhoods in prokaryotic genomes," *Nucleic acids research*, vol. 30, no. 10, pp. 2212–2223, 2002.

[17] M. Sémon and L. Duret, "Evolutionary origin and maintenance of coexpressed gene clusters in mammals," *Molecular biology and evolution*, vol. 23, no. 9, pp. 1715–1723, 2006.

[18] S. Winter, K. Jahn, S. Wehner, L. Kuchenbecker, M. Marz, J. Stoye, and S. Böcker, "Finding approximate gene clusters with gecko 3," *Nucleic acids research*, vol. 44, no. 20, pp. 9600–9610, 2016.

[19] C. Rödelsperger and C. Dieterich, "Cyntenator: progressive gene order alignment of 17 vertebrate genomes," *PloS one*, vol. 5, no. 1, p. e8861, 2010.

[20] X. Ling, X. He, and D. Xin, "Detecting gene clusters under evolutionary constraint in a large number of genomes," *Bioinformatics*, vol. 25, no. 5, pp. 571–577, 2009.

[21] G. Yi, S.-H. Sze, and M. R. Thon, "Identifying clusters of functionally related genes in genomes," *Bioinformatics*, vol. 23, no. 9, pp. 1053–1060, 2007.

[22] S. Heber and J. Stoye, "Algorithms for finding gene clusters," pp. 252–263, 2001.

[23] R. Hoberman and D. Durand, "The incompatible desiderata of gene cluster properties," *Comparative Genomics*, vol. 3678, pp. 73–87, 2005.

[24] W. Tutte, "Graph theory. cambridge mathematical library," *WT Tutte.–January 29, 2001.–333 p*, 2001.

[25] E. Belda, A. Moya, and F. J. Silva, "Genome rearrangement distances and gene order phylogeny in -proteobacteria," *Molecular Biology and Evolution*, vol. 22, no. 6, pp. 1456–1467, 2005.

[26] I. Eremin, E. K. Gimadi, A. Kelmanov, A. Pyatkin, and M. Y. Khachai, "2-approximation algorithm for finding a clique with minimum weight of vertices and edges," *Proceedings of the Steklov Institute of Mathematics*, vol. 284, no. 1, pp. 87–95, 2014.

[27] A. K. Bansal, "A framework of automated reconstruction of microbial metabolic pathways," in *Bio-Informatics and Biomedical Engineering, 2000. Proceedings. IEEE International Symposium on*, pp. 184–190, IEEE, 2000.

X. Appendix

A. Source Code

Listing 1: Vertex

```
package geneclustertool;

import java.util.TreeSet;

/**
 * A class for the vertex object and its helper functions
 *
 * @author Bianca Silmaro
 */
public class Vertex {

        private TreeSet<Integer> geneContent = new TreeSet<Integer>();
        private TreeSet<String> geneClusterName = new TreeSet<String>();
        private int partitionNumber;
        private int startIndex;
        private int endIndex;

        public Vertex(TreeSet<Integer> geneContent, int partitionNumber, int startIndex, int
             endIndex) {
                this.geneContent = geneContent;
                this.partitionNumber = partitionNumber;
                this.startIndex = startIndex;
                this.endIndex = endIndex;
        }

        /**
         * Returns the gene contents of a vertex
         */
        public TreeSet<Integer> getGeneContent() {
                return geneContent;
        }

        /**
         * Returns the partition number of the vertex
         */
        public int getPartitionNumber() {
                return partitionNumber;
        }

        /**
         * Returns the starting index of the interval used to form the vertex
         */
        public int getStartIndex() {
                return startIndex;
        }

        /**
         * Returns the end index of the interval used to form the vertex
         */
        public int getEndIndex() {
                return endIndex;
        }

        /**
         * Renames the integer representation of the gene content to its corresponding gene name
         * @param geneClusterName
         */
        public void setGeneClusterName(TreeSet<String> geneClusterName) {
                this.geneClusterName = geneClusterName;
        }

        /**
         * Returns the corresponding gene name of the gene contents
         * @return
         */
        public TreeSet<String> getGeneClusterName() {
                return geneClusterName;
        }

        /**
         * Computes the weight of the vertex using set difference
         * @param v The second vertex being compared
         * @param missing The weight for missing genes
         * @param additional The weight for additional genes
         * @return
         */
        public int computeWeight(Vertex v, int missing, int additional) {
                int weight = 0;
```

```
                TreeSet<Integer> removed = new TreeSet<Integer>(this.getGeneContent());
                TreeSet<Integer> added = new TreeSet<Integer>(v.getGeneContent());
                removed.removeAll(v.getGeneContent());
                added.removeAll(this.getGeneContent());
                weight = (missing * removed.size()) + (additional * added.size());
                return weight;
        }
}
```

Listing 2: Canvas

```
package geneclustertool;

import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.TreeSet;
import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.GroupLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;
import javax.swing.SwingWorker;
import javax.swing.border.EmptyBorder;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableCellRenderer;
import org.graphstream.graph.Graph;
import org.graphstream.ui.swingViewer.ViewPanel;
import org.graphstream.ui.view.Viewer;

public class Canvas {

        public JFrame frame;
        public JPanel cards;
        CardLayout cl = new CardLayout();

        JPanel main, upload, dataset, method, input, process, results, adjMat, graph, alignment,
            help, progress;
        public int width, height;
        private JFileChooser fc, sfc;
        private JProgressBar progressBar;
        private FileNameExtensionFilter filter = new FileNameExtensionFilter("CSV", "csv");
        private int typeOfGeneClustering = -1;
        private File file;
        private BufferedImage headerImg, nextButtonIcon, backButtonIcon, nextButtonRolloverIcon,
            backButtonRolloverIcon = null;
        private JTextField fileName;
        private JLabel genomeStats = new JLabel();
        private JLabel gpuNotification;
        private JComboBox<String> listOfGenomes = new JComboBox<String>();
        private JTextArea genesIntRepresentation = new JTextArea();
        private JTextField sizeOfGeneClusterField, missingWeightField, additionalWeightField,
            numOfCandidateGeneClustersField;
        private JTable alignmentTable;

        String font = "PT Sans";
        private ArrayList<String> genomes = new ArrayList<String>();
```

```
private ArrayList<ArrayList<Integer>> genesIntegerForm = new ArrayList<ArrayList<Integer
    >>();
private ArrayList<TreeSet<String>> baseGeneClusters = new ArrayList<TreeSet<String>>();
private ArrayList<ArrayList<Vertex>> sortedGeneClusters = new ArrayList<ArrayList<Vertex
    >>();
private int sizeOfGeneCluster, missing, additional, numOfCandidateGeneClusters;
private int minVertexSizeVal, maxVertexSizeVal = 1;

GeneClusteringAlgorithm gc = new GeneClusteringAlgorithm();
ApproximateGeneClustering approx;

/**
 * Initializes and sets up the GUI of the program
 * @param bg The background image used for the main window
 */
@SuppressWarnings("serial")
public Canvas(Image bg) {
        // Create and set up the window.
        frame = new JFrame();
        frame.setTitle("Clustar: A gene cluster detection tool");
        try {
                frame.setIconImage(ImageIO.read(getClass().getResource("/images/clustar-
                    icon.png")));
        } catch (IOException e2) {
                e2.printStackTrace();
        }
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create and set up the content pane.
        cards = new JPanel();
        cards.setLayout(cl);
        this.width = 800;
        this.height = 600;
        cards.setPreferredSize(new Dimension(width, height));

        // Images used throughout the swing program
        try {
                headerImg = ImageIO.read(getClass().getResource("/images/header.png"));
        } catch (IOException e1) {
                e1.printStackTrace();
        }

        try {
                nextButtonIcon = ImageIO.read(getClass().getResource("/images/right-arrow.
                    png"));
        } catch (IOException e1) {
                e1.printStackTrace();
        }

        try {
                backButtonIcon = ImageIO.read(getClass().getResource("/images/left-arrow.
                    png"));
        } catch (IOException e1) {
                e1.printStackTrace();
        }

        try {
                nextButtonRolloverIcon = ImageIO.read(getClass().getResource("/images/
                    right-arrow-hover.png"));
        } catch (IOException e1) {
                e1.printStackTrace();
        }

        try {
                backButtonRolloverIcon = ImageIO.read(getClass().getResource("/images/left
                    -arrow-hover.png"));
        } catch (IOException e1) {
                e1.printStackTrace();
        }

        // Items for card 0 -- main title
        main = new JPanel(true) {
                        @Override
                        public void paintComponent(Graphics g) {
                                super.paintComponent(g);
                            g.drawImage(bg, 0, 0, null);
                         }
                };
        main.setLayout(new GridBagLayout());
        JPanel subMain = new JPanel();
        subMain.setOpaque(false);
        BufferedImage title = null;
        try {
                title = ImageIO.read(getClass().getResource("/images/main-title.png"));
        } catch (IOException e1) {
                e1.printStackTrace();
        }
        JLabel mainTitle = new JLabel(new ImageIcon(title));
        JLabel subtitle = new JLabel("A clique-finding tool for detecting gene clusters");
        subtitle.setFont(new Font(font, Font.PLAIN, 26));
        subtitle.setForeground(Color.WHITE);
        JButton startBtn = new JButton("Find gene clusters");
```

```
startBtn.setMinimumSize(new Dimension(200, 45));
startBtn.setFont(new Font(font, Font.PLAIN, 16));
JButton helpBtn = new JButton("About Clustar");
helpBtn.setMinimumSize(new Dimension(200, 45));
helpBtn.setFont(new Font(font, Font.PLAIN, 16));

GroupLayout mainLayout = new GroupLayout(subMain);
subMain.setLayout(mainLayout);
mainLayout.setAutoCreateGaps(true);
mainLayout.setAutoCreateContainerGaps(true);

mainLayout.setHorizontalGroup(
                mainLayout.createSequentialGroup()
                .addContainerGap(150,150)
                .addGroup(mainLayout.createParallelGroup(GroupLayout.Alignment.
                    CENTER)
                        .addComponent(mainTitle)
                        .addComponent(subtitle)
                        .addGroup(mainLayout.createSequentialGroup()
                                .addComponent(startBtn)
                                .addGap(50)
                                .addComponent(helpBtn)))
                .addContainerGap(150,150)
);

mainLayout.setVerticalGroup(
                mainLayout.createSequentialGroup()
                .addComponent(mainTitle)
                .addGap(20)
                .addComponent(subtitle)
                .addGap(40)
                .addGroup(mainLayout.createParallelGroup(GroupLayout.Alignment.
                    CENTER)
                        .addComponent(startBtn)
                        .addComponent(helpBtn))
                .addGap(100)
);

subMain.add(mainTitle);
subMain.add(subtitle);
subMain.add(startBtn);
subMain.add(helpBtn);
main.add(subMain);

// Items for card 10 -- help view
help = new JPanel();

JLabel headerLbl = new JLabel(new ImageIcon(headerImg));
JLabel helpTitle = new JLabel("Welcome to Clustar!");
helpTitle.setFont(new Font(font, Font.BOLD, 28));

String shortDescText = "Clustar is a tool for detecting gene clusters. Clustar
    offers three methods on finding gene clusters: Approximate Gene Clustering,
    Exact Gene Clustering (using GPU), and Exact Gene Clustering (without GPU).";
JLabel shortDesc = new JLabel("<html><body style='width: 90%'>" + shortDescText);
shortDesc.setFont(new Font(font, Font.PLAIN, 14));

JLabel dataSubTitle = new JLabel("Dataset");
dataSubTitle.setFont(new Font(font, Font.BOLD, 20));
dataSubTitle.setForeground(new Color(0, 169, 157));
JTextArea dataSubTextArea = new JTextArea(2, 20);
dataSubTextArea.setFocusable(false);
dataSubTextArea.setEditable(false);
String dataSubText = "Your dataset must be in CSV format and should look like this
    : \n";
dataSubText += "\t scientific name of species,gene 1,gene 2,gene 3,...,gene n";
dataSubTextArea.setFont(new Font(font, Font.PLAIN, 14));
dataSubTextArea.setText(dataSubText);
dataSubTextArea.setTabSize(1);

JLabel inputSubTitle = new JLabel("Input");
inputSubTitle.setFont(new Font(font, Font.BOLD, 20));
inputSubTitle.setForeground(new Color(0, 169, 157));
String geneConstraintsStr = "The range for the size of gene contents specifies the
     interval size of the gene contents to be created.";
String sizeOfGeneClusterStr = "The size of gene cluster specifies the number of
    genes to be considered as a gene cluster.\n";
String integerWeights = "The missing and additional weights specify the cost of
    missing and additional genes.\n";
String numOfOutput = "The number of candidate gene clusters to be displayed
    specifies the number of output gene clusters to be displayed by the tool.";
JLabel geneConstraintsTextArea = new JLabel("<html><body style='width: 100%'>" +
    geneConstraintsStr);
geneConstraintsTextArea.setFont(new Font(font, Font.PLAIN, 14));
JLabel sizeOfGeneClusterTextArea = new JLabel("<html><body style='width: 100%'>"
    + sizeOfGeneClusterStr);
sizeOfGeneClusterTextArea.setFont(new Font(font, Font.PLAIN, 14));
JLabel integerWeightsTextArea = new JLabel("<html><body style='width: 100%'>" +
    integerWeights);
integerWeightsTextArea.setFont(new Font(font, Font.PLAIN, 14));
JLabel numOfOutputTextArea = new JLabel("<html><body style='width: 100%'>" +
    numOfOutput);
```

```
numOfOutputTextArea.setFont(new Font(font, Font.PLAIN, 14));

JLabel resultsSubTitle = new JLabel("Results");
resultsSubTitle.setFont(new Font(font, Font.BOLD, 20));
resultsSubTitle.setForeground(new Color(0, 169, 157));
String resultsText = "Results of the chosen method will show the list of candidate
     gene clusters generated. The user can choose between viewing its graph
    representation, viewing its matrix of weights, viewing its alignment, or
    generating a report of the results.";
JLabel resultsTextArea = new JLabel("<html><body style='width: 100%'>" +
    resultsText);
resultsTextArea.setFont(new Font(font, Font.PLAIN, 14));

JLabel helpBackLabel = new JLabel("Home");
helpBackLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton helpBack = new JButton();
helpBack.setIcon(new ImageIcon(backButtonIcon));
helpBack.setContentAreaFilled(false);
helpBack.setRolloverIcon(new ImageIcon(backButtonRolloverIcon));

GroupLayout helpLayout = new GroupLayout(help);
help.setLayout(helpLayout);
helpLayout.setAutoCreateGaps(true);
helpLayout.setAutoCreateContainerGaps(true);

helpLayout.setHorizontalGroup(
                helpLayout.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addComponent(headerLbl)
                .addGroup(helpLayout.createSequentialGroup()
                        .addComponent(helpBack)
                        .addGroup(helpLayout.createParallelGroup(GroupLayout.
                             Alignment.LEADING)
                                .addComponent(helpTitle)
                                .addComponent(shortDesc)
                                .addComponent(dataSubTitle)
                                .addComponent(dataSubTextArea)
                                .addComponent(inputSubTitle)
                                .addComponent(geneConstraintsTextArea)
                                .addComponent(sizeOfGeneClusterTextArea)
                                .addComponent(integerWeightsTextArea)
                                .addComponent(numOfOutputTextArea)
                                .addComponent(resultsSubTitle)
                                .addComponent(resultsTextArea)
                                .addComponent(helpBackLabel))
                .addGap(55))

);

helpLayout.setVerticalGroup(
                helpLayout.createSequentialGroup()
                .addComponent(headerLbl)
                .addGap(10)
                .addGroup(helpLayout.createParallelGroup(GroupLayout.Alignment.
                     CENTER)
                        .addComponent(helpTitle))
                        .addGap(20)
                        .addComponent(shortDesc)
                        .addGap(10)
                        .addComponent(dataSubTitle)
                        .addComponent(dataSubTextArea)
                        .addGap(10)
                        .addComponent(inputSubTitle)
                        .addComponent(geneConstraintsTextArea)
                        .addComponent(sizeOfGeneClusterTextArea)
                        .addComponent(integerWeightsTextArea)
                        .addComponent(numOfOutputTextArea)
                        .addGap(10)
                        .addComponent(resultsSubTitle)
                        .addComponent(resultsTextArea)
                        .addContainerGap(300, 400)
                        .addGroup(helpLayout.createParallelGroup(GroupLayout.
                             Alignment.CENTER)
                                .addComponent(helpBack)
                                .addComponent(helpBackLabel))
                .addContainerGap(50, 50)
);
help.add(headerLbl);
help.add(helpTitle);
help.add(shortDesc);
help.add(dataSubTitle);
help.add(inputSubTitle);
help.add(geneConstraintsTextArea);
help.add(sizeOfGeneClusterTextArea);
help.add(integerWeightsTextArea);
help.add(numOfOutputTextArea);
help.add(resultsSubTitle);
help.add(helpBack);
help.add(helpBackLabel);
help.setBackground(Color.WHITE);

// Items for card 11 -- progress view
progress = new JPanel(new GridBagLayout());
```

```
BufferedImage progressIcon = null;
try {
        progressIcon = ImageIO.read(getClass().getResource("/images/clustar-icon.
            png"));
} catch (IOException e1) {
        e1.printStackTrace();
}
JLabel progressIconLbl = new JLabel(new ImageIcon(progressIcon));
JPanel subProgress = new JPanel();
JLabel progressLbl = new JLabel();
progressLbl.setFont(new Font(font, Font.PLAIN, 18));
progressBar = new JProgressBar();
progressBar.setString(null);
progressBar.setIndeterminate(true);
progressBar.setBackground(new Color(0, 169, 157));
progressBar.setPreferredSize(new Dimension(width - 400, height - 150));

GroupLayout progressLayout = new GroupLayout(subProgress);
subProgress.setLayout(progressLayout);
progressLayout.setAutoCreateGaps(true);
progressLayout.setAutoCreateContainerGaps(true);

progressLayout.setHorizontalGroup(
                progressLayout.createSequentialGroup()
                .addContainerGap(150,150)
                .addGroup(progressLayout.createParallelGroup(GroupLayout.Alignment
                    .CENTER)
                        .addComponent(progressIconLbl)
                        .addComponent(progressLbl)
                        .addComponent(progressBar))
                .addContainerGap(150,150)
);

progressLayout.setVerticalGroup(
                progressLayout.createSequentialGroup()
                .addComponent(progressIconLbl)
                .addGap(20)
                .addComponent(progressLbl)
                .addGap(20)
                .addComponent(progressBar)
                .addGap(100)
);

subProgress.add(progressLbl);
subProgress.add(progressBar);
subProgress.setBackground(Color.WHITE);
progress.add(subProgress);
progress.setBackground(Color.WHITE);

// Items for card 1 -- upload view
upload = new JPanel();
BufferedImage uploadStepIcon = null;
try {
        uploadStepIcon = ImageIO.read(getClass().getResource("/images/step1.png"))
            ;
} catch (IOException e1) {
        e1.printStackTrace();
}
JLabel uploadHeaderLbl = new JLabel(new ImageIcon(headerImg));
JLabel uploadStep = new JLabel(new ImageIcon(uploadStepIcon));
JLabel uploadTitle = new JLabel("Upload your dataset");
uploadTitle.setFont(new Font(font, Font.PLAIN, 24));
fileName = new JTextField("", 30);
fileName.setEditable(false);
fileName.setFont(new Font(font, Font.PLAIN, 13));
fileName.setPreferredSize(new Dimension(50, 35));
JButton uploadButton = new JButton("Choose a file");
uploadButton.setMinimumSize(new Dimension(50, 35));
uploadButton.setFont(new Font(font, Font.PLAIN, 13));
JTextArea formatDesc = new JTextArea();
String formatDescTxt = "Your dataset must be in CSV format and should look like
    this: \n";
formatDescTxt += "\t scientific name of species,gene 1,gene 2,gene 3,...,gene n";
formatDesc.setText(formatDescTxt);
formatDesc.setTabSize(1);
formatDesc.setFont(new Font(font, Font.ITALIC, 15));
formatDesc.setEditable(false);
JLabel vertexSizeLbl = new JLabel("Specify the range for the size of gene contents
    : ");
vertexSizeLbl.setFont(new Font(font, Font.BOLD, 18));
JLabel minVertexSizeLbl = new JLabel("Minimum size: ");
minVertexSizeLbl.setFont(new Font(font, Font.PLAIN, 15));
JTextField minVertexSize = new JTextField("1", 10);
minVertexSize.setFont(new Font(font, Font.PLAIN, 13));
JLabel maxVertexSizeLbl = new JLabel("Maximum size: ");
maxVertexSizeLbl.setFont(new Font(font, Font.PLAIN, 15));
JTextField maxVertexSize = new JTextField("1", 10);
maxVertexSize.setFont(new Font(font, Font.PLAIN, 15));

JLabel uploadLabel = new JLabel("Display dataset");
uploadLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton enterData = new JButton();
```

```
enterData.setIcon(new ImageIcon(nextButtonIcon));
enterData.setContentAreaFilled(false);
enterData.setRolloverIcon(new ImageIcon(nextButtonRolloverIcon));

GroupLayout uploadLayout = new GroupLayout(upload);
upload.setLayout(uploadLayout);
uploadLayout.setAutoCreateGaps(true);
uploadLayout.setAutoCreateContainerGaps(true);

uploadLayout.setHorizontalGroup(
                uploadLayout.createSequentialGroup()
                .addGap(55)
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    TRAILING)
                        .addComponent(uploadStep))
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    LEADING)
                        .addComponent(uploadTitle)
                        .addComponent(fileName)
                        .addComponent(vertexSizeLbl)
                        .addGroup(uploadLayout.createSequentialGroup()
                                .addComponent(minVertexSizeLbl)
                                .addComponent(minVertexSize)
                                .addGap(20)
                                .addComponent(maxVertexSizeLbl)
                                .addComponent(maxVertexSize)))
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    TRAILING)
                                .addComponent(uploadButton)
                                .addComponent(uploadLabel))
                .addComponent(enterData)
);

uploadLayout.setVerticalGroup(
        uploadLayout.createSequentialGroup()
                .addGap(135)
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    CENTER)
                        .addComponent(uploadStep)
                        .addComponent(uploadTitle))
                .addGap(20)
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    BASELINE)
                        .addComponent(fileName)
                        .addComponent(uploadButton))
                .addGap(20)
                .addComponent(vertexSizeLbl)
                .addGap(10)
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    BASELINE)
                                .addComponent(minVertexSizeLbl)
                                .addComponent(minVertexSize)
                                .addComponent(maxVertexSizeLbl)
                                .addComponent(maxVertexSize))
                .addContainerGap(300, 400)
                .addGroup(uploadLayout.createParallelGroup(GroupLayout.Alignment.
                    CENTER)
                        .addComponent(uploadLabel)
                        .addComponent(enterData))
);

upload.add(uploadHeaderLbl);
upload.add(uploadStep);
upload.add(uploadTitle);
upload.add(uploadLabel);
upload.add(uploadButton);
upload.add(enterData);
upload.add(vertexSizeLbl);
upload.add(minVertexSizeLbl);
upload.add(maxVertexSizeLbl);
upload.add(minVertexSize);
upload.add(maxVertexSize);
upload.setBackground(Color.WHITE);

// Items for card 2 -- display data view
dataset = new JPanel();

BufferedImage displayStepIcon = null;
try {
        displayStepIcon = ImageIO.read(getClass().getResource("/images/step2.png")
            );
} catch (IOException e1) {
        e1.printStackTrace();
}
JLabel displayDataStep = new JLabel(new ImageIcon(displayStepIcon));
JLabel displayDataTitle = new JLabel("View your dataset in its integer form");
displayDataTitle.setFont(new Font(font, Font.PLAIN, 24));
genomeStats.setFont(new Font(font, Font.ITALIC, 15));
listOfGenomes.setFont(new Font(font, Font.PLAIN, 15));
genesIntRepresentation.setEditable(false);
genesIntRepresentation.setFont(new Font("PT Sans", Font.PLAIN, 15));
genesIntRepresentation.setLineWrap(true);
```

```
genesIntRepresentation.setWrapStyleWord(true);
JScrollPane scroll = new JScrollPane(genesIntRepresentation, ScrollPaneConstants.
    VERTICAL_SCROLLBAR_AS_NEEDED, ScrollPaneConstants.
    HORIZONTAL_SCROLLBAR_AS_NEEDED);
scroll.setPreferredSize(new Dimension(200, 400));

JLabel displayNextLabel = new JLabel("Choose gene clustering method");
displayNextLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton displayNext = new JButton();
displayNext.setIcon(new ImageIcon(nextButtonIcon));
displayNext.setContentAreaFilled(false);
displayNext.setRolloverIcon(new ImageIcon(nextButtonRolloverIcon));

GroupLayout displayDataLayout = new GroupLayout(dataset);
dataset.setLayout(displayDataLayout);
displayDataLayout.setAutoCreateGaps(true);
displayDataLayout.setAutoCreateContainerGaps(true);

displayDataLayout.setHorizontalGroup(
        displayDataLayout.createSequentialGroup()
        .addGap(55)
        .addGroup(displayDataLayout.createParallelGroup(GroupLayout.Alignment.
            TRAILING)
                .addComponent(displayDataStep))
        .addGroup(displayDataLayout.createParallelGroup(GroupLayout.Alignment.
            LEADING)
                .addComponent(displayDataTitle)
                .addComponent(genomeStats)
                .addComponent(listOfGenomes)
                .addComponent(scroll)
                .addGroup(displayDataLayout.createParallelGroup(GroupLayout.
                    Alignment.TRAILING)
                        .addComponent(scroll)
                        .addComponent(displayNextLabel)))
        .addComponent(displayNext)
);

displayDataLayout.setVerticalGroup(
                displayDataLayout.createSequentialGroup()
                .addGap(30)
                .addGroup(displayDataLayout.createParallelGroup(GroupLayout.
                    Alignment.CENTER)
                        .addComponent(displayDataStep)
                        .addComponent(displayDataTitle))
                .addGap(20)
                .addComponent(genomeStats)
                .addComponent(listOfGenomes)
                .addGap(15)
                .addComponent(scroll)
                .addGap(15)
                .addGroup(displayDataLayout.createParallelGroup(GroupLayout.
                    Alignment.CENTER)
                                .addComponent(displayNextLabel)
                                .addComponent(displayNext))
);

dataset.add(displayDataStep);
dataset.add(displayDataTitle);
dataset.add(genomeStats);
dataset.add(listOfGenomes);
dataset.add(scroll);
dataset.add(displayNext);
dataset.add(displayNextLabel);
dataset.setBackground(Color.WHITE);


// Items for card 3 -- choose gene clustering tool
method = new JPanel();

BufferedImage chooseMethodIcon = null;
try {
        chooseMethodIcon = ImageIO.read(getClass().getResource("/images/step3.png
            "));
} catch (IOException e1) {
        e1.printStackTrace();
}
JLabel chooseMethodStep = new JLabel(new ImageIcon(chooseMethodIcon));
JLabel chooseMethodTitle = new JLabel("Choose method of gene clustering");
chooseMethodTitle.setFont(new Font(font, Font.PLAIN, 24));
gpuNotification = new JLabel();
JLabel gpuAdv = new JLabel("GPUs are helpful to speed up processing for large
    datasets.");
gpuNotification.setFont(new Font(font, Font.BOLD, 16));
gpuAdv.setFont(new Font(font, Font.BOLD, 16));
gpuNotification.setForeground(new Color(0, 169, 157));
gpuAdv.setForeground(new Color(0, 169, 157));

JRadioButton approxGeneClustering = new JRadioButton("Approximate Gene Clustering
    ");
approxGeneClustering.setMnemonic(KeyEvent.VK_A);
approxGeneClustering.setActionCommand("Approximate Gene Clustering");
approxGeneClustering.setContentAreaFilled(false);
```

```
approxGeneClustering.setFont(new Font(font, Font.PLAIN, 20));
approxGeneClustering.setSelected(true);

JRadioButton exactGeneClustering = new JRadioButton("Exact Gene Clustering (
    without GPU)");
exactGeneClustering.setMnemonic(KeyEvent.VK_E);
exactGeneClustering.setActionCommand("Exact Gene Clustering (without GPU)");
exactGeneClustering.setContentAreaFilled(false);
exactGeneClustering.setFont(new Font(font, Font.PLAIN, 20));
exactGeneClustering.setMaximumSize(new Dimension(200, 200));

JRadioButton exactGeneClusteringGPU = new JRadioButton("Exact Gene Clustering (
    using GPU)");
exactGeneClusteringGPU.setMnemonic(KeyEvent.VK_G);
exactGeneClusteringGPU.setActionCommand("Exact Gene Clustering (using GPU)");
exactGeneClusteringGPU.setContentAreaFilled(false);
exactGeneClusteringGPU.setFont(new Font(font, Font.PLAIN, 20));
exactGeneClusteringGPU.setMaximumSize(new Dimension(200, 200));

ButtonGroup methodGroup = new ButtonGroup();
methodGroup.add(approxGeneClustering);
methodGroup.add(exactGeneClustering);
methodGroup.add(exactGeneClusteringGPU);

JLabel methodNextLabel = new JLabel("Input parameters");
methodNextLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton methodNext = new JButton();
methodNext.setIcon(new ImageIcon(nextButtonIcon));
methodNext.setContentAreaFilled(false);
methodNext.setRolloverIcon(new ImageIcon(nextButtonRolloverIcon));

GroupLayout chooseMethodLayout = new GroupLayout(method);
method.setLayout(chooseMethodLayout);
chooseMethodLayout.setAutoCreateGaps(true);
chooseMethodLayout.setAutoCreateContainerGaps(true);

chooseMethodLayout.setHorizontalGroup(
        chooseMethodLayout.createSequentialGroup()
        .addGap(55)
        .addGroup(chooseMethodLayout.createParallelGroup(GroupLayout.Alignment.
            TRAILING)
                        .addComponent(chooseMethodStep))
        .addGroup(chooseMethodLayout.createParallelGroup(GroupLayout.Alignment.
            LEADING)
                        .addComponent(chooseMethodTitle)
                        .addComponent(gpuNotification)
                        .addComponent(gpuAdv)
                        .addComponent(approxGeneClustering, 0, GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(exactGeneClustering, 0, GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(exactGeneClusteringGPU, 0, GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE))
        .addComponent(methodNextLabel)
        .addComponent(methodNext)
);

chooseMethodLayout.setVerticalGroup(
        chooseMethodLayout.createSequentialGroup()
        .addGap(135)
        .addGroup(chooseMethodLayout.createParallelGroup(GroupLayout.Alignment.
            CENTER)
                        .addComponent(chooseMethodStep)
                        .addComponent(chooseMethodTitle))
        .addComponent(gpuNotification)
        .addComponent(gpuAdv)
        .addGap(10)
        .addComponent(approxGeneClustering)
        .addComponent(exactGeneClustering)
        .addComponent(exactGeneClusteringGPU)
        .addContainerGap(300, 350)
        .addGroup(chooseMethodLayout.createParallelGroup(GroupLayout.Alignment.
            CENTER)
                        .addComponent(methodNextLabel)
                        .addComponent(methodNext))
);

method.add(chooseMethodStep);
method.add(chooseMethodTitle);
method.add(gpuNotification);
method.add(gpuAdv);
method.add(approxGeneClustering);
method.add(exactGeneClustering);
method.add(exactGeneClusteringGPU);
method.add(methodNextLabel);
method.add(methodNext);
method.setBackground(Color.WHITE);


// Items for card 4 -- input parameters view
input = new JPanel();
```

```
BufferedImage inputParamsIcon = null;
try {
        inputParamsIcon = ImageIO.read(getClass().getResource("/images/step4.png")
            );
} catch (IOException e1) {
        e1.printStackTrace();
}
JLabel inputParamsStep = new JLabel(new ImageIcon(inputParamsIcon));
JLabel inputParamsTitle = new JLabel("Enter input parameters");
inputParamsTitle.setFont(new Font(font, Font.PLAIN, 24));
JLabel sizeOfGeneClusterLabel = new JLabel("Size of gene cluster: ");
sizeOfGeneClusterLabel.setFont(new Font(font, Font.PLAIN, 18));
sizeOfGeneClusterField = new JTextField("2", 25);
sizeOfGeneClusterField.setMinimumSize(new Dimension(50, 25));
sizeOfGeneClusterField.setFont(new Font(font, Font.PLAIN, 17));
JLabel missingWeightLabel = new JLabel("Weight of missing genes*: ");
missingWeightLabel.setFont(new Font(font, Font.PLAIN, 18));
missingWeightField = new JTextField("1", 25);
missingWeightField.setMinimumSize(new Dimension(50, 25));
missingWeightField.setFont(new Font(font, Font.PLAIN, 17));
JLabel additionalWeightLabel = new JLabel("Weight of additional genes*: ");
additionalWeightLabel.setFont(new Font(font, Font.PLAIN, 18));
additionalWeightField = new JTextField("1", 25);
additionalWeightField.setMinimumSize(new Dimension(50, 25));
additionalWeightField.setFont(new Font(font, Font.PLAIN, 17));
JLabel numOfCandidateGeneClustersLabel = new JLabel("Number of candidate gene
    clusters to be displayed: ");
numOfCandidateGeneClustersLabel.setFont(new Font(font, Font.PLAIN, 18));
numOfCandidateGeneClustersField = new JTextField("10", 25);
numOfCandidateGeneClustersField.setMinimumSize(new Dimension(50, 25));
numOfCandidateGeneClustersField.setFont(new Font(font, Font.PLAIN, 17));
JLabel weightNotice = new JLabel("*Specify integer weights only");
weightNotice.setFont(new Font(font, Font.ITALIC, 13));

JLabel inputNextLabel = new JLabel("Find gene clusters");
inputNextLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton inputNext = new JButton();
inputNext.setIcon(new ImageIcon(nextButtonIcon));
inputNext.setContentAreaFilled(false);
inputNext.setRolloverIcon(new ImageIcon(nextButtonRolloverIcon));

input.add(inputParamsStep);
input.add(inputParamsTitle);
input.add(sizeOfGeneClusterLabel);
input.add(sizeOfGeneClusterField);
input.add(numOfCandidateGeneClustersLabel);
input.add(numOfCandidateGeneClustersField);
input.add(inputNextLabel);
input.add(inputNext);
input.setBackground(Color.WHITE);

// Items for card 4 -- processing algorithm view
process = new JPanel();
JLabel processing = new JLabel("Processing data...");
JButton printResults = new JButton("Print results");
process.add(processing);
process.add(printResults);

// Items for card 5 -- displaying candidate gene clusters algorithm view
results = new JPanel();

BufferedImage displayResultsIcon = null;
try {
        displayResultsIcon = ImageIO.read(getClass().getResource("/images/step5.
            png"));
} catch (IOException e1) {
        e1.printStackTrace();
}
JLabel displayResultsStep = new JLabel(new ImageIcon(displayResultsIcon));
JLabel displayResultsTitle = new JLabel("Overview of results");
displayResultsTitle.setFont(new Font(font, Font.PLAIN, 24));

JTable geneClusTable = new JTable();
geneClusTable.setPreferredScrollableViewportSize(new Dimension(1200, 1000));
geneClusTable.doLayout();
geneClusTable.setAutoResizeMode(JTable.AUTO_RESIZE_LAST_COLUMN);
geneClusTable.setBorder(null);
geneClusTable.setFillsViewportHeight(true);
geneClusTable.setGridColor(Color.WHITE);
geneClusTable.setFont(new Font(font, Font.PLAIN, 15));
geneClusTable.setRowHeight(35);
geneClusTable.setCellSelectionEnabled(false);
geneClusTable.setFocusable(false);

JScrollPane listOfGeneClustersPanel = new JScrollPane(geneClusTable,
    ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED, ScrollPaneConstants.
    HORIZONTAL_SCROLLBAR_AS_NEEDED);
listOfGeneClustersPanel.getViewport().setOpaque(false);
listOfGeneClustersPanel.setOpaque(false);
listOfGeneClustersPanel.setBorder(new EmptyBorder(2, 2, 2, 2));
listOfGeneClustersPanel.setPreferredSize(new Dimension(1200, 1000));
```

```
JTableHeader header = geneClusTable.getTableHeader();
header.setDefaultRenderer(new HeaderRenderer(geneClusTable));
header.setFont(new Font(font, Font.BOLD, 16));
header.setEnabled(false);
header.setFocusable(false);
header.setPreferredSize(new Dimension(listOfGeneClustersPanel.getWidth(), 35));

DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
centerRenderer.setHorizontalAlignment(JLabel.CENTER);
geneClusTable.setDefaultRenderer(Integer.class, centerRenderer);
geneClusTable.setDefaultRenderer(String.class, centerRenderer);

JButton generatePdf = new JButton("Generate report");
generatePdf.setFont(new Font(font, Font.PLAIN, 14));
generatePdf.setMinimumSize(new Dimension(50, 35));
JButton viewAdjacencyMatrix = new JButton("View adjacency matrix");
viewAdjacencyMatrix.setFont(new Font(font, Font.PLAIN, 14));
viewAdjacencyMatrix.setMinimumSize(new Dimension(50, 35));
viewAdjacencyMatrix.setLayout(new BorderLayout(0, 5));
JButton viewGraph = new JButton("View graph");
viewGraph.setFont(new Font(font, Font.PLAIN, 14));
viewGraph.setMinimumSize(new Dimension(50, 35));
JButton viewAlignment = new JButton("View alignment of genes");
viewAlignment.setMinimumSize(new Dimension(50, 35));
viewAlignment.setFont(new Font(font, Font.PLAIN, 14));

JLabel resultsNextLabel = new JLabel("Home");
resultsNextLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton resultsNext = new JButton();
resultsNext.setIcon(new ImageIcon(nextButtonIcon));
resultsNext.setContentAreaFilled(false);
resultsNext.setRolloverIcon(new ImageIcon(nextButtonRolloverIcon));

GroupLayout displayResultsLayout = new GroupLayout(results);
results.setLayout(displayResultsLayout);
displayResultsLayout.setAutoCreateGaps(true);
displayResultsLayout.setAutoCreateContainerGaps(true);

displayResultsLayout.setHorizontalGroup(
        displayResultsLayout.createSequentialGroup()
        .addGap(55)
        .addGroup(displayResultsLayout.createParallelGroup(GroupLayout.Alignment.
            TRAILING)
                .addComponent(displayResultsStep))
        .addGroup(displayResultsLayout.createParallelGroup(GroupLayout.Alignment.
            LEADING)
                .addComponent(displayResultsTitle)
                .addGroup(displayResultsLayout.createParallelGroup(GroupLayout.
                    Alignment.TRAILING)
                                .addComponent(listOfGeneClustersPanel, 0,
                                    GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(resultsNextLabel))
                .addGroup(displayResultsLayout.createSequentialGroup()
                                .addComponent(generatePdf, 0, GroupLayout.
                                    DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(viewAdjacencyMatrix, 0, GroupLayout.
                                    DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(viewGraph, 0, GroupLayout.
                                    DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(viewAlignment, 0, GroupLayout.
                                    DEFAULT_SIZE, Short.MAX_VALUE)))
        .addComponent(resultsNext)
);

displayResultsLayout.setVerticalGroup(
        displayResultsLayout.createSequentialGroup()
        .addGap(30)
        .addGroup(displayResultsLayout.createParallelGroup(GroupLayout.Alignment.
            CENTER)
                .addComponent(displayResultsStep)
                .addComponent(displayResultsTitle))
        .addGap(10)
        .addGroup(displayResultsLayout.createParallelGroup(GroupLayout.Alignment.
            CENTER)
                        .addComponent(generatePdf)
                        .addComponent(viewAdjacencyMatrix)
                        .addComponent(viewGraph)
                        .addComponent(viewAlignment))
        .addGap(10)
        .addComponent(listOfGeneClustersPanel, 0, GroupLayout.DEFAULT_SIZE, Short.
            MAX_VALUE)
        .addContainerGap(300, 350)
        .addGroup(displayResultsLayout.createParallelGroup(GroupLayout.Alignment.
            CENTER)
                .addComponent(resultsNextLabel)
                .addComponent(resultsNext))
);

results.add(displayResultsStep);
results.add(displayResultsTitle);
results.add(generatePdf);
results.add(listOfGeneClustersPanel);
```

```
results.add(viewAdjacencyMatrix);
results.add(viewGraph);
results.add(viewAlignment);
results.add(resultsNextLabel);
results.add(resultsNext);
results.setBackground(Color.WHITE);

// Items for card 7 -- displaying adjacency matrix view
adjMat = new JPanel();
JLabel displayAdjMatTitle = new JLabel("Adjacency matrix");
displayAdjMatTitle.setFont(new Font(font, Font.PLAIN, 24));
JTextArea matrixTextArea = new JTextArea();
matrixTextArea.setEditable(false);
matrixTextArea.setFont(new Font("PT Sans", Font.PLAIN, 13));
JScrollPane adjacencyMatrixPanel = new JScrollPane(matrixTextArea,
     ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED, ScrollPaneConstants.
     HORIZONTAL_SCROLLBAR_AS_NEEDED);
adjacencyMatrixPanel.setPreferredSize(new Dimension(width - 150, height));
adjacencyMatrixPanel.setBorder(BorderFactory.createEmptyBorder());

JLabel adjMatBackLabel = new JLabel("Overview of results");
adjMatBackLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton adjMatBack = new JButton();
adjMatBack.setIcon(new ImageIcon(backButtonIcon));
adjMatBack.setContentAreaFilled(false);
adjMatBack.setRolloverIcon(new ImageIcon(backButtonRolloverIcon));

GroupLayout adjMatLayout = new GroupLayout(adjMat);
adjMat.setLayout(adjMatLayout);
adjMatLayout.setAutoCreateGaps(true);
adjMatLayout.setAutoCreateContainerGaps(true);

adjMatLayout.setHorizontalGroup(
        adjMatLayout.createSequentialGroup()
        .addComponent(adjMatBack)
        .addGroup(adjMatLayout.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addComponent(displayAdjMatTitle)
                .addComponent(adjacencyMatrixPanel)
                .addComponent(adjMatBackLabel))
        .addContainerGap(150, 150)
);

adjMatLayout.setVerticalGroup(
        adjMatLayout.createSequentialGroup()
        .addGap(30)
        .addGroup(adjMatLayout.createParallelGroup(GroupLayout.Alignment.CENTER)
                .addComponent(displayAdjMatTitle))
        .addGap(20)
        .addComponent(adjacencyMatrixPanel)
        .addGap(50)
        .addGroup(adjMatLayout.createParallelGroup(GroupLayout.Alignment.CENTER)
                .addComponent(adjMatBack)
                .addComponent(adjMatBackLabel))
        .addContainerGap(50, 50)
);

adjMat.add(displayAdjMatTitle);
adjMat.add(adjacencyMatrixPanel);
adjMat.add(adjMatBack);
adjMat.add(adjMatBackLabel);
adjMat.setBackground(Color.WHITE);


// Items for card 8 -- displaying graph representation view
graph = new JPanel();
JLabel displayGraphTitle = new JLabel("Graph representation");
displayGraphTitle.setFont(new Font(font, Font.PLAIN, 24));

JPanel graphPanel = new JPanel();
graphPanel.setPreferredSize(new Dimension(width, height));

JLabel displayGraphBackLabel = new JLabel("Overview of results");
displayGraphBackLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton displayGraphBack = new JButton();
displayGraphBack.setIcon(new ImageIcon(backButtonIcon));
displayGraphBack.setContentAreaFilled(false);
displayGraphBack.setRolloverIcon(new ImageIcon(backButtonRolloverIcon));

GroupLayout graphLayout = new GroupLayout(graph);
graph.setLayout(graphLayout);
graphLayout.setAutoCreateGaps(true);
graphLayout.setAutoCreateContainerGaps(true);

graphLayout.setHorizontalGroup(
        graphLayout.createSequentialGroup()
        .addComponent(displayGraphBack)
        .addGroup(graphLayout.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addComponent(displayGraphTitle)
                .addComponent(graphPanel, 0, GroupLayout.DEFAULT_SIZE, Short.
                     MAX_VALUE)
                .addComponent(displayGraphBackLabel))
        .addGap(100)
```

```
);

graphLayout.setVerticalGroup(
        graphLayout.createSequentialGroup()
        .addGap(30)
        .addGroup(graphLayout.createParallelGroup(GroupLayout.Alignment.CENTER)
                .addComponent(displayGraphTitle))
        .addGap(20)
        .addComponent(graphPanel)
        .addGap(20)
        .addGroup(graphLayout.createParallelGroup(GroupLayout.Alignment.CENTER)
                .addComponent(displayGraphBack)
                .addComponent(displayGraphBackLabel))
        .addContainerGap(50, 50)
);

graph.add(displayGraphTitle);
graph.add(displayGraphBack);
graph.add(displayGraphBackLabel);
graph.setBackground(Color.WHITE);

// Items for card 9 -- displaying alignment view
alignment = new JPanel();
JLabel displayAlignmentTitle = new JLabel("Alignment of genes");
displayAlignmentTitle.setFont(new Font(font, Font.PLAIN, 24));
JComboBox<String> listOfCandidateGeneClusters = new JComboBox<String>();
listOfCandidateGeneClusters.setFont(new Font(font, Font.PLAIN, 15));

/** START OF TABLE **/
alignmentTable = new JTable();
alignmentTable.setPreferredScrollableViewportSize(new Dimension(1200, 1000));
alignmentTable.doLayout();
alignmentTable.setAutoResizeMode(JTable.AUTO_RESIZE_LAST_COLUMN);
alignmentTable.setBorder(null);
alignmentTable.setFillsViewportHeight(true);
alignmentTable.setGridColor(Color.WHITE);
alignmentTable.setFont(new Font(font, Font.PLAIN, 15));
alignmentTable.setRowHeight(35);
alignmentTable.setCellSelectionEnabled(false);
alignmentTable.setFocusable(false);
alignmentTable.setTableHeader(null);
alignmentTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

JScrollPane alignmentResultsPanel = new JScrollPane(alignmentTable,
     ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED, ScrollPaneConstants.
     HORIZONTAL_SCROLLBAR_AS_NEEDED);
alignmentResultsPanel.getViewport().setOpaque(false);
alignmentResultsPanel.setOpaque(false);
alignmentResultsPanel.setBorder(new EmptyBorder(2, 2, 2, 2));
alignmentResultsPanel.setPreferredSize(new Dimension(width - 150, height));

DefaultTableCellRenderer centerRenderer_alignment = new DefaultTableCellRenderer()
     ;
centerRenderer_alignment.setHorizontalAlignment(JLabel.CENTER);
alignmentTable.setDefaultRenderer(Integer.class, centerRenderer_alignment);
alignmentTable.setDefaultRenderer(String.class, centerRenderer_alignment);
/** END OF TABLE **/

JLabel alignmentBackLabel = new JLabel("Overview of results");
alignmentBackLabel.setFont(new Font(font, Font.PLAIN, 15));
JButton alignmentBack = new JButton();
alignmentBack.setIcon(new ImageIcon(backButtonIcon));
alignmentBack.setContentAreaFilled(false);
alignmentBack.setRolloverIcon(new ImageIcon(backButtonRolloverIcon));

GroupLayout displayAlignmentLayout = new GroupLayout(alignment);
alignment.setLayout(displayAlignmentLayout);
displayAlignmentLayout.setAutoCreateGaps(true);
displayAlignmentLayout.setAutoCreateContainerGaps(true);

displayAlignmentLayout.setHorizontalGroup(
        displayAlignmentLayout.createSequentialGroup()
        .addComponent(alignmentBack)
        .addGroup(displayAlignmentLayout.createParallelGroup(GroupLayout.Alignment
             .LEADING)
                .addComponent(displayAlignmentTitle)
                .addComponent(listOfCandidateGeneClusters)
                .addComponent(alignmentResultsPanel)
                .addComponent(alignmentBackLabel))
        .addContainerGap(150, 150)
);

displayAlignmentLayout.setVerticalGroup(
                displayAlignmentLayout.createSequentialGroup()
                .addGap(30)
                .addGroup(displayAlignmentLayout.createParallelGroup(GroupLayout.
                     Alignment.CENTER)
                        .addComponent(displayAlignmentTitle))
                .addGap(20)
                .addComponent(listOfCandidateGeneClusters)
                .addGap(15)
                .addComponent(alignmentResultsPanel)
```

```
                .addContainerGap(400, 500)
                .addGroup(displayAlignmentLayout.createParallelGroup(GroupLayout.
                    Alignment.CENTER)
                        .addComponent(alignmentBack)
                        .addComponent(alignmentBackLabel))
                .addContainerGap(50, 50)
);

alignment.add(displayAlignmentTitle);
alignment.add(listOfCandidateGeneClusters);
alignment.add(alignmentResultsPanel);
alignment.add(alignmentBack);
alignment.add(alignmentBackLabel);
alignment.setBackground(Color.WHITE);

cards.add(main, "0");
cards.add(upload, "1");
cards.add(dataset, "2");
cards.add(method, "3");
cards.add(input, "4");
cards.add(process, "5");
cards.add(results, "6");
cards.add(adjMat, "7");
cards.add(graph, "8");
cards.add(alignment, "9");
cards.add(help, "10");
cards.add(progress, "11");
cl.show(cards, "0");

// buttons
startBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                cl.show(cards, "1");
        }
});

helpBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                cl.show(cards, "10");
        }
});

helpBack.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                cl.show(cards, "0");
        }
});

uploadButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                fc = new JFileChooser();
                fc.setFileFilter(filter);
                int returnVal = fc.showOpenDialog(upload);

                if(returnVal == JFileChooser.APPROVE_OPTION) {
                        file = fc.getSelectedFile();
                        fileName.setText(file.getName());
                        System.out.println("Opened " + file.getName());
                }
        }
});

enterData.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                boolean validInput = false;
                if (file == null) {
                        JOptionPane.showMessageDialog(frame, "No file uploaded!",
                            "Error", JOptionPane.ERROR_MESSAGE);
            } else if (!file.getName().toLowerCase().endsWith(".csv")) {
                JOptionPane.showMessageDialog(frame, "You have uploaded an invalid
                      file type.", "Error", JOptionPane.ERROR_MESSAGE);
            } else {
                try {
                                minVertexSizeVal = Integer.parseInt(minVertexSize.
                                    getText());
                                maxVertexSizeVal = Integer.parseInt(maxVertexSize.
                                    getText());
                                validInput = true;
                        } catch (NumberFormatException nfe) {
                                validInput = false;
                                cl.show(cards, "1");
                                JOptionPane.showMessageDialog(frame, "Invalid
                                    input format", "Error", JOptionPane.
                                    ERROR_MESSAGE);
                        }

                        if(minVertexSizeVal < 1 || maxVertexSizeVal < 1) {
                                cl.show(cards, "1");
                                JOptionPane.showMessageDialog(frame, "Size of gene
                                    contents should be greater than 0.", "Error",
                                    JOptionPane.ERROR_MESSAGE);
                                validInput = false;
```

```
                        }

                        if(validInput) {
                                cl.show(cards, "11");
                        uploadClicked(progressLbl);
                        }
                }
        }
});

listOfGenomes.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                genesIntRepresentation.setText("");
                genesIntegerForm = gc.getAllGenesIntegerForm();
                int indexOfGenome = genomes.indexOf(listOfGenomes.getSelectedItem
                    ());
                for(int i = 0; i < genesIntegerForm.get(indexOfGenome).size(); i
                    ++) {
                        genesIntRepresentation.append(genesIntegerForm.get(
                            indexOfGenome).get(i) + " ");
                }
        }
});

displayNext.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                int numOfGPU = gc.getNumOfGPUAvailable();
                if(numOfGPU == 0) {
                        gpuNotification.setText("No GPU device is available.");
                        exactGeneClustering.setEnabled(false);
                } else {
                        gpuNotification.setText(numOfGPU + " GPU device/s found.")
                            ;
                }
                cl.show(cards, "3");
        }
});

methodNext.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                if(exactGeneClustering.isSelected() || exactGeneClusteringGPU.
                    isSelected()) {
                        if(exactGeneClustering.isSelected()) {
                                typeOfGeneClustering = 0;
                        } else {
                                typeOfGeneClustering = 2;
                        }

                        cl.show(cards, "4");

                        // GroupLayout for Exact Gene Clustering View
                        GroupLayout inputParamsExactLayout = new GroupLayout(input
                            );
                        inputParamsExactLayout.setAutoCreateGaps(true);
                        inputParamsExactLayout.setAutoCreateContainerGaps(true);

                        inputParamsExactLayout.setHorizontalGroup(
                                inputParamsExactLayout.createSequentialGroup()
                                .addGap(55)
                                .addGroup(inputParamsExactLayout.
                                    createParallelGroup(GroupLayout.Alignment.
                                    TRAILING)
                                                .addComponent(inputParamsStep))
                                .addGroup(inputParamsExactLayout.
                                    createParallelGroup(GroupLayout.Alignment.
                                    LEADING)
                                        .addComponent(inputParamsTitle)
                                        .addComponent(
                                            numOfCandidateGeneClustersLabel))
                                .addGap(50)
                                .addComponent(numOfCandidateGeneClustersField)
                                .addComponent(inputNextLabel)
                                .addComponent(inputNext)
                        );

                        inputParamsExactLayout.setVerticalGroup(
                                inputParamsExactLayout.createSequentialGroup()
                                .addGap(135)
                                .addGroup(inputParamsExactLayout.
                                    createParallelGroup(GroupLayout.Alignment.
                                    CENTER)
                                        .addComponent(inputParamsStep)
                                        .addComponent(inputParamsTitle))
                                .addGap(20)
                                .addGroup(inputParamsExactLayout.
                                    createParallelGroup(GroupLayout.Alignment.
                                    BASELINE)
                                        .addComponent(
                                            numOfCandidateGeneClustersLabel)
                                        .addComponent(
                                            numOfCandidateGeneClustersField))
```

```
                .addContainerGap(400, 450)
                .addGroup(inputParamsExactLayout.
                    createParallelGroup(GroupLayout.Alignment.
                    CENTER)
                        .addComponent(inputNextLabel)
                        .addComponent(inputNext))
            );

            input.setLayout(inputParamsExactLayout);
            input.remove(sizeOfGeneClusterLabel);
            input.remove(sizeOfGeneClusterField);
            input.remove(missingWeightLabel);
            input.remove(missingWeightField);
            input.remove(additionalWeightLabel);
            input.remove(additionalWeightField);
        }
        else if(approxGeneClustering.isSelected()) {
            typeOfGeneClustering = 1;
            cl.show(cards, "4");

            // GroupLayout for Approximate Gene Clustering View
            GroupLayout inputParamsApproxLayout = new GroupLayout(
                input);
            inputParamsApproxLayout.setAutoCreateGaps(true);
            inputParamsApproxLayout.setAutoCreateContainerGaps(true);

            inputParamsApproxLayout.setHorizontalGroup(
                    inputParamsApproxLayout.createSequentialGroup()
                    .addGap(55)
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        TRAILING)
                                .addComponent(inputParamsStep))
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        LEADING)
                                .addComponent(inputParamsTitle)
                                .addComponent(
                                    sizeOfGeneClusterLabel)
                                .addComponent(missingWeightLabel)
                                .addComponent(
                                    additionalWeightLabel)
                                .addComponent(
                                    numOfCandidateGeneClustersLabel
                                    )
                                .addComponent(weightNotice))
                    .addGap(50)
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        LEADING)
                                .addComponent(
                                    sizeOfGeneClusterField)
                                .addComponent(missingWeightField)
                                .addComponent(
                                    additionalWeightField)
                                .addComponent(
                                    numOfCandidateGeneClustersField
                                    ))
                    .addComponent(inputNextLabel)
                    .addComponent(inputNext)
            );

            inputParamsApproxLayout.setVerticalGroup(
                    inputParamsApproxLayout.createSequentialGroup()
                    .addGap(135)
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        CENTER)
                                .addComponent(inputParamsStep)
                                .addComponent(inputParamsTitle))
                    .addGap(20)
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        BASELINE)
                                .addComponent(
                                    sizeOfGeneClusterLabel)
                                .addComponent(
                                    sizeOfGeneClusterField))
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        BASELINE)
                                .addComponent(missingWeightLabel)
                                .addComponent(missingWeightField))
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
                        BASELINE)
                                .addComponent(
                                    additionalWeightLabel)
                                .addComponent(
                                    additionalWeightField))
                    .addGroup(inputParamsApproxLayout.
                        createParallelGroup(GroupLayout.Alignment.
```

```
                                        BASELINE)
                                                .addComponent(
                                                        numOfCandidateGeneClustersLabel
                                                        )
                                                .addComponent(
                                                        numOfCandidateGeneClustersField
                                                        ))
                                        .addComponent(weightNotice)
                                        .addContainerGap(400, 450)
                                        .addGroup(inputParamsApproxLayout.
                                                createParallelGroup(GroupLayout.Alignment.
                                                CENTER)
                                                .addComponent(inputNextLabel)
                                                .addComponent(inputNext))
                                );

                                input.setLayout(inputParamsApproxLayout);
                                input.add(missingWeightLabel);
                                input.add(missingWeightField);
                                input.add(additionalWeightLabel);
                                input.add(additionalWeightField);
                                input.add(weightNotice);
                        }
                }
        });

        inputNext.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        cl.show(cards, "11");

                        if(typeOfGeneClustering == 1) {
                                boolean validInput = false;
                                missing = 1;
                                additional = 1;
                                sizeOfGeneCluster = 2;
                                numOfCandidateGeneClusters = 10;

                                try {
                                        missing = Integer.parseInt(missingWeightField.
                                                getText());
                                        additional = Integer.parseInt(
                                                additionalWeightField.getText());
                                        numOfCandidateGeneClusters = Integer.parseInt(
                                                numOfCandidateGeneClustersField.getText());
                                        sizeOfGeneCluster = Integer.parseInt(
                                                sizeOfGeneClusterField.getText());
                                        validInput = true;
                                } catch (NumberFormatException nfe) {
                                        validInput = false;
                                        cl.show(cards, "4");
                                        JOptionPane.showMessageDialog(frame, "Invalid
                                                input format", "Error", JOptionPane.
                                                ERROR_MESSAGE);
                                }

                                if(sizeOfGeneCluster < 1) {
                                        cl.show(cards, "4");
                                        JOptionPane.showMessageDialog(frame, "Size of gene
                                                cluster should not be negative", "Error",
                                                JOptionPane.ERROR_MESSAGE);
                                        validInput = false;

                                }

                                if(numOfCandidateGeneClusters < 1) {
                                        cl.show(cards, "4");
                                        JOptionPane.showMessageDialog(frame, "Number of
                                                candidate gene clusters should be at least
                                                1.", "Error", JOptionPane.ERROR_MESSAGE);
                                        validInput = false;

                                }

                                if(validInput) {
                                        resultsApproxClicked(progressLbl, geneClusTable,
                                                viewAlignment);
                                }
                        } else if(typeOfGeneClustering == 0){
                                boolean validInput = false;
                                numOfCandidateGeneClusters = 10;

                                try {
                                        numOfCandidateGeneClusters = Integer.parseInt(
                                                numOfCandidateGeneClustersField.getText());
                                        validInput = true;
                                } catch (NumberFormatException nfe) {
                                        validInput = false;
                                        JOptionPane.showMessageDialog(frame, "Invalid
                                                input format", "Error", JOptionPane.
                                                ERROR_MESSAGE);
                                }
```

```
                        if(numOfCandidateGeneClusters < 1) {
                                cl.show(cards, "4");
                                JOptionPane.showMessageDialog(frame, "Number of
                                    candidate gene clusters should be at least
                                    1.", "Error", JOptionPane.ERROR_MESSAGE);
                                validInput = false;

                        }

                        if(validInput) {
                                resultsExactClicked(progressLbl, geneClusTable);
                        }
                } else {
                        boolean validInput = false;
                        numOfCandidateGeneClusters = 10;

                        try {
                                numOfCandidateGeneClusters = Integer.parseInt(
                                    numOfCandidateGeneClustersField.getText());
                                validInput = true;
                        } catch (NumberFormatException nfe) {
                                validInput = false;
                                JOptionPane.showMessageDialog(frame, "Invalid
                                    input format", "Error", JOptionPane.
                                    ERROR_MESSAGE);
                        }

                        if(numOfCandidateGeneClusters < 1) {
                                cl.show(cards, "4");
                                JOptionPane.showMessageDialog(frame, "Number of
                                    candidate gene clusters should be at least
                                    1.", "Error", JOptionPane.ERROR_MESSAGE);
                                validInput = false;

                        }

                        if(validInput) {
                                ExactGeneClustering exact = new
                                    ExactGeneClustering(gc.getVertices(),
                                    numOfCandidateGeneClusters);
                                try {
                                        exact.allocateHostInputData();
                                } catch (IOException e1) {
                                        e1.printStackTrace();
                                }

                                if(exact.getIsDone()) {
                                        sortedGeneClusters = exact.
                                            getSortedGeneClusters();
                                        gc.renameGenes(sortedGeneClusters);

                                        GeneClusTableModel gcTableModel = new
                                            GeneClusTableModel();
                                        for(int i = 1; i <= sortedGeneClusters.
                                            size(); i++) {
                                                ArrayList<Vertex> clique =
                                                    sortedGeneClusters.get(i-1);
                                                String data = "";
                                                for(Vertex v : clique) {
                                                        data += v.
                                                            getGeneClusterName() +
                                                            " ";
                                                }
                                                gcTableModel.addData(i, data);
                                        }
                                        geneClusTable.setModel(gcTableModel);
                                        geneClusTable.getColumnModel().getColumn
                                            (0).setMaxWidth(300);
                                        cl.show(cards, "6");
                                }
                        }
                }
        }
});

generatePdf.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                sfc = new JFileChooser();
                int returnVal = sfc.showSaveDialog(generatePdf);

                if(returnVal == JFileChooser.APPROVE_OPTION) {
                        File file_save = sfc.getSelectedFile();
                        if (file_save == null) {
                                return;
                    }
                    if (!file_save.getName().toLowerCase().endsWith(".pdf")) {
                        file_save = new File(file_save.getParentFile(), file_save.
                            getName() + ".pdf");
                    }
                        new PDFManager(baseGeneClusters, file_save,
                            typeOfGeneClustering, genomes, sizeOfGeneCluster,
                            missing, additional, numOfCandidateGeneClusters,
```

```
                                        sortedGeneClusters, gc);
                        }
                }
        });

        viewAdjacencyMatrix.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        cl.show(cards, "11");

                        adjMatrixClicked(progressLbl, matrixTextArea);
                }

        });

        adjMatBack.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        matrixTextArea.setText("");
                        cl.show(cards, "6");
                }
        });

        viewGraph.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        cl.show(cards, "11");
                        graphClicked(progressLbl, graphPanel);
                }
        });

        displayGraphBack.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        graphPanel.removeAll();
                        cl.show(cards, "6");
                }
        });

        viewAlignment.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        if(typeOfGeneClustering == 1) {
                                for(TreeSet<String> baseGeneCluster : baseGeneClusters) {
                                        listOfCandidateGeneClusters.addItem(
                                                baseGeneCluster.toString());
                                }
                                alignmentTable.repaint();
                                alignmentResultsPanel.repaint();
                                alignment.repaint();
                                cards.repaint();
                        } else {
                                for(int i = 1; i <= sortedGeneClusters.size(); i++) {
                                        listOfCandidateGeneClusters.addItem("Clique #" + i
                                                );
                                }
                                alignmentTable.repaint();
                                alignmentResultsPanel.repaint();
                                alignment.repaint();
                                cards.repaint();
                        }
                        cl.show(cards, "9");
                }
        });

        // FIX THIS!!!!! ((Fixed as of 05/07/18))
        listOfCandidateGeneClusters.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        cl.show(cards, "11");
                        int indexOfBaseGeneCluster = listOfCandidateGeneClusters.
                                getSelectedIndex();
                        alignmentClicked(indexOfBaseGeneCluster, progressLbl,
                                alignmentTable, alignmentResultsPanel);
                        cl.show(cards, "9");
                }
        });

        alignmentBack.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        cl.show(cards, "6");
                }
        });

        resultsNext.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        cl.show(cards, "0");
                        init();
                }
        });

        frame.getContentPane().add(cards);
        frame.pack();
        frame.setVisible(true);
}

/**
 * Resets the fileName field
```

```
 */
public void resetUpload() {
        fileName.setText("");
        file = null;
}

/**
 * Resets the results
 */
public void resetResults() {
        // Reset data
        baseGeneClusters.clear();
        sortedGeneClusters.clear();
        sizeOfGeneCluster = 0;
        missing = 0;
        additional = 0;
        numOfCandidateGeneClusters = 0;
}

/**
 * Initializes the variables to show a new window
 */
public void init() {
        genomes.clear();
        genesIntegerForm.clear();
        baseGeneClusters.clear();
        sortedGeneClusters.clear();
        sizeOfGeneCluster = 0;
        missing = 0;
        additional = 0;
        numOfCandidateGeneClusters = 0;

        try {
                new Canvas(ImageIO.read(getClass().getResource("/images/bg2.png")));
                frame.dispose();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

/**
 * Runs the background process for parsing the dataset
 */
public void uploadClicked(JLabel progressLbl) {
        progressBar.setVisible(true);
    progressBar.setIndeterminate(true);
    progressLbl.setText("Processing dataset...");
        class UploadTask extends SwingWorker<Void, Void> {
             protected Void doInBackground() {
               gc.parseDataset(file, minVertexSizeVal, maxVertexSizeVal);
               return null;
             }

             protected void done() {
                progressBar.setVisible(false);
                cl.show(cards, "2");
                        int numOfGenomes = gc.getNumberOfGenomes();
                        genomeStats.setText("You uploaded " + numOfGenomes + " genomes.");
                        genomes = gc.getAllGenomes();
                        for(String genome : genomes) {
                                listOfGenomes.addItem(genome);
                        }
             }
           }
           new UploadTask().execute();
}

/**
 * Runs the background process for running the approximate gene clustering algorithm
 */
public void resultsApproxClicked(JLabel progressLbl, JTable geneClusTable, JButton
     viewAlignment) {
        progressBar.setVisible(true);
    progressBar.setIndeterminate(true);
    progressLbl.setText("Detecting gene clusters...");
        class ResultsApproxTask extends SwingWorker<Void, Void> {
             protected Void doInBackground() {
                 approx = new ApproximateGeneClustering(gc, sizeOfGeneCluster, missing,
                      additional);
                 return null;
             }

             protected void done() {
                progressBar.setVisible(false);
                approx.computeTotalWeightsOfMinStars();
                        HashMap<ArrayList<Vertex>, Integer> starsAndWeights = new HashMap<
                             ArrayList<Vertex>, Integer>();
                        starsAndWeights = approx.getGeneratedStarsAndWeight();

                        // Start of AlgoUtils
                        AlgoUtils au = new AlgoUtils(starsAndWeights,
                             numOfCandidateGeneClusters, gc);
```

```
                        sortedGeneClusters = au.getSortedGeneClusters();
                        gc.renameGenes(sortedGeneClusters);
                        baseGeneClusters = gc.getBaseGeneClusters(sortedGeneClusters);

                        if(baseGeneClusters.size() == 0) {
                                JOptionPane.showMessageDialog(frame, "No base gene
                                     clusters found!");
                                viewAlignment.setEnabled(false);
                                cl.show(cards, "4");
                                resetResults();
                        } else {
                                viewAlignment.setEnabled(true);
                                GeneClusTableModel gcTableModel = new GeneClusTableModel()
                                     ;
                                for(int i = 1; i <= baseGeneClusters.size(); i++) {
                                        TreeSet<String> geneCluster = baseGeneClusters.get
                                             (i-1);
                                        gcTableModel.addData(i, geneCluster.toString());
                                }
                                geneClusTable.setModel(gcTableModel);
                                geneClusTable.getColumnModel().getColumn(0).setMaxWidth
                                     (300);
                                cl.show(cards, "6");
                        }
                }
            }
            new ResultsApproxTask().execute();
}

/**
 * Runs the background process for running the exact gene clustering algorithm
 */
public void resultsExactClicked(JLabel progressLbl, JTable geneClusTable) {
        progressBar.setVisible(true);
    progressBar.setIndeterminate(true);
    progressLbl.setText("Detecting gene clusters...");
        class ResultsExactTask extends SwingWorker<Void, Void> {
                ExactGeneClustering exact;

                protected Void doInBackground() {
                        exact = new ExactGeneClustering(gc.getVertices(),
                             numOfCandidateGeneClusters);
                        exact.runAlgorithm();
                        return null;
            }

            protected void done() {
                progressBar.setVisible(false);
                sortedGeneClusters = exact.getSortedGeneClusters();
                        gc.renameGenes(sortedGeneClusters);

                        GeneClusTableModel gcTableModel = new GeneClusTableModel();
                        for(int i = 1; i <= sortedGeneClusters.size(); i++) {
                                ArrayList<Vertex> clique = sortedGeneClusters.get(i-1);
                                String data = "";
                                for(Vertex v : clique) {
                                        data += v.getGeneClusterName() + " ";
                                }
                                gcTableModel.addData(i, data);
                        }
                        geneClusTable.setModel(gcTableModel);
                        geneClusTable.getColumnModel().getColumn(0).setMaxWidth(300);
                        cl.show(cards, "6");
            }
        }
        new ResultsExactTask().execute();
}

/**
 * Runs the background process for running the exact gene clustering algorithm using the
     GPU
 */
public void resultsExactGPUClicked(JLabel progressLbl, JTable geneClusTable) {
        progressBar.setVisible(true);
    progressBar.setIndeterminate(true);
    progressLbl.setText("Detecting gene clusters...");
        class ResultsExactGPUTask extends SwingWorker<Void, Void> {
                ExactGeneClustering exact;

                protected Void doInBackground() {
                        exact = new ExactGeneClustering(gc.getVertices(),
                             numOfCandidateGeneClusters);
                        try {
                                exact.allocateHostInputData();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }

                        sortedGeneClusters = exact.getSortedGeneClusters();
                        gc.renameGenes(sortedGeneClusters);

                        GeneClusTableModel gcTableModel = new GeneClusTableModel();
```

```
                        for(int i = 1; i <= sortedGeneClusters.size(); i++) {
                                ArrayList<Vertex> clique = sortedGeneClusters.get(i-1);
                                String data = "";
                                for(Vertex v : clique) {
                                        data += v.getGeneClusterName() + " ";
                                }
                                gcTableModel.addData(i, data);
                        }
                        geneClusTable.setModel(gcTableModel);
                        geneClusTable.getColumnModel().getColumn(0).setMaxWidth(300);
                        return null;
                }

                protected void done() {
                    progressBar.setVisible(false);

                        cl.show(cards, "6");
                }
        }
        new ResultsExactGPUTask().execute();
}

/**
 * Runs the background process for creating the adjacency matrix
 */
public void adjMatrixClicked(JLabel progressLbl, JTextArea matrixTextArea) {
        progressBar.setVisible(true);
    progressBar.setIndeterminate(true);
    progressLbl.setText("Computing adjacency matrix...");
        class AdjacencyMatrixTask extends SwingWorker<Void, Void> {

                protected Void doInBackground() {
                        AdjacencyMatrixManager adjMatManager;
                        if(typeOfGeneClustering == 1) {
                                adjMatManager = new AdjacencyMatrixManager(gc.getVertices
                                        (), missing, additional);
                        } else {
                                adjMatManager = new AdjacencyMatrixManager(gc.getVertices
                                        (), 1, 1);
                        }
                        int[][] adjacencyMatrix = adjMatManager.getAdjacencyMatrix();
                        HashMap<Integer, String> geneDictionary = gc.
                             getGeneDictionaryInverse();
                        adjMatManager.renameVerticesToGenes(geneDictionary);
                        matrixTextArea.append("\t");
                        for(int i = 0; i < adjacencyMatrix.length; i++) {
                                matrixTextArea.append("v" + i + "\t");
                        }
                        matrixTextArea.append("\n");
                        for(int i = 0; i < adjacencyMatrix.length; i++) {
                                matrixTextArea.append("v" + i + "\t");
                                for(int j = 0; j < adjacencyMatrix.length; j++) {
                                        matrixTextArea.append(adjacencyMatrix[i][j] + "\t
                                             ");
                                }
                                matrixTextArea.append("\n");
                        }

                        // Show table of vertices
                        gc.renameGenes(gc.getVertices());
                        adjMatManager.showTableOfVertices();

                        return null;
                }

                protected void done() {
                    progressBar.setVisible(false);
                        cl.show(cards, "7");
                }
        }
        new AdjacencyMatrixTask().execute();
}

/**
 * Runs the background process for creating the graph representation
 */
public void graphClicked(JLabel progressLbl, JPanel graphPanel) {
        progressBar.setVisible(true);
    progressBar.setIndeterminate(true);
    progressLbl.setText("Generating graph...");
        class GraphTask extends SwingWorker<Void, Void> {
                Graph graphOfGenes;

                protected Void doInBackground() {
                        GraphManager graphManager = new GraphManager(gc.getVertices());
                        graphOfGenes = graphManager.constructGraph();
                        Viewer viewer = new Viewer(graphOfGenes, Viewer.ThreadingModel.
                             GRAPH_IN_ANOTHER_THREAD);
                        ViewPanel viewPanel = viewer.addDefaultView(false);
                        viewPanel.setPreferredSize(new Dimension(graphPanel.getWidth(),
                             graphPanel.getHeight()));
                        viewer.enableAutoLayout();
```

```
                        graphPanel.setLayout(new BorderLayout());
                        graphPanel.add(viewPanel);

                        // Show table of vertices
                        gc.renameGenes(gc.getVertices());
                        graphManager.showTableOfVertices();

                        return null;
                }

                protected void done() {
                    progressBar.setVisible(false);
                    cl.show(cards, "8");
                }
            }
            new GraphTask().execute();
    }

    /**
     * Runs the background process for displaying the alignment of genes
     */
    public void alignmentClicked(int indexOfBaseGeneCluster, JLabel progressLbl, JTable
         alignmentTable, JScrollPane alignmentResultsPanel) {
            progressBar.setVisible(true);
        progressBar.setIndeterminate(true);
        progressLbl.setText("Listing alignment of genes...");
            class AlignmentTask extends SwingWorker<Void, Void> {
                ArrayList<String> commonElements;

                protected Void doInBackground() {
                        int maxNumOfCols = gc.getMaxNumOfColumns(sortedGeneClusters,
                             indexOfBaseGeneCluster) + 1;
                        AlignmentTableModel gcAlignmentTableModel = new
                             AlignmentTableModel(maxNumOfCols);
                        Vertex orig_v = sortedGeneClusters.get(indexOfBaseGeneCluster).get
                             (0);
                        ArrayList<String> blockOfGenesOrig = gc.getBlockOfGenes(orig_v.
                             getPartitionNumber(), orig_v.getStartIndex(), orig_v.
                             getEndIndex());
                        commonElements = new ArrayList<String>(blockOfGenesOrig);

                        for(int i = 0; i < gc.getAllGenomes().size(); i++) {
                                ArrayList<String> rowData = new ArrayList<String>();
                                rowData.add(gc.getAllGenomes().get(i));

                                for(int j = 0; j < sortedGeneClusters.get(
                                     indexOfBaseGeneCluster).size(); j++) {
                                        Vertex v = sortedGeneClusters.get(
                                             indexOfBaseGeneCluster).get(j);
                                        if(v.getPartitionNumber() == i) {
                                                ArrayList<String> blockOfGenes = gc.
                                                     getBlockOfGenes(v.getPartitionNumber()
                                                     , v.getStartIndex(), v.getEndIndex());
                                                commonElements.retainAll(blockOfGenes);

                                                int k;
                                                for(k = 0; k < blockOfGenes.size(); k++) {
                                                        rowData.add(blockOfGenes.get(k));
                                                }
                                                while(k <= maxNumOfCols) {
                                                        rowData.add(" ");
                                                        k++;
                                                }
                                        }
                                }
                                gcAlignmentTableModel.addData(rowData);
                        }
                        alignmentTable.setModel(gcAlignmentTableModel);
                        return null;
                }

                protected void done() {
                    progressBar.setVisible(false);

                    alignmentTable.repaint();
                            alignmentResultsPanel.repaint();
                            alignment.repaint();
                            cards.repaint();
                    alignmentTable.setDefaultRenderer(Object.class, new TableCellRenderer() {
                            private DefaultTableCellRenderer DEFAULT_RENDERER = new
                                 DefaultTableCellRenderer();

                            @Override
                        public Component getTableCellRendererComponent(JTable table, Object
                             value, boolean isSelected, boolean hasFocus, int row, int column)
                             {
                            Component c = DEFAULT_RENDERER.getTableCellRendererComponent(table
                                 , value, isSelected, hasFocus, row, column);
                            c.setForeground(Color.BLACK);
                            if(commonElements != null){//Here `Status` is column name
                                    if(commonElements.contains(table.getValueAt(row, column)))
                                            {
```

```
                                        c.setForeground(Color.RED);
                                }
                        }
                        return c;
                    }
                });
                        alignmentTable.setTableHeader(null);
                        alignmentTable.repaint();
                        alignmentResultsPanel.repaint();
                        alignment.repaint();
                        cards.repaint();
            }
        }
        new AlignmentTask().execute();
}

/**
 * Sets the header properties for the table
 *
 * @author Bianca Silmaro
 */
private static class HeaderRenderer implements TableCellRenderer {

    DefaultTableCellRenderer renderer;

    public HeaderRenderer(JTable table) {
        renderer = (DefaultTableCellRenderer)
            table.getTableHeader().getDefaultRenderer();
        renderer.setHorizontalAlignment(JLabel.CENTER);
    }

    /**
     * Sets the properties of the cells in the header
     */
    @Override
    public Component getTableCellRendererComponent(
        JTable table, Object value, boolean isSelected,
        boolean hasFocus, int row, int col) {
        JComponent c = (JComponent)renderer.getTableCellRendererComponent(table, value,
             isSelected, hasFocus, row, col);
        c.setBorder(new EmptyBorder(2, 2, 2, 2));
        c.setBackground(new Color(243, 243, 243));
        return c;
    }
}

/**
 * Creates the model for the table of gene clusters
 *
 * @author Bianca Silmaro
 */
@SuppressWarnings("serial")
class GeneClusTableModel extends AbstractTableModel {
        private String[] columnNames = {"#", "Candidate Gene Clusters"};

        private List<Object[]> data = new ArrayList<Object[]>();

        public GeneClusTableModel() {

        }

        /**
         * Returns the number of columns in the table
         */
        public int getColumnCount() {
                return columnNames.length;
        }

        /**
         * Returns the number of rows in the table
         */
        public int getRowCount() {
                return data.size();
        }

        /**
         * Returns the column name of the specified column
         */
        public String getColumnName(int col) {
    return columnNames[col];
}

        /**
         * Returns the value at a specified row and column
         */
public Object getValueAt(int row, int col) {
        return data.get(row)[col];
}

        public void addData(int pos, String geneCluster) {
                data.add(new Object[]{pos, geneCluster});
        }
```

```
                /**
         * JTable uses this method to determine the default renderer/
         * editor for each cell.  If we didn't implement this method,
         * then the last column would contain text ("true"/"false"),
         * rather than a check box.
         */
                public Class getColumnClass(int c) {
            return getValueAt(0, c).getClass();
        }
        }

        /**
         * Creates the model for the table for the alignment of gene clusters
         * @author Bianca Silmaro
         *
         */
        @SuppressWarnings("serial")
        class AlignmentTableModel extends AbstractTableModel {
                private ArrayList<String> columnNames = new ArrayList<String>();
                private List<Object[]> data = new ArrayList<Object[]>();

                public AlignmentTableModel(int maxNumOfCols) {
                        columnNames.add("Genome");
                        for(int i = 1; i < maxNumOfCols; i++) {
                                columnNames.add("a");
                        }
                }

                /**
                 * Returns the number of columns in the table
                 */
                public int getColumnCount() {
                        return columnNames.size();
                }

                /**
                 * Returns the number of rows in the table
                 */
                public int getRowCount() {
                        return data.size();
                }

                /**
                 * Returns the column name for the specified column
                 */
                public String getColumnName(int col) {
            return columnNames.get(col);
        }

                /**
                 * Gets the value at a certain position
                 */
        public Object getValueAt(int row, int col) {
                return data.get(row)[col];
        }

        /**
         * Adds data to the table
         * @param geneCluster The gene cluster to be added
         */
                public void addData(ArrayList<String> geneCluster) {
                        Object[] newData = new Object[geneCluster.size() + 1];
                        for(int i = 0; i < geneCluster.size(); i++) {
                                newData[i] = geneCluster.get(i);
                        }
                        data.add(newData);
                }
        }
}
```

Listing 3: GeneClusteringAlgorithm

```
package geneclustertool;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map.Entry;
import java.util.TreeSet;
import jcuda.driver.*;
import static jcuda.driver.JCudaDriver.*;

public class GeneClusteringAlgorithm {
```

```
ArrayList<String> genomes = new ArrayList<String>();
ArrayList<ArrayList<String>> genes = new ArrayList<ArrayList<String>>();
HashMap<Integer, String> geneDictionaryInverse = new HashMap<Integer, String>();
ArrayList<ArrayList<Integer>> genesIntegerForm = new ArrayList<ArrayList<Integer>>();
private List<ArrayList<Vertex>> vertices = Collections.synchronizedList(new ArrayList<
    ArrayList<Vertex>>());
private int minVertexSize, maxVertexSize;

/**
 * Lays out the steps on processing the dataset
 */
public void processGeneClusters() {
        HashMap<String, Integer> geneDictionary = new HashMap<String, Integer>();
        createGeneToIntegerDictionary(geneDictionary);
        generateGraph();
}

/**
 * Reads and parses the CSV file
 * @param file                  The file uploaded by the user
 * @param minVertexSize The minimum size of the vertices to be generated
 * @param maxVertexSize The maximum size of the vertices to be generated
 */
public void parseDataset(File file, int minVertexSize, int maxVertexSize) {
        BufferedReader br = null;
        String line = "";
        String delimeter = ",";
        this.minVertexSize = minVertexSize;
        this.maxVertexSize = maxVertexSize;

        try {
                br = new BufferedReader(new FileReader(file));
                while ((line = br.readLine()) != null) {
                        String[] aGenome = line.split(delimeter);
                        ArrayList<String> genesOfGenome = new ArrayList<String>();

                        for (int i = 1; i < aGenome.length; i++) {
                                genesOfGenome.add(aGenome[i]);
                        }

                        genomes.add(aGenome[0]);
                        genes.add(genesOfGenome);
                }
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        } finally {
                if (br != null) {
                        try {
                                br.close();
                                processGeneClusters();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }
}

/**
 * Creates a directory of the gene to integer correspondence
 */
public void createGeneToIntegerDictionary(HashMap<String, Integer> geneDictionary) {
        int counter = 1;
        for(int i = 0; i < genes.size(); i++) {
                for(int j = 0; j < genes.get(i).size(); j++) {
                        String aGene = genes.get(i).get(j);
                        if(aGene.equals("0")) {
                                geneDictionary.put("0", 0);
                        }
                        else {
                                if(!geneDictionary.containsKey(aGene)) {
                                        geneDictionary.put(aGene, 0);
                                        geneDictionaryInverse.put(0, aGene);
                                }
                                // second and up instances
                                else {
                                        if(geneDictionary.get(aGene) == 0) {
                                                geneDictionary.put(aGene, counter);
                                                geneDictionaryInverse.put(counter, aGene);
                                                counter++;
                                        }
                                }
                        }
                }
        }
        convertGenesToInteger(geneDictionary);
}

/**
 * Converts genes to its integer representation
```

```
 * @param geneDictionary The mapping of genes to its integer representation
 */
public void convertGenesToInteger(HashMap<String, Integer> geneDictionary) {
        for(int i = 0; i < genes.size(); i++) {
                ArrayList<Integer> tempGenesInGenome = new ArrayList<Integer>();
                for(int j = 0; j < genes.get(i).size(); j++) {
                        String aGene = genes.get(i).get(j);
                        tempGenesInGenome.add(geneDictionary.get(aGene));
                }
                genesIntegerForm.add(tempGenesInGenome);
        }
}

/**
 * Displays the number of GPU devices available in the machine
 * @return      The number of GPU devices available
 */
public int getNumOfGPUAvailable() {
        JCudaDriver.setExceptionsEnabled(true);
        cuInit(0);

        // Obtain the number of devices
        int deviceCountArray[] = { 0 };
        cuDeviceGetCount(deviceCountArray);
        int deviceCount = deviceCountArray[0];
        return deviceCount;
}

/**
 * Creates a graph by generating vertices using threads
 */
public void generateGraph() {
        VertexGenerator[] arrOfVG = new VertexGenerator[genesIntegerForm.size()];
        for(int partition = 0; partition < genesIntegerForm.size(); partition++) {
                VertexGenerator vertexGenerator = new VertexGenerator(genesIntegerForm,
                     partition, this, minVertexSize, maxVertexSize);
                arrOfVG[partition] = vertexGenerator;
                vertexGenerator.start();
        }
        for(VertexGenerator vg : arrOfVG) {
                try {
                        vg.join();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}

/**
 * Returns the list of genes
 */
public ArrayList<ArrayList<String>> getAllGenes() {
        return genes;
}

/**
 * Returns the list of genomes
 */
public ArrayList<String> getAllGenomes() {
        return genomes;
}

/**
 * Returns the number of genomes uploaded by the user
 */
public int getNumberOfGenomes() {
        return genomes.size();
}

/**
 * Returns the list of genes in integer form
 */
public ArrayList<ArrayList<Integer>> getAllGenesIntegerForm() {
        return genesIntegerForm;
}

/**
 * Returns the vertices from the generated graph
 * @return
 */
public List<ArrayList<Vertex>> getVertices() {
        return vertices;
}

/**
 * Returns the block of genes specified by the interval
 * @param genome         The partition number or the genome
 * @param startIndex The start index of the interval of the vertex
 * @param endIndex       The end index of the interval of the vertex
 */
public ArrayList<String> getBlockOfGenes(int genome, int startIndex, int endIndex) {
        ArrayList<String> blockOfGenes = new ArrayList<String>();
```

```
        for(int i = startIndex; i < endIndex; i++) {
                blockOfGenes.add(genes.get(genome).get(i));
        }
        return blockOfGenes;
}

/**
 * Returns the mapping of integer representation of genes to its genes
 */
public HashMap<Integer, String> getGeneDictionaryInverse() {
        return geneDictionaryInverse;
}

/**
 * Adds a new partition to the graph
 * @param verticesInPartition
 */
public void addVerticesToGraph(ArrayList<Vertex> verticesInPartition) {
        vertices.add(verticesInPartition);
}

/**
 * Renames the genes in integer form to its gene names
 * @param candidateGeneClusters The candidate gene clusters produced by the algorithm
 */
public void renameGenes(List<ArrayList<Vertex>> candidateGeneClusters) {
        for(int i = 0; i < candidateGeneClusters.size(); i++) {
                for(Vertex v : candidateGeneClusters.get(i)) {
                        TreeSet<String> geneCluster = new TreeSet<String>();
                        for(Integer gene : v.getGeneContent()) {
                                if(gene != 0) {
                                        geneCluster.add(geneDictionaryInverse.get(gene));
                                }
                        }
                        v.setGeneClusterName(geneCluster);
                }
        }
}

/**
 * Returns the base gene clusters from the list of candidate gene clusters
 * @param candidateGeneClusters
 * @return
 */
public ArrayList<TreeSet<String>> getBaseGeneClusters(ArrayList<ArrayList<Vertex>>
     candidateGeneClusters) {
        ArrayList<TreeSet<String>> baseGeneClusters = new ArrayList<TreeSet<String>>();
        for(int i = 0; i < candidateGeneClusters.size(); i++) {
                baseGeneClusters.add(candidateGeneClusters.get(i).get(0).
                     getGeneClusterName());
        }
        return baseGeneClusters;
}

/**
 * Returns the key based on the value of a HashMap
 * @param geneCluster
 * @return
 */
public Vertex getKeyByValue(HashMap<Vertex, Double> geneCluster) {
        for (Entry<Vertex, Double> entry : geneCluster.entrySet()) {
        if (entry.getValue() == 0) {
            return entry.getKey();
        }
    }
        return null;
}

/**
 * Returns the maximum number of columns based on the largest size of gene cluster from
     the results
 * @param candidateGeneClusters  The candidate gene clusters
 * @param indexOfBaseGeneCluster The position of the base gene cluster
 * @return
 */
public int getMaxNumOfColumns(List<ArrayList<Vertex>> candidateGeneClusters, int
     indexOfBaseGeneCluster) {
        int maxNumOfColumns = 0;
        for(int i = 0; i < candidateGeneClusters.get(indexOfBaseGeneCluster).size(); i++)
             {
                Vertex v = candidateGeneClusters.get(indexOfBaseGeneCluster).get(i);
                if(v.getEndIndex() - v.getStartIndex() + 1 > maxNumOfColumns) {
                        maxNumOfColumns = v.getEndIndex() - v.getStartIndex() + 1;
                }
        }
        return maxNumOfColumns;
}

public void printData() {
        System.out.println("List of genomes: ");
        for (int i = 0; i < genomes.size(); i++) {
                System.out.print(genomes.get(i) + " ");
```

```
                        for(int j = 0; j < genes.get(i).size(); j++) {
                                System.out.print(genes.get(i).get(j) + " ");
                        }
                        System.out.println();
                }

                System.out.println("List of genes in integer form: ");
                for (int i = 0; i < genesIntegerForm.size(); i++) {
                        for(int j = 0; j < genesIntegerForm.get(i).size(); j++) {
                                System.out.print(genesIntegerForm.get(i).get(j) + " ");
                        }
                        System.out.println();
                }

                System.out.println("List of vertices: ");
                for (int i = 0; i < vertices.size(); i++) {
                        for(int j = 0; j < vertices.get(i).size(); j++) {
                                System.out.print(vertices.get(i).get(j).getGeneContent() + " ");
                        }
                        System.out.println();
                }
        }
}

/**
 * Generates vertices for the graph using threads
 *
 * @author Bianca Silmaro
 */
class VertexGenerator extends Thread {
        private final GeneClusteringAlgorithm gc;
        private int partition = 0;
        private ArrayList<ArrayList<Integer>> genesIntegerForm = new ArrayList<ArrayList<Integer
            >>();
        private int k1, k2;

        public VertexGenerator(ArrayList<ArrayList<Integer>> genesIntegerForm, int partition,
            GeneClusteringAlgorithm gc, int k1, int k2) {
                this.partition = partition;
                this.genesIntegerForm = genesIntegerForm;
                this.gc = gc;
                this.k1 = k1;
                this.k2 = k2;
        }

        public boolean isSetAllZeroes(TreeSet<Integer> set) {
                for(int gene : set) {
                        if(gene != 0) {
                                return false;
                        }
                }
                return true;
        }

        public void run() {
                ArrayList<Vertex> verticesInPartition = new ArrayList<Vertex>();

                for(int k = k1; k <= k2; k++) {
                        for(int i = 0; i <= genesIntegerForm.get(partition).size() - k; i++) {
                                TreeSet<Integer> geneContent = new TreeSet<Integer>();
                                for(int l = 0; l < k; l++) {
                                        if(genesIntegerForm.get(partition).get(i + l) != null) {
                                                geneContent.add(genesIntegerForm.get(partition).
                                                    get(i + l));
                                        }
                                }
                                if(!isSetAllZeroes(geneContent) && !geneContent.isEmpty()) {
                                        verticesInPartition.add(new Vertex(new TreeSet<Integer>(
                                            geneContent), partition, i, i+k));
                                }
                        }
                }

                gc.addVerticesToGraph(verticesInPartition);
                return;
        }
}
```

Listing 4: ApproximateGeneClustering

```
package geneclustertool;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;

/**
 * Handles the methods for the approximate gene clustering algorithm
```

```
 *
 * @author Bianca Silmaro
 */
public class ApproximateGeneClustering {

        private int sizeOfGeneCluster = 0;
        private int missingWeight = 0;
        private int additionalWeight = 0;
        private List<ArrayList<Vertex>> vertices;
        private List<LinkedHashMap<Vertex, Integer>> minStars = Collections.synchronizedList(new
             ArrayList<LinkedHashMap<Vertex, Integer>>());
        private HashMap<ArrayList<Vertex>, Integer> stars = new HashMap<ArrayList<Vertex>, Integer
             >();
        private boolean isDone = false;

        public ApproximateGeneClustering(GeneClusteringAlgorithm gc, int sizeOfGeneCluster, int
             missingWeight, int additionalWeight) {
                this.sizeOfGeneCluster = sizeOfGeneCluster;
                this.missingWeight = missingWeight;
                this.additionalWeight = additionalWeight;

                vertices = gc.getVertices();
                runAlgorithm();
        }

        /**
     * Forms a star from the vertices of each partition with the minimum weight
     * Initializes and creates threads to hasten the generation of minimum-weighted stars
     */
        public void runAlgorithm() {
                long startTime = System.nanoTime();
                // Create new thread
                WeightCalculator[] arrOfThreads = new WeightCalculator[vertices.size()];
                for(int partition = 0; partition < vertices.size(); partition++) {
                        WeightCalculator wc = new WeightCalculator(vertices, partition,
                             missingWeight, additionalWeight, sizeOfGeneCluster, this);
                        arrOfThreads[partition] = wc;
                        wc.start();
                }

                for(WeightCalculator wc : arrOfThreads) {
                        try {
                                wc.join();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
                isDone = true;

                long endTime = System.nanoTime();
                long totalTime = endTime - startTime;
                System.out.println("Total time: " + totalTime);
        }

        /**
     * Computes the total weight from the edges of the star
     */
        public void computeTotalWeightsOfMinStars() {
                for(int i = 0; i < minStars.size(); i++){
                        LinkedHashMap<Vertex, Integer> star = minStars.get(i);
                        int totalWeight = 0;
                        for(int weight : star.values()) {
                                totalWeight += weight;
                        }
                        ArrayList<Vertex> starVertices = new ArrayList<Vertex>();
                        for(Vertex v : star.keySet()) {
                                starVertices.add(v);
                        }
                        stars.put(starVertices, totalWeight);
                }
        }

        /**
     * Gets the HashMap of the set of vertices of a clique and its total weight
     *
     * @return A HashMap of the generated stars (or cliques) with its total weight
     */
        public HashMap<ArrayList<Vertex>, Integer> getGeneratedStarsAndWeight() {
                return stars;
        }

        /**
     * Checks if the algorithm is done running and returns the status of the algorithm
     *
     * @return true if the algorithm is done running
     */
        public boolean getIsDone() {
                return isDone;
        }

        /**
     * Adds the newly generated star (and its corresponding weight) to the list of minimum stars
```

```
     *
     * @param star A LinkedHashMap containing the set of vertices that forms a star or clique and
          its total weight
     */
        public void addStars(LinkedHashMap<Vertex, Integer> star) {
                minStars.add(star);
        }

        /**
         * A class that handles the generation of vertices using threads
         */
        class WeightCalculator extends Thread {
                private ApproximateGeneClustering apx;
                private List<ArrayList<Vertex>> vertices;
                private int partition = 0;
                private int missing = 0;
                private int additional = 0;
                private int sizeOfGeneCluster = 0;

                public WeightCalculator(List<ArrayList<Vertex>> vertices, int partition, int
                     missing, int additional, int sizeOfGeneCluster, ApproximateGeneClustering apx)
                     {
                        this.vertices = vertices;
                        this.apx = apx;
                        this.partition = partition;
                        this.missing = missing;
                        this.additional = additional;
                        this.sizeOfGeneCluster = sizeOfGeneCluster;
                }

                public void run() {
                        // Iterate through vertices in partition
                        for(int i = 0; i < vertices.get(partition).size(); i++) {
                                // Check if the length of the base vertex is the same as the size
                                     of gene cluster specified
                                if(vertices.get(partition).get(i).getGeneContent().size() ==
                                     sizeOfGeneCluster && !vertices.get(partition).get(i).
                                     getGeneContent().contains(0)) {
                                    // Iterate through other partitions
                                    LinkedHashMap<Vertex, Integer> star = new LinkedHashMap<
                                         Vertex, Integer>();
                                    star.put(vertices.get(partition).get(i), 0);
                                    for(int j = 0; j < vertices.size(); j++) {
                                            // Check if you're comparing the same partition
                                            if(partition != j) {
                                                    double minWeight = Double.
                                                         POSITIVE_INFINITY;
                                                    Vertex minVertex = null;
                                                    // Iterate through vertices in other
                                                         partition
                                                    for(int k = 0; k < vertices.get(j).size();
                                                          k++) {
                                                       Vertex v1 = vertices.get(partition
                                                            ).get(i);
                                                       Vertex v2 = vertices.get(j).get(k)
                                                            ;
                                                       int tempWeight = v1.computeWeight(
                                                            v2, missing, additional);
                                                       if(tempWeight < minWeight) {
                                                               minWeight = tempWeight;
                                                               minVertex = v2;
                                                       }
                                                    }
                                                    star.put(minVertex, (int)minWeight);
                                            }
                                    }
                                    apx.addStars(star);
                                }
                        }
                }

                /**
                 * Prints the vertices and its weight from the star formed
                 *
                 * @param star A HashMap of the vertices and its corresponding weight
                 */
                public void printStar(HashMap<Vertex, Integer> star) {
                        System.out.println("---");
                        for (HashMap.Entry<Vertex, Integer> entry : star.entrySet()) {
                            Vertex key = entry.getKey();
                            int value = entry.getValue();

                            System.out.println(key.getGeneContent() + " - " + value);
                        }
                        System.out.println();


                }
        }

}
```

Listing 5: ExactGeneClustering

```
package geneclustertool;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import static jcuda.driver.JCudaDriver.*;
import java.io.*;
import jcuda.*;
import jcuda.driver.*;
import jcuda.runtime.JCuda;
import jcuda.runtime.cudaDeviceProp;

/**
 * Handles the methods for the exact gene clustering algorithm for both GPU and non-GPU algorithms
 *
 * @author Bianca Silmaro
 */
public class ExactGeneClustering {

        private List<ArrayList<Vertex>> vertices;
        private List<ArrayList<Vertex>> generatedStars = Collections.synchronizedList(new
            ArrayList<ArrayList<Vertex>>());
        private LinkedHashMap<ArrayList<Vertex>, Integer> stars = new LinkedHashMap<ArrayList<
            Vertex>, Integer>();
        private int numOfCandidateGeneClusters = 0;
        private boolean isDone = false;

        public ExactGeneClustering(List<ArrayList<Vertex>> vertices, int
            numOfCandidateGeneClusters) {
                this.numOfCandidateGeneClusters = numOfCandidateGeneClusters;
                this.vertices = vertices;
        }

        /**
     * Computes for the total number of vertices in the graph
     *
     * @return The number of vertices
     */
        public int getTotalNumOfVertices() {
                int size = 0;
                for(int i = 0; i < vertices.size(); i++) {
                        size += vertices.get(i).size();
                }

                return size;
        }

        /**
     * Converts the matrix of vertices into a 1D array of vertices
     *
     * @return The 1D array of the vertices
     */
        public Vertex[] convertToArrOfVertices() {
                Vertex[] arrOfVertices = new Vertex[getTotalNumOfVertices()];
                int ctr = 0;
                for(int i = 0; i < vertices.size(); i++) {
                        for(int j = 0; j < vertices.get(i).size(); j++) {
                                arrOfVertices[ctr] = vertices.get(i).get(j);
                                ctr++;
                        }
                }

                return arrOfVertices;
        }

        /**
     * Creates an array of the mapping of index to vertex
     *
     * @return The 1D array of indices corresponding to the vertices
     */
        public int[] getIndicesOf1DArray(int totalNumOfVertices) {
                int[] indices = new int[totalNumOfVertices];
                for(int i = 0; i < totalNumOfVertices; i++) {
                        indices[i] = i;
                }

                return indices;
        }

        /**
     * Creates an array for the cumulative number of vertices per partition
     *
     * @return An integer array of the cumulative number of vertices per partition
     */
        public int[] getCumSizesOfPartition() {
```

```
            int[] cumSizesOfPartition = new int[vertices.size()];
            cumSizesOfPartition[0] = vertices.get(0).size();
            for(int i = 1; i < vertices.size(); i++) {
                    cumSizesOfPartition[i] = vertices.get(i).size() + cumSizesOfPartition[i
                        -1];
            }

            return cumSizesOfPartition;
    }

    /**
* Creates an array of the number of vertices per partition
*
* @return An integer array of the number of vertices per partition
*/
    public int[] getSizesOfPartition() {
            int[] sizesOfPartition = new int[vertices.size()];
            for(int i = 0; i < vertices.size(); i++) {
                    sizesOfPartition[i] = vertices.get(i).size();
            }
            return sizesOfPartition;
    }

    /**
* Computes for the total number of combinations (output) that will be generated
*
* @param sizesOfPartition An integer array of the number of vertices per partition
* @return The total number of combinations that will be generated
*/
    public int computeOutputArraySize(int[] sizesOfPartition) {
            int outputSize = 1;
        for(int i = 0; i < vertices.size(); i++) {
            outputSize *= sizesOfPartition[i];
        }
            return outputSize * vertices.size();
    }

    /**
* Creates an array of the cumulative number of combinations (output) that will be generated
     per vertex
*
* @return An integer array of the cumulative number of combinations that will be generated
     per base gene cluster
*/
    public long[] getOutputSizesOfBaseGeneCluster() {
            long[] outputSizesOfBaseGeneCluster = new long[vertices.get(0).size()];

            int tempOutputSize = 1;
            for(int j = 1; j < vertices.size(); j++) {
                    tempOutputSize *= vertices.get(j).size();
            }
            outputSizesOfBaseGeneCluster[0] = tempOutputSize * vertices.size();
            for(int i = 1; i < vertices.get(0).size(); i++) {
                    outputSizesOfBaseGeneCluster[i] = tempOutputSize * vertices.size() +
                        outputSizesOfBaseGeneCluster[i - 1];
            }

            return outputSizesOfBaseGeneCluster;
    }

    /**
* Gets the total device memory available in the GPU
*
* @return The total device memory available in the GPU
*/
    public long getDeviceMemory() {
// Obtain the number of devices

 cudaDeviceProp deviceProp = new cudaDeviceProp();
 JCuda.cudaGetDeviceProperties(deviceProp, 0);
 long deviceMem = deviceProp.totalGlobalMem;
 return deviceMem;

    }

    /**
* Computes for the total number of combinations that will be generated given the constraints
     of the device memory of the GPU
*
* @param outputSizesOfBaseGeneCluster An array of the cumulative number of combinations (
     output) per base gene cluster
* @return The total number of combinations (output) that will be generated given the
     constraints of the device memory of the GPU
*/
    public int computeOutputSizeAllocation(long[] outputSizesOfBaseGeneCluster) {
            long deviceMem = getDeviceMemory();
            long memAvailable = (long) (deviceMem - deviceMem * 0.8);
            long availableOutputSize = memAvailable * 8 / 32;

            for(int i = 0; i < outputSizesOfBaseGeneCluster.length; i++) {
                    if(outputSizesOfBaseGeneCluster[i] > availableOutputSize && i != 0) {
                            return i - 1;
```

```
                }
            }
            return outputSizesOfBaseGeneCluster.length;
    }

    /**
 * Allocates the necessary input data and runs the GPU algorithm
 */
    public void allocateHostInputData() throws IOException {
            // Compute time of algorithm
            long startTime = System.nanoTime();

            // Start creating modules, function, and context
            // Enable exceptions and omit all subsequent error checks
            JCudaDriver.setExceptionsEnabled(true);

            // Create the PTX file by calling the NVCC
            byte ptxData[] = toByteArray(getClass().getResourceAsStream("/combination.ptx"));
            //String ptxFileName = preparePtxFile("combination.cu");

            // Initialize the driver and create a context for the first device
            cuInit(0);
            CUdevice device = new CUdevice();
            cuDeviceGet(device, 0);
            CUcontext context = new CUcontext();
            cuCtxCreate(context, 0, device);

            // Load the ptx file
            CUmodule module = new CUmodule();
            cuModuleLoadData(module, ptxData);
            //cuModuleLoad(module, ptxFileName);

            // Obtain a function pointer to the "combination" algorithm function
            CUfunction function = new CUfunction();
            cuModuleGetFunction(function, module, "combination");

            // Allocate the host input data
            Vertex[] arrOfVertices = convertToArrOfVertices();
            int[] indices = getIndicesOf1DArray(getTotalNumOfVertices());
            int[] cumSizesOfPartition = getCumSizesOfPartition();
            int[] sizesOfPartition = getSizesOfPartition();
            long[] outputSizesOfBaseGeneCluster = getOutputSizesOfBaseGeneCluster();

            int numOfIndices = indices.length;
            int numOfIndicesOfBaseGeneClusters = vertices.get(0).size();
            int numOfPartitions = vertices.size();
            int outputSize = 0;
            int starSize = numOfPartitions * numOfIndicesOfBaseGeneClusters;

            // Stop index is the index in the array base gene clusters where its output is the
                 maximum integer value
            int startIndex = 0;
            int stopIndex = computeOutputSizeAllocation(outputSizesOfBaseGeneCluster);

            if(stopIndex < outputSizesOfBaseGeneCluster.length) {
                    while(startIndex < vertices.get(0).size()) {
                            // Perform data slicing
                            if(stopIndex > vertices.get(0).size() - startIndex) {
                                    numOfIndicesOfBaseGeneClusters = vertices.get(0).size() -
                                         startIndex;
                            } else {
                                    numOfIndicesOfBaseGeneClusters = stopIndex + 1;
                            }

                            int[] tempOutputSizesOfBaseGeneCluster = new int[
                                 numOfIndicesOfBaseGeneClusters];
                            for(int i = 0; i < numOfIndicesOfBaseGeneClusters; i++) {
                                    tempOutputSizesOfBaseGeneCluster[i] = (int)
                                         outputSizesOfBaseGeneCluster[i];
                            }
                            starSize = numOfPartitions * numOfIndicesOfBaseGeneClusters;
                            outputSize = numOfIndicesOfBaseGeneClusters;
                            for(int i = 1; i < numOfPartitions; i++) {
                                    outputSize *= vertices.get(i).size();
                            }
                            outputSize *= numOfPartitions;

                            // Run algorithm
                            try {
                                    runGPUAlgorithm(startIndex, arrOfVertices, indices,
                                         cumSizesOfPartition, sizesOfPartition,
                                         tempOutputSizesOfBaseGeneCluster,
                                                 numOfIndices,
                                                      numOfIndicesOfBaseGeneClusters,
                                                      numOfPartitions, outputSize, starSize,
                                                       startTime, function);
                            } catch (IOException e) {
                                    e.printStackTrace();
                            }

                            // Set start indices and stop indices
                            startIndex += stopIndex + 1;
```

```
                }
        } else {
                // Do not perform data slicing and continue with the algorithm
                outputSize = computeOutputArraySize(sizesOfPartition);
                int[] tempOutputSizesOfBaseGeneCluster = new int[
                     numOfIndicesOfBaseGeneClusters];
                for(int i = 0; i < numOfIndicesOfBaseGeneClusters; i++) {
                        tempOutputSizesOfBaseGeneCluster[i] = (int)
                             outputSizesOfBaseGeneCluster[i];
                }
                try {
                        runGPUAlgorithm(startIndex, arrOfVertices, indices,
                             cumSizesOfPartition, sizesOfPartition,
                             tempOutputSizesOfBaseGeneCluster,
                                     numOfIndices, numOfIndicesOfBaseGeneClusters,
                                          numOfPartitions, outputSize, starSize,
                                          startTime, function);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        isDone = true;
        long endTime = System.nanoTime();
        long totalTime = endTime - startTime;
        System.out.println("Total time: " + totalTime);
}

public void runGPUAlgorithm(int startIndex, Vertex[] arrOfVertices, int[] indices, int[]
     cumSizesOfPartition, int[] sizesOfPartition, int[] outputSizesOfBaseGeneCluster,
                int numOfIndices, int numOfIndicesOfBaseGeneClusters, int numOfPartitions,
                     int outputSize, int starSize, long startTime, CUfunction function)
                    throws IOException {
        // Allocate the device input data, and copy the host input data to the device
        CUdeviceptr dev_indices = new CUdeviceptr();
        cuMemAlloc(dev_indices, numOfIndices * Sizeof.INT);
        cuMemcpyHtoD(dev_indices, Pointer.to(indices), numOfIndices * Sizeof.INT);

        CUdeviceptr dev_cumSizesOfPartition = new CUdeviceptr();
        cuMemAlloc(dev_cumSizesOfPartition, numOfPartitions * Sizeof.INT);
        cuMemcpyHtoD(dev_cumSizesOfPartition, Pointer.to(cumSizesOfPartition),
             numOfPartitions * Sizeof.INT);

        CUdeviceptr dev_sizesOfPartition = new CUdeviceptr();
        cuMemAlloc(dev_sizesOfPartition, numOfPartitions * Sizeof.INT);
        cuMemcpyHtoD(dev_sizesOfPartition, Pointer.to(sizesOfPartition), numOfPartitions *
              Sizeof.INT);

        CUdeviceptr dev_outputSizesOfBaseGeneCluster = new CUdeviceptr();
        cuMemAlloc(dev_outputSizesOfBaseGeneCluster, numOfIndicesOfBaseGeneClusters *
             Sizeof.INT);
        cuMemcpyHtoD(dev_outputSizesOfBaseGeneCluster, Pointer.to(
             outputSizesOfBaseGeneCluster), numOfIndicesOfBaseGeneClusters * Sizeof.INT);

        CUdeviceptr dev_star = new CUdeviceptr();
        cuMemAlloc(dev_star, numOfPartitions * numOfIndicesOfBaseGeneClusters * Sizeof.INT
             );

        // Allocate device output memory
        CUdeviceptr deviceOutput = new CUdeviceptr();
        cuMemAlloc(deviceOutput, outputSize * Sizeof.INT);

        // Set up the kernel parameters: A pointer to an array of pointers which point to
             the actual values
        Pointer kernelParameters = Pointer.to(
                Pointer.to(new int[]{numOfIndicesOfBaseGeneClusters}),
                Pointer.to(new int[]{numOfPartitions}),
                Pointer.to(dev_indices),
                Pointer.to(dev_cumSizesOfPartition),
                Pointer.to(dev_sizesOfPartition),
                Pointer.to(dev_outputSizesOfBaseGeneCluster),
                Pointer.to(dev_star),
                Pointer.to(deviceOutput),
                Pointer.to(new int[]{startIndex})
        );

        // Call the kernel function
        int blockSizeX = 32;
        int gridSizeX = 128;
        cuLaunchKernel(function,
                gridSizeX, 1, 1,                        // Grid dimension
                blockSizeX, 1, 1,                       // Block dimension
                0, null,                                        // Shared memory size and
                     stream
                kernelParameters, null          // Kernel and extra parameters
        );
        cuCtxSynchronize();

        // Free memory
        cuMemFree(dev_indices);
        cuMemFree(dev_cumSizesOfPartition);
        cuMemFree(dev_sizesOfPartition);
```

```
            cuMemFree(dev_outputSizesOfBaseGeneCluster);
            cuMemFree(dev_star);

            indices = null;
            cumSizesOfPartition = null;
            sizesOfPartition = null;
            outputSizesOfBaseGeneCluster = null;

            // Allocate host output memory and copy the device output to the host
            int[] output = new int[outputSize];
            cuMemcpyDtoH(Pointer.to(output), deviceOutput, outputSize * Sizeof.INT);

            // Process output
            processOutput(output, numOfPartitions, cumSizesOfPartition, arrOfVertices);

            // Clean up
            cuMemFree(deviceOutput);
    }

    /**
 * The extension of the given file name is replaced with "ptx".
 * If the file with the resulting name does not exist, it is
 * compiled from the given file using NVCC. The name of the
 * PTX file is returned.
 *
 * @param cuFileName The name of the .CU file
 * @return The name of the PTX file
 * @throws IOException If an I/O error occurs
 */
    private String preparePtxFile(String cuFileName) throws IOException {
            int endIndex = cuFileName.lastIndexOf('.');
    if (endIndex == -1) {
        endIndex = cuFileName.length()-1;
    }
    String ptxFileName = cuFileName.substring(0, endIndex+1)+"ptx";
    File ptxFile = new File(ptxFileName);
    if (ptxFile.exists()) {
        return ptxFileName;
    }

    File cuFile = new File(cuFileName);
    if (!cuFile.exists()) {
        throw new IOException("Input file not found: "+cuFileName);
    }
    String modelString = "-m" + System.getProperty("sun.arch.data.model");
    System.out.println(cuFile.getPath());
    String command = "nvcc " + modelString + " -ptx " + cuFile.getPath() + " -o " +
         ptxFileName;

    System.out.println("Executing\n"+command);
    Process process = Runtime.getRuntime().exec(command);

    String errorMessage = new String(toByteArray(process.getErrorStream()));
    String outputMessage = new String(toByteArray(process.getInputStream()));
    int exitValue = 0;
    try {
        exitValue = process.waitFor();
    }
    catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        throw new IOException(
            "Interrupted while waiting for nvcc output", e);
    }

    if (exitValue != 0) {
        System.out.println("nvcc process exitValue "+exitValue);
        System.out.println("errorMessage:\n"+errorMessage);
        System.out.println("outputMessage:\n"+outputMessage);
        throw new IOException(
            "Could not create .ptx file: "+errorMessage);
    }

    System.out.println("Finished creating PTX file");
    return ptxFileName;
    }

    /**
 * Fully reads the given InputStream and returns it as a byte array
 *
 * @param inputStream The input stream to read
 * @return The byte array containing the data from the input streamf
 * @throws IOException If an I/O error occurs
 */
private static byte[] toByteArray(InputStream inputStream) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    byte buffer[] = new byte[8192];
    while (true) {
        int read = inputStream.read(buffer);
        if (read == -1)
        {
            break;
        }
```

```
            baos.write(buffer, 0, read);
        }
        return baos.toByteArray();
    }

    /**
     * Computes for the weight for each possible edge in the star formed
     *
     * @param star A set of vertices, also called a clique, from each combination generated
     * @return The total weight of all edges
     */
    public int computeWeightOfStar(Vertex[] star) {
                int missingWeight = 1;
                int additionalWeight = 1;
                int totalWeight = 0;

                for(int i = 0; i < star.length - 1; i++) {
                        for(int j = i + 1; j < star.length; j++) {
                                int weight = star[i].computeWeight(star[j], missingWeight,
                                        additionalWeight);
                                totalWeight += weight;
                        }
                }

                return totalWeight;
        }

    /**
     * Converts the HashMap to a 2D list to get the vertices only, and not including its weight
     *
     * @return A sorted list of the candidate gene clusters by its weight
     */
    public ArrayList<ArrayList<Vertex>> getSortedGeneClusters() {
            List<HashMap.Entry<ArrayList<Vertex>, Integer>> list = new LinkedList<HashMap.Entry<
                    ArrayList<Vertex>, Integer>>(stars.entrySet());
        ArrayList<ArrayList<Vertex>> resultWithoutTotalWeight = new ArrayList<ArrayList<Vertex>>()
                ;
            for (HashMap.Entry<ArrayList<Vertex>, Integer> entry : list) {
                resultWithoutTotalWeight.add(entry.getKey());
            }
            return resultWithoutTotalWeight;
    }

    /**
     * Post-processing of the combinations generated from the GPU algorithm
     * It converts the combinations from integers to its corresponding vertices and gets its total
            weight
     * The total weights are sorted to get the minimum-weighted clique (combination)
     *
     * @param output The output array from the GPU algorithm which contains the combinations
            generated
     * @param numOfPartitions The number of partitions (or genomes) in the graph
     * @param cumSizesOfPartition The cumulative number of vertices per partition
     * @param arrOfVertices A 1D array of all vertices
     */
    public void processOutput(int[] output, int numOfPartitions, int[] cumSizesOfPartition, Vertex
            [] arrOfVertices) {
        int index = stars.size();
        Vertex arr[] = new Vertex[numOfPartitions];
        int ctr = 0;

        for(int i = 0; i < output.length; i++) {
                arr[ctr] = arrOfVertices[output[i]];
                ctr++;
                if((i+1)%numOfPartitions == 0) {
                                ctr = 0;

                                // Get weight of star
                        int weight = computeWeightOfStar(arr);

                        // Populate data first
                        if(index < numOfCandidateGeneClusters) {
                                stars.put(new ArrayList<Vertex>(Arrays.asList(arr)), Integer.
                                        valueOf(weight));
                                index++;
                        } else {
                                // Add star to hashmap of stars
                                stars.put(new ArrayList<Vertex>(Arrays.asList(arr)), Integer.
                                        valueOf(weight));

                                // Sort stars and remove last element
                                stars = sortByValue();
                                index++;
                        }
                        }
        }
    }

    /**
     * Sorts the candidate gene clusters through its total weight
     *
     * @return  A LinkedHashMap of the sorted gene clusters with its corresponding total weight
```

```
 */
public LinkedHashMap<ArrayList<Vertex>, Integer> sortByValue() {
        List<HashMap.Entry<ArrayList<Vertex>, Integer>> list = new LinkedList<HashMap.Entry<
             ArrayList<Vertex>, Integer>>(stars.entrySet());
        Collections.sort(list, new Comparator<HashMap.Entry<ArrayList<Vertex>, Integer>>() {
            public int compare(HashMap.Entry<ArrayList<Vertex>, Integer> o1, HashMap.Entry<
                 ArrayList<Vertex>, Integer> o2) {
                return (o1.getValue()).compareTo( o2.getValue() );
            }
        });

        LinkedHashMap<ArrayList<Vertex>, Integer> result = new LinkedHashMap<ArrayList<Vertex
             >, Integer>();
        for(int i = 0; i < list.size() - 1; i++) {
            result.put(list.get(i).getKey(), list.get(i).getValue());
        }

        return result;
    }

/**
 * Post-processing of the combinations generated from the non-GPU algorithm
 *
 * @param arr An array of vertices for the corresponding clique
 * @param weight The total weight computed for all edges of arr
 */
    public void processStar(Vertex[] arr, int weight) {
            if(stars.size() < numOfCandidateGeneClusters) {
                    stars.put(new ArrayList<Vertex>(Arrays.asList(arr)), Integer.valueOf(
                         weight));
            } else {
                    // Add star to hashmap of stars
                    stars.put(new ArrayList<Vertex>(Arrays.asList(arr)), Integer.valueOf(
                         weight));
                    // Sort stars and remove last element
                    stars = sortByValue();
            }
    }

    /**
 * The recursive function for generating combinations of vertices for the non-GPU algorithm
 *
 * @param numOfPartitions The number of partitions in the graph (or number of genomes entered)
 * @param arr An array of vertices that make up the clique
 * @param currPartition The current partition number being processed
 * @param vertices The matrix of vertices in the graph
 */
    public void recursive(int numOfPartitions, Vertex[] arr, int currPartition, List<ArrayList
         <Vertex>> vertices) {
            if(currPartition == numOfPartitions - 1) {
                    // Get weight of star
                    int weight = computeWeightOfStar(arr);
                    processStar(arr, weight);
                    return;
            } else {
                    currPartition++;
                    for(int i = 0; i < vertices.get(currPartition).size(); i++) {
                            arr[currPartition] = vertices.get(currPartition).get(i);
                            recursive(numOfPartitions, arr, currPartition, vertices);
                    }
            }
    }

    /**
 * Finds combinations of vertices to form a clique without using a GPU
 */
    public void runAlgorithm() {
            long startTime = System.nanoTime();

            int numOfPartitions = vertices.size();

            for(int i = 0; i < vertices.get(0).size(); i++) {
                    Vertex[] arr = new Vertex[numOfPartitions];
                    arr[0] = vertices.get(0).get(i);
                    int currPartition = 0;

                    recursive(numOfPartitions, arr, currPartition, vertices);
            }

            isDone = true;
            long endTime = System.nanoTime();
            long totalTime = endTime - startTime;
            System.out.println("Total time: " + totalTime);
    }

    /**
 * Gets the LinkedHashMap of the set of vertices of a clique and its total weight
 *
 * @return A LinkedHashMap of the generated stars (or cliques) with its total weight
 */
    public LinkedHashMap<ArrayList<Vertex>, Integer> getGeneratedStarsAndWeight() {
            return stars;
```

```
        }

        /**
     * Adds the newly generated star to the list of generated stars
     *
     * @param star A set of vertices that forms a star or clique
     */
        public void compileStars(ArrayList<Vertex> star) {
                generatedStars.add(star);
        }

        /**
     * Checks if the algorithm is done running and returns the status of the algorithm
     *
     * @return true if the algorithm is done running
     */
        public boolean getIsDone() {
                return isDone;
        }

}
```

Listing 6: AlgoUtils

```
package geneclustertool;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;

/**
 * Class for additional helper functions
 *
 * @author Bianca Silmaro
 */
public class AlgoUtils {

        private ArrayList<ArrayList<Vertex>> sortedGeneClusters = new ArrayList<ArrayList<Vertex
            >>();
        private int numOfCandidateGeneClusters = 0;

        public AlgoUtils(HashMap<ArrayList<Vertex>, Integer> starsAndWeights, int
            numOfCandidateGeneClusters, GeneClusteringAlgorithm gc) {
                this.numOfCandidateGeneClusters = numOfCandidateGeneClusters;

                getGeneClusters(starsAndWeights);
        }

        /**
         * Sorts the stars formed by its total weight
         */
        public void getGeneClusters(HashMap<ArrayList<Vertex>, Integer> stars) {
                // Sort candidateGeneClusters based on their total weights <Double>
                sortedGeneClusters = sortByValue(stars);
        }

        /**
         * Helper function that sorts a HashMap by its value
         * @return Sorted ArrayList of stars formed (without the weight)
         */
        public ArrayList<ArrayList<Vertex>> sortByValue(HashMap<ArrayList<Vertex>, Integer> stars)
            {
            List<HashMap.Entry<ArrayList<Vertex>, Integer>> list = new LinkedList<HashMap.Entry<
                ArrayList<Vertex>, Integer>>(stars.entrySet());
            Collections.sort(list, new Comparator<HashMap.Entry<ArrayList<Vertex>, Integer>>() {
                public int compare(HashMap.Entry<ArrayList<Vertex>, Integer> o1, HashMap.Entry<
                    ArrayList<Vertex>, Integer> o2) {
                    return (o1.getValue()).compareTo( o2.getValue() );
                }
            });

            LinkedHashMap<ArrayList<Vertex>, Integer> result = new LinkedHashMap<ArrayList<Vertex
                >, Integer>();
            ArrayList<ArrayList<Vertex>> resultWithoutTotalWeight = new ArrayList<ArrayList<Vertex
                >>();
            for (HashMap.Entry<ArrayList<Vertex>, Integer> entry : list) {
                result.put(entry.getKey(), entry.getValue());
                resultWithoutTotalWeight.add(entry.getKey());
            }
            return resultWithoutTotalWeight;
        }

        /**
         * Returns the sorted gene clusters with size specified by the number of candidate gene
             clusters
         */
```

```
        public ArrayList<ArrayList<Vertex>> getSortedGeneClusters() {
                if(sortedGeneClusters.size() > numOfCandidateGeneClusters) {
                        return new ArrayList<ArrayList<Vertex>>(sortedGeneClusters.subList(0,
                            numOfCandidateGeneClusters));
                } else {
                        return sortedGeneClusters;
                }
        }

}
```

Listing 7: GraphManager

```
package geneclustertool;

import java.awt.Dimension;
import java.awt.Font;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;

import org.graphstream.graph.*;
import org.graphstream.graph.implementations.*;
import org.graphstream.ui.view.View;
import org.graphstream.ui.view.Viewer;

/**
 * Creates a graph visualization of the dataset
 *
 * @author Bianca Silmaro
 */
public class GraphManager {

        private List<ArrayList<Vertex>> vertices = new ArrayList<ArrayList<Vertex>>();
        private Graph graph;

        public GraphManager(List<ArrayList<Vertex>> vertices) {
                this.vertices = vertices;
                constructGraph();
        }

        /**
         * Returns the view created by GraphStream with its graph
         */
        public View getView() {
                Viewer viewer = new Viewer(graph, Viewer.ThreadingModel.GRAPH_IN_GUI_THREAD);
                viewer.enableAutoLayout();
                View view = viewer.addDefaultView(false);
                View defaultView = viewer.getDefaultView();
                return defaultView;
        }

        /**
         * Creates a new window that shows the table of vertices and its corresponding gene
             content
         */
        public void showTableOfVertices() {
                JFrame directory = new JFrame();
                directory.setTitle("Directory of vertices");
                try {
                        directory.setIconImage(ImageIO.read(getClass().getResource("/images/
                            clustar-icon.png")));
                } catch (IOException e2) {
                        e2.printStackTrace();
                }

                JTextArea table = new JTextArea();
                table.setEditable(false);
                table.setFont(new Font("PT Sans", Font.PLAIN, 15));
                table.setLineWrap(true);
                table.setWrapStyleWord(true);

                populateTable(table);

                JScrollPane verticesPane = new JScrollPane(table, ScrollPaneConstants.
                    VERTICAL_SCROLLBAR_AS_NEEDED, ScrollPaneConstants.
                    HORIZONTAL_SCROLLBAR_AS_NEEDED);
                verticesPane.setPreferredSize(new Dimension(300, 350));
                directory.getContentPane().add(verticesPane);
                directory.pack();
                directory.setVisible(true);
        }

        /**
```

```
         * Populates the table of vertices and its correspoding gene contents
         */
        public void populateTable(JTextArea table) {
                for(int i = 0; i < vertices.size(); i++) {
                        for(Vertex v : vertices.get(i)) {
                                String nodeName = "v" + v.getPartitionNumber() + "(" + v.
                                    getStartIndex() + "," + v.getEndIndex() + ")\t" + v.
                                    getGeneClusterName() + "\n";
                                table.append(nodeName);
                        }
                }
        }

        /**
         * Constructs the graph by adding vertices and edges
         * @return      The graph generated by GraphStream
         */
        public Graph constructGraph() {
                graph = new SingleGraph("My first graph");

                for(int i = 0; i < vertices.size(); i++) {
                        for(Vertex v : vertices.get(i)) {
                                String nodeName = "v" + v.getPartitionNumber() + "(" + v.
                                    getStartIndex() + "," + v.getEndIndex() + ")";
                                Node n = graph.addNode(nodeName);
                                n.setAttribute("ui.label", nodeName);
                        }
                }

                for(int i = 0; i < vertices.size(); i++) {
                        for(int j = i + 1; j < vertices.size(); j++) {
                                for(Vertex v1 : vertices.get(i)) {
                                        String v1_name = "v" + v1.getPartitionNumber() + "(" + v1.
                                            getStartIndex() + "," + v1.getEndIndex() + ")";
                                        for(Vertex v2 : vertices.get(j)) {
                                                String v2_name = "v" + v2.getPartitionNumber() +
                                                    "(" + v2.getStartIndex() + "," + v2.
                                                    getEndIndex() + ")";
                                                graph.addEdge(v1_name + "-" + v2_name, v1_name,
                                                    v2_name);
                                        }
                                }
                        }
                }

                return graph;
        }

}
```

Listing 8: AdjacencyMatrixManager

```
package geneclustertool;

import java.awt.Dimension;
import java.awt.Font;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.TreeSet;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;

/**
 * Creates and populates the corresponding adjacency matrix of the graph
 *
 * @author Bianca Silmaro
 */
public class AdjacencyMatrixManager {

        private int[][] adjacencyMatrix;
        private List<ArrayList<Vertex>> vertices = new ArrayList<ArrayList<Vertex>>();
        private int missingWeight = 1;
        private int additionalWeight = 1;
        private List<Vertex> listOfVertices = new ArrayList<Vertex>();

        public AdjacencyMatrixManager(List<ArrayList<Vertex>> vertices, int missingWeight, int
            additionalWeight) {
                System.out.println("Constructing adjacency matrix...");
                this.vertices = vertices;
                this.missingWeight = missingWeight;
                this.additionalWeight = additionalWeight;

                constructListOfVertices();
                adjacencyMatrix = new int[listOfVertices.size()][listOfVertices.size()];
                computeAdjacencyMatrix(listOfVertices);
```

```
}

/**
 * Constructs a 1D array of a list of vertices
 */
public void constructListOfVertices() {
        for(int i = 0; i < vertices.size(); i++) {
                listOfVertices.addAll(vertices.get(i));
        }
}

/**
 * Computes for the weights in the adjacency matrix using threads
 * @param listOfVertices
 */
public void computeAdjacencyMatrix(List<Vertex> listOfVertices) {
        WeightMatrixThread[] arrOfMatrixThreads = new WeightMatrixThread[listOfVertices.
             size()];

        for(int posOfVertex = 0; posOfVertex < listOfVertices.size(); posOfVertex++) {
                // Create thread to compute weight
                WeightMatrixThread weightMatrixThread = new WeightMatrixThread(
                     listOfVertices, posOfVertex, missingWeight, additionalWeight, this);
                arrOfMatrixThreads[posOfVertex] = weightMatrixThread;
                weightMatrixThread.start();
        }

        for(WeightMatrixThread wmt : arrOfMatrixThreads) {
                try {
                        wmt.join();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}

/**
 * Set values to the adjacency matrix
 * @param pos1          The row index of the matrix
 * @param pos2          The column index of the matrix
 * @param weight        The value at position col, row
 */
public void constructAdjacencyMatrix(int pos1, int pos2, int weight) {
        if(pos1 == pos2) {
                adjacencyMatrix[pos1][pos2] = 0;

        } else {
                adjacencyMatrix[pos1][pos2] = weight;
                adjacencyMatrix[pos2][pos1] = weight;
        }
}

/**
 * Displays a new window which shows a table of the vertices and its corresponding gene
     content
 */
public void showTableOfVertices() {
        JFrame directory = new JFrame();
        directory.setTitle("Directory of vertices");
        try {
                directory.setIconImage(ImageIO.read(getClass().getResource("/images/
                     clustar-icon.png")));
        } catch (IOException e2) {
                e2.printStackTrace();
        }

        JTextArea table = new JTextArea();
        table.setEditable(false);
        table.setFont(new Font("PT Sans", Font.PLAIN, 15));
        table.setLineWrap(true);
        table.setWrapStyleWord(true);

        populateTable(table);

        JScrollPane verticesPane = new JScrollPane(table, ScrollPaneConstants.
             VERTICAL_SCROLLBAR_AS_NEEDED, ScrollPaneConstants.
             HORIZONTAL_SCROLLBAR_AS_NEEDED);
        verticesPane.setPreferredSize(new Dimension(300, 350));
        directory.getContentPane().add(verticesPane);
        directory.pack();
        directory.setVisible(true);
}

/**
 * Populates the table of vertices and its correspoding gene contents
 */
public void populateTable(JTextArea table) {
        for(int i = 0; i < listOfVertices.size(); i++) {
                table.append("v" + i + "\t" + listOfVertices.get(i).getGeneClusterName() +
                     "\n");
        }
}
```

```
        /**
         * Computes for the total number of vertices in the matrix
         */
        public long getTotalNumOfVertices() {
                long total = 0;
                for(int i = 0; i < vertices.size(); i++) {
                        total += vertices.get(i).size();
                }
                return total;
        }

        /**
         * Returns the list of vertices in 1D array
         */
        public List<Vertex> getListOfVertices() {
                return listOfVertices;
        }

        /**
         * Returns the adjacency matrix created
         */
        public int[][] getAdjacencyMatrix() {
                return adjacencyMatrix;
        }

        /**
         * Renames gene contents of the vertices into its corresponding gene name
         * @param geneDictionary
         */
        public void renameVerticesToGenes(HashMap<Integer, String> geneDictionary) {
                for(Vertex v : listOfVertices) {
                        if(v.getGeneClusterName().isEmpty()) {
                                TreeSet<String> geneCluster = new TreeSet<String>();
                                for(Integer gene : v.getGeneContent()) {
                                        geneCluster.add(geneDictionary.get(gene));
                                }
                                v.setGeneClusterName(geneCluster);
                        }
                }
        }

        /**
         * Prints the adjacency matrix
         */
        public void printAdjacencyMatrix() {
                for(int i = 0; i < adjacencyMatrix.length; i++) {
                        for(int j = 0; j < adjacencyMatrix.length; j++) {
                                System.out.print(adjacencyMatrix[i][j] + "\t");
                        }
                        System.out.println();
                }
        }
}
/**
 * A class that handles the computation of weights for the adjacency matrix using threads
 */
class WeightMatrixThread extends Thread {
        private AdjacencyMatrixManager matrixManager;
        private List<Vertex> listOfVertices;
        private int posOfBaseVertex;
        private int missing = 0;
        private int additional = 0;

        public WeightMatrixThread(List<Vertex> listOfVertices, int posOfBaseVertex, int missing,
             int additional, AdjacencyMatrixManager matrixManager) {
                this.listOfVertices = listOfVertices;
                this.matrixManager = matrixManager;
                this.posOfBaseVertex = posOfBaseVertex;
                this.missing = missing;
                this.additional = additional;
        }

        public void run() {
                for(int j = 0; j <= posOfBaseVertex; j++) {
                        Vertex baseVertex = listOfVertices.get(posOfBaseVertex);
                        int weight = baseVertex.computeWeight(listOfVertices.get(j), missing,
                             additional);
                        matrixManager.constructAdjacencyMatrix(posOfBaseVertex, j, weight);
                }
        }
}
```

Listing 9: PDFManager

```
package geneclustertool;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.Date;
import java.util.TreeSet;
import com.itextpdf.text.BadElementException;
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.Image;
import com.itextpdf.text.List;
import com.itextpdf.text.ListItem;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.BaseFont;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;

/**
 * Creates and exports a PDF document of the results
 *
 * @author Bianca Silmaro
 */
public class PDFManager {

        private static final String FONT = "fonts/PT_Sans-Web-Regular.ttf";
        private static final String FONT_BOLD = "fonts/PT_Sans-Web-Bold.ttf";
        private static final String FONT_ITALIC = "fonts/PT_Sans-Web-Italic.ttf";
        private ArrayList<TreeSet<String>> candidateGeneClusters = new ArrayList<TreeSet<String
            >>();
        private ArrayList<ArrayList<Vertex>> sortedGeneClusters = new ArrayList<ArrayList<Vertex
            >>();
    private static Font font, bold;
    private int typeOfGeneClustering, numOfCandidateGeneClusters;
    private static BaseFont bf, bfbold, bfitalic;
    private static GeneClusteringAlgorithm gc;

    public PDFManager(ArrayList<TreeSet<String>> candidateGeneClusters, File file, int
         typeOfGeneClustering, ArrayList<String> genomes, int sizeOfGeneCluster, int missingWeight,
          int additionalWeight,
                int numOfCandidateGeneClusters, ArrayList<ArrayList<Vertex>> sortedGeneClusters,
                     GeneClusteringAlgorithm gc) {
        this.candidateGeneClusters = candidateGeneClusters;
        this.typeOfGeneClustering = typeOfGeneClustering;
        this.sortedGeneClusters = sortedGeneClusters;
        this.numOfCandidateGeneClusters = numOfCandidateGeneClusters;
        this.gc = gc;

        try {
                Document document = new Document();
                PdfWriter.getInstance(document, new FileOutputStream(file));
                document.open();

                // Add fonts
                bf = BaseFont.createFont(FONT, BaseFont.IDENTITY_H, BaseFont.EMBEDDED);
                font = new Font(bf, 12);

                bfbold = BaseFont.createFont(FONT_BOLD, BaseFont.IDENTITY_H, BaseFont.EMBEDDED);
                bold = new Font(bfbold, 12, Font.NORMAL, BaseColor.BLACK);

                bfitalic = BaseFont.createFont(FONT_ITALIC, BaseFont.IDENTITY_H, BaseFont.EMBEDDED
                    );

                addMetaData(document);
                addContent(document, genomes, sizeOfGeneCluster, missingWeight, additionalWeight,
                     numOfCandidateGeneClusters, sortedGeneClusters);
                document.close();
        } catch (Exception e) {
                e.printStackTrace();
        }
    }

    /**
     * Adds the header of the PDF document
     */
    private void addImage(Document document) {
        Image img;
                try {
                        img = Image.getInstance(getClass().getResource("/images/logo.png"));
                document.add(img);
                } catch (IOException | DocumentException e) {
                        e.printStackTrace();
                }
    }

    /**
     * Adds meta data to the PDF document
     */
    private static void addMetaData(Document document) {
        document.addTitle("Clustar Report");
        document.addKeywords("Gene Clusters, Approximate Gene Clustering, Exact Gene Clustering");
```

```
}

/**
 * Adds the content to the PDF document
 * @param document                                  The Document object from iText5
 * @param genomes                                   The list of genomes uploaded
 * @param sizeOfGeneCluster                         The size of gene cluster specified by the
      user
 * @param missingWeight                             The weight for missing genes
      specified by the user
 * @param additionalWeight                          The weight for additional genes specified
      by the user
 * @param numOfCandidateGeneClusters        The number of candidate gene clusters to be
      displayed specified by the user
 * @param sortedGeneClusters                        The list of sorted gene clusters (results
      of the algorithm)
 */
private void addContent(Document document, ArrayList<String> genomes, int sizeOfGeneCluster,
     int missingWeight, int additionalWeight,
             int numOfCandidateGeneClusters, ArrayList<ArrayList<Vertex>> sortedGeneClusters)
                     throws DocumentException {
    Font chapFont = new Font(bfbold, 16);
    Font subFont = new Font(bfitalic, 12);
    Font titleFont = new Font(bfbold, 24);

    addImage(document);
    Paragraph preface = new Paragraph();
    addEmptyLine(preface, 1);

    Paragraph title;
    if(typeOfGeneClustering == 0) {
             title = new Paragraph("A Report On Detecting Gene Clusters Using Exact Gene
                  Clustering", titleFont);
             title.setLeading(15, 0);
    } else if (typeOfGeneClustering == 1) {
             title = new Paragraph("A Report On Detecting Gene Clusters Using Approximate Gene
                  Clustering", titleFont);
             title.setLeading(15, 0);
    } else {
             title = new Paragraph("A Report On Detecting Gene Clusters Using Exact Gene
                  Clustering (with GPU)", titleFont);
             title.setLeading(15, 0);
    }

    preface.setAlignment(Element.ALIGN_CENTER);
    preface.add(title);
    addEmptyLine(preface, 1);

    // Will create: Report generated by: _name, _date
    preface.add(new Paragraph("Report generated by: " + System.getProperty("user.name") + ", "
         + new Date(), font));
    document.add(preface);

    Paragraph content = new Paragraph();
    addEmptyLine(content, 1);

    // First chapter: Summary of Results
    Paragraph summary = new Paragraph("A. Summary of Results", chapFont);
    content.setAlignment(Element.ALIGN_LEFT);
    content.add(summary);

    addEmptyLine(content, 1);

    // Add list of genomes
    Paragraph listOfGenomes = new Paragraph("List of genomes", subFont);
    listOfGenomes.setAlignment(Element.ALIGN_LEFT);
    content.add(listOfGenomes);
    List genomesList = new List(true, false, 10);
    for(int i = 0; i < genomes.size(); i++) {
             genomesList.add(new ListItem("    " + genomes.get(i)));
    }
    content.add(genomesList);

    addEmptyLine(content, 1);

    // Add input parameters
    content.add(new Paragraph("Input parameters", subFont));
    if(typeOfGeneClustering == 1) {
             String sizeOfGeneClusterStr = "Size of gene cluster: " + sizeOfGeneCluster;
             String missingWeightStr = "Weight for missing genes: " + missingWeight;
             String additionalWeightStr = "Weight for missing genes: " + additionalWeight;
             content.add(new Paragraph(sizeOfGeneClusterStr, font));
             content.add(new Paragraph(missingWeightStr, font));
             content.add(new Paragraph(additionalWeightStr, font));

             String numOfCandidateGeneClustersStr = "Number of candidate gene clusters: " +
                  numOfCandidateGeneClusters;
             content.add(new Paragraph(numOfCandidateGeneClustersStr, font));

             addEmptyLine(content, 1);

             // Add list of candidate gene clusters
```

```
            content.add(new Paragraph("List of candidate gene clusters", subFont));
            List candidateGeneClusterList = new List(true, false, 10);
            for(int i = 0; i < candidateGeneClusters.size(); i++) {
                candidateGeneClusterList.add(new ListItem("      " + candidateGeneClusters
                    .get(i).toString()));
            }
            content.add(candidateGeneClusterList);


            document.add(content);
        } else {
            String numOfCandidateGeneClustersStr = "Number of candidate gene clusters: " +
                numOfCandidateGeneClusters;
            content.add(new Paragraph(numOfCandidateGeneClustersStr, font));

            addEmptyLine(content, 1);

            // Add list of candidate gene clusters
            content.add(new Paragraph("List of cliques generated: ", subFont));
            List candidateGeneClusterList = new List(true, false, 10);
            for(int i = 0; i < sortedGeneClusters.size(); i++) {
                for(int j = 0; j < sortedGeneClusters.get(i).size(); j++) {
                candidateGeneClusterList.add(new ListItem("      " + sortedGeneClusters.
                    get(i).get(j).getGeneClusterName()));
                }
            }
            content.add(candidateGeneClusterList);


            document.add(content);
        }

        document.newPage();

        // Second chapter: Alignment of genes
        Paragraph alignment = new Paragraph();
        alignment.add(new Paragraph("B. Alignment of genes", chapFont));
        addEmptyLine(alignment, 1);

        Paragraph subPara;
        // Loop through the candidate gene clusters
        if(typeOfGeneClustering == 1) {
            for(int i = 0; i < sortedGeneClusters.size(); i++) {
                int maxNumOfCols = gc.getMaxNumOfColumns(sortedGeneClusters, i) + 1;
                int index = i + 1;
                subPara = new Paragraph(index + ".   " + candidateGeneClusters.get(i).
                    toString(), subFont);
                alignment.add(subPara);
                // Create table
                createTable(maxNumOfCols, sortedGeneClusters, i, alignment);
                addEmptyLine(alignment, 2);
            }
        } else {
            for(int i = 0; i < sortedGeneClusters.size(); i++) {
                int maxNumOfCols = gc.getMaxNumOfColumns(sortedGeneClusters, i) + 1;
                int index = i + 1;
                subPara = new Paragraph("Clique #" + index, subFont);
                alignment.add(subPara);
                // Create table
                createTable(maxNumOfCols, sortedGeneClusters, i, alignment);
                addEmptyLine(alignment, 2);
            }
        }

        document.add(alignment);
    }

    /**
     * Gets the common genes across all genomes given a gene cluster
     * @param sortedGeneClusters                The list of sorted gene clusters (results from the
         algorithm)
     * @param indexOfBaseGeneCluster    The current base gene cluster being evaluated
     * @return
     */
    private static ArrayList<String> getCommonElements(ArrayList<ArrayList<Vertex>>
        sortedGeneClusters, int indexOfBaseGeneCluster) {
        ArrayList<String> commonElements = new ArrayList<String>();

        Vertex orig_v = sortedGeneClusters.get(indexOfBaseGeneCluster).get(0);
            ArrayList<String> blockOfGenesOrig = gc.getBlockOfGenes(orig_v.getPartitionNumber
                (), orig_v.getStartIndex(), orig_v.getEndIndex());
            commonElements = new ArrayList<String>(blockOfGenesOrig);

            for(int i = 0; i < sortedGeneClusters.get(indexOfBaseGeneCluster).size(); i++) {
            Vertex v = sortedGeneClusters.get(indexOfBaseGeneCluster).get(i);
            ArrayList<String> blockOfGenes = gc.getBlockOfGenes(v.getPartitionNumber(), v.
                getStartIndex(), v.getEndIndex());
            commonElements.retainAll(blockOfGenes);
        }

        return commonElements;
    }
```

```
    /**
     * Creates the table for the alignment of genes per gene cluster
     * @param maxNumOfCols                          The maximum number of columns for the
         table
     * @param sortedGeneClusters                    The list of sorted gene clusters (result from the
         algorithm)
     * @param indexOfBaseGeneCluster     The current gene cluster being evaluated
     * @param alignment                             The current paragraph being used
     */
    private static void createTable(int maxNumOfCols, ArrayList<ArrayList<Vertex>>
         sortedGeneClusters, int indexOfBaseGeneCluster, Paragraph alignment) throws
         BadElementException {
        PdfPTable table = new PdfPTable(maxNumOfCols);
        Font font_red = new Font(bf, 12, Font.NORMAL, BaseColor.RED);

        ArrayList<String> commonElements = getCommonElements(sortedGeneClusters,
             indexOfBaseGeneCluster);

        PdfPCell cell;
        for(int i = 0; i < gc.getAllGenomes().size(); i++) {
                cell = new PdfPCell(new Phrase(gc.getAllGenomes().get(i), font));
                cell.setHorizontalAlignment(Element.ALIGN_CENTER);
                cell.setBorderColor(BaseColor.LIGHT_GRAY);
                table.addCell(cell);

                        for(int j = 0; j < sortedGeneClusters.get(indexOfBaseGeneCluster).size();
                             j++) {
                            Vertex v = sortedGeneClusters.get(indexOfBaseGeneCluster).get(j);
                            if(v.getPartitionNumber() == i) {
                                    ArrayList<String> blockOfGenes = gc.getBlockOfGenes(v.
                                         getPartitionNumber(), v.getStartIndex(), v.getEndIndex
                                         ());

                                    int k;
                                    for(k = 0; k < blockOfGenes.size(); k++) {
                                            if(commonElements.contains(blockOfGenes.get(k))) {
                                                    cell = new PdfPCell(new Phrase(
                                                         blockOfGenes.get(k), font_red));
                                            } else {
                                                    cell = new PdfPCell(new Phrase(
                                                         blockOfGenes.get(k), font));
                                            }
                                            cell.setBorderColor(BaseColor.LIGHT_GRAY);
                                            cell.setPadding(10);
                                            table.addCell(cell);
                                    }
                                    while(k < maxNumOfCols - 1) {
                                            cell = new PdfPCell(new Phrase(" "));
                                            cell.setBorderColor(BaseColor.LIGHT_GRAY);
                                            table.addCell(cell);
                                            k++;
                                    }
                            }
                        }
                }

        alignment.add(table);
    }

    /**
     * Adds new line to a paragraph
     * @param paragraph The current paragraph being used
     * @param number    The number of empty lines to be added
     */
    private static void addEmptyLine(Paragraph paragraph, int number) {
        for (int i = 0; i < number; i++) {
            paragraph.add(new Paragraph(" "));
        }
    }
}
```

Listing 10: Main

```
package geneclustertool;

import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.UIManager;

public class Main {

        public static final void main(String args[]) throws Exception {
                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                                        new Canvas(ImageIO.read(getClass().getResource("/images/
                                             bg2.png")));
```

```
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                });

                }
}
```

Listing 11: combination.cu

```
//#include "cuda_by_example/common/book.h"
//#include <iostream>
//using namespace std;

extern "C" __device__ void recursive(int tid, int numOfPartitions, int *star, int currPartition,
     int *vertices, int *cumSizesOfPartition, int *sizesOfPartition, int *output, long *
     outputSizesOfBaseGeneCluster, int *outputIndex, int lastIndex) {
    if(currPartition == numOfPartitions - 1) {
        int ctr = tid * numOfPartitions;
        for(int i = 0; i < numOfPartitions; i++) {
            output[*outputIndex] = star[ctr];
            ctr = ctr + 1;
            *outputIndex = *outputIndex + 1;
        }
        return;
    } else {
        currPartition++;
        for(int i = 0; i < sizesOfPartition[currPartition]; i++) {
            star[lastIndex + currPartition] = vertices[cumSizesOfPartition[currPartition - 1] + i
                ];
            recursive(tid, numOfPartitions, star, currPartition, vertices, cumSizesOfPartition,
                sizesOfPartition, output, outputSizesOfBaseGeneCluster, outputIndex, lastIndex);
        }
    }
}

extern "C" __global__ void combination(int numOfIndices, int numOfPartitions, int *vertices, int *
     cumSizesOfPartition, int *sizesOfPartition, long *outputSizesOfBaseGeneCluster, int *star, int
      *output, int startIndex) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;

    while(tid < numOfIndices) {
        int lastIndex = tid * numOfPartitions;
        star[lastIndex] = vertices[tid + startIndex];

        int currPartition = 0;
        int outputIndex = 0;
        if(tid != 0) {
            outputIndex = outputSizesOfBaseGeneCluster[tid - 1];
        }

        recursive(tid, numOfPartitions, star, currPartition, vertices, cumSizesOfPartition,
            sizesOfPartition, output, outputSizesOfBaseGeneCluster, &outputIndex, lastIndex);
        tid += blockDim.x * gridDim.x;
    }
}

/*int main() {
    int numOfIndices = 30;
    int numOfIndicesOfBaseGeneClusters = 10;
    int numOfPartitions = 3;
    int outputSize = 3000;
    int starSize = numOfIndicesOfBaseGeneClusters * numOfPartitions;
    int startIndex = 0;

    int indices[30];
    for(int i = 0; i < numOfIndices; i++) {
        indices[i] = i;
    }
    int cumSizesOfPartition[3] = {10, 20, 30};
    int sizesOfPartition[3] = {10, 10, 10};
    int outputSizesOfBaseGeneCluster[10] = {300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700,
        3000};
    int output[3000] = {};
    //int star[30] = {};

    int *dev_indices, *dev_cumSizesOfPartition, *dev_sizesOfPartition, *
        dev_outputSizesOfBaseGeneCluster, *dev_star, *dev_output;
    HANDLE_ERROR(cudaMalloc( (void**)&dev_indices, numOfIndices * sizeof(int)) );
    HANDLE_ERROR(cudaMalloc( (void**)&dev_cumSizesOfPartition, numOfPartitions * sizeof(int)) );
    HANDLE_ERROR(cudaMalloc( (void**)&dev_sizesOfPartition, numOfPartitions * sizeof(int)) );
    HANDLE_ERROR(cudaMalloc( (void**)&dev_outputSizesOfBaseGeneCluster,
        numOfIndicesOfBaseGeneClusters * sizeof(int)) );
    HANDLE_ERROR(cudaMalloc( (void**)&dev_star, starSize * sizeof(int)) );
    HANDLE_ERROR(cudaMalloc( (void**)&dev_output, outputSize * sizeof(int)) );

    HANDLE_ERROR(cudaMemcpy(dev_indices, indices, numOfIndices * sizeof(int),
        cudaMemcpyHostToDevice));
```

```
    HANDLE_ERROR(cudaMemcpy(dev_cumSizesOfPartition, cumSizesOfPartition, numOfPartitions * sizeof
        (int), cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpy(dev_sizesOfPartition, sizesOfPartition, numOfPartitions * sizeof(int),
         cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpy(dev_outputSizesOfBaseGeneCluster, outputSizesOfBaseGeneCluster,
        numOfIndicesOfBaseGeneClusters * sizeof(int), cudaMemcpyHostToDevice));

    combination<<<2,128>>>(numOfIndicesOfBaseGeneClusters, numOfPartitions, dev_indices,
        dev_cumSizesOfPartition, dev_sizesOfPartition, dev_outputSizesOfBaseGeneCluster, dev_star,
         dev_output, startIndex);

    // Copy the array output to host
    HANDLE_ERROR(cudaMemcpy(output, dev_output, outputSize * sizeof(int), cudaMemcpyDeviceToHost))
        ;

    for(int i = 0; i < outputSize; i++) {
        cout << output[i] << " ";
        if((i+1)%numOfPartitions == 0) {
            cout << "\n";
        }
    }
    //cout << numOfIndicesOfBaseGeneClusters;

    //combination(numOfIndices, numOfIndicesOfBaseGeneClusters, numOfPartitions, indices,
        indicesOfBaseGeneClusters, cumSizesOfPartition, sizesOfPartition,
        outputSizesOfBaseGeneCluster, star, output);
}*/
```

Listing 12: combination.ptx

```
//
// Generated by NVIDIA NVVM Compiler
//
// Compiler Build ID: CL-23083092
// Cuda compilation tools, release 9.1, V9.1.85
// Based on LLVM 3.4svn
//

.version 6.1
.target sm_30
.address_size 64


.func recursive(
        .param .b32 recursive_param_0,
        .param .b32 recursive_param_1,
        .param .b64 recursive_param_2,
        .param .b32 recursive_param_3,
        .param .b64 recursive_param_4,
        .param .b64 recursive_param_5,
        .param .b64 recursive_param_6,
        .param .b64 recursive_param_7,
        .param .b64 recursive_param_8,
        .param .b32 recursive_param_9
)
{
        .reg .pred      %p<10>;
        .reg .b32       %r<74>;
        .reg .b64       %rd<46>;


        ld.param.u32    %r27, [recursive_param_0];
        ld.param.u32    %r28, [recursive_param_1];
        ld.param.u64    %rd11, [recursive_param_2];
        ld.param.u32    %r29, [recursive_param_3];
        ld.param.u64    %rd12, [recursive_param_4];
        ld.param.u64    %rd13, [recursive_param_5];
        ld.param.u64    %rd14, [recursive_param_6];
        ld.param.u64    %rd15, [recursive_param_7];
        ld.param.u64    %rd16, [recursive_param_8];
        ld.param.u32    %r30, [recursive_param_9];
        cvta.to.global.u64      %rd1, %rd15;
        cvta.to.global.u64      %rd2, %rd11;
        cvta.to.local.u64       %rd3, %rd16;
        add.s32         %r31, %r28, -1;
        setp.eq.s32     %p1, %r31, %r29;
        @%p1 bra        BB0_4;
        bra.uni         BB0_1;

BB0_4:
        mul.lo.s32      %r4, %r28, %r27;
        setp.lt.s32     %p4, %r28, 1;
        @%p4 bra        BB0_16;

        and.b32         %r5, %r28, 3;
        setp.eq.s32     %p5, %r5, 0;
        mov.u32         %r73, 0;
        @%p5 bra        BB0_13;

        setp.eq.s32     %p6, %r5, 1;
```

```
        @%p6 bra        BB0_11;
        bra.uni         BB0_7;

BB0_11:
        ld.local.u32    %r67, [%rd3];
        mov.u32         %r68, 0;
        bra.uni         BB0_12;

BB0_1:
        cvta.to.global.u64      %rd17, %rd14;
        mul.wide.s32    %rd18, %r29, 4;
        add.s64         %rd4, %rd17, %rd18;
        ld.global.u32   %r32, [%rd4+4];
        setp.lt.s32     %p2, %r32, 1;
        @%p2 bra        BB0_16;

        cvta.to.global.u64      %rd5, %rd12;
        cvta.to.global.u64      %rd19, %rd13;
        add.s64         %rd6, %rd19, %rd18;
        add.s32         %r34, %r29, %r30;
        mul.wide.s32    %rd21, %r34, 4;
        add.s64         %rd7, %rd2, %rd21;
        mov.u32         %r63, 0;

BB0_3:
        ld.global.u32   %r35, [%rd6];
        add.s32         %r36, %r63, %r35;
        mul.wide.s32    %rd22, %r36, 4;
        add.s64         %rd23, %rd5, %rd22;
        ld.global.u32   %r37, [%rd23];
        st.global.u32   [%rd7+4], %r37;
        // Callseq Start 0
        {
        .reg .b32 temp_param_reg;
        // <end>}
        .param .b32 param0;
        st.param.b32    [param0+0], %r27;
        .param .b32 param1;
        st.param.b32    [param1+0], %r28;
        .param .b64 param2;
        st.param.b64    [param2+0], %rd11;
        .param .b32 param3;
        add.s32         %r62, %r29, 1;
        st.param.b32    [param3+0], %r62;
        .param .b64 param4;
        st.param.b64    [param4+0], %rd12;
        .param .b64 param5;
        st.param.b64    [param5+0], %rd13;
        .param .b64 param6;
        st.param.b64    [param6+0], %rd14;
        .param .b64 param7;
        st.param.b64    [param7+0], %rd15;
        .param .b64 param8;
        st.param.b64    [param8+0], %rd16;
        .param .b32 param9;
        st.param.b32    [param9+0], %r30;
        call.uni
        recursive,
        (
        param0,
        param1,
        param2,
        param3,
        param4,
        param5,
        param6,
        param7,
        param8,
        param9
        );

        //{
        }// Callseq End 0
        ld.global.u32   %r38, [%rd4+4];
        add.s32         %r63, %r63, 1;
        setp.lt.s32     %p3, %r63, %r38;
        @%p3 bra        BB0_3;
        bra.uni         BB0_16;

BB0_7:
        setp.eq.s32     %p7, %r5, 2;
        @%p7 bra        BB0_9;
        bra.uni         BB0_8;

BB0_9:
        ld.local.u32    %r64, [%rd3];
        mov.u32         %r68, 1;
        bra.uni         BB0_10;

BB0_8:
        mul.wide.s32    %rd24, %r4, 4;
        add.s64         %rd25, %rd2, %rd24;
```

```
        ld.global.u32   %r41, [%rd25];
        ld.local.u32    %r42, [%rd3];
        mul.wide.s32    %rd26, %r42, 4;
        add.s64         %rd27, %rd1, %rd26;
        st.global.u32   [%rd27], %r41;
        ld.local.u32    %r43, [%rd3];
        add.s32         %r64, %r43, 1;
        st.local.u32    [%rd3], %r64;
        add.s32         %r4, %r4, 1;
        mov.u32         %r68, 2;

BB0_10:
        mul.wide.s32    %rd28, %r4, 4;
        add.s64         %rd29, %rd2, %rd28;
        ld.global.u32   %r45, [%rd29];
        mul.wide.s32    %rd30, %r64, 4;
        add.s64         %rd31, %rd1, %rd30;
        st.global.u32   [%rd31], %r45;
        ld.local.u32    %r46, [%rd3];
        add.s32         %r67, %r46, 1;
        st.local.u32    [%rd3], %r67;
        add.s32         %r4, %r4, 1;

BB0_12:
        mul.wide.s32    %rd32, %r4, 4;
        add.s64         %rd33, %rd2, %rd32;
        ld.global.u32   %r48, [%rd33];
        mul.wide.s32    %rd34, %r67, 4;
        add.s64         %rd35, %rd1, %rd34;
        st.global.u32   [%rd35], %r48;
        ld.local.u32    %r49, [%rd3];
        add.s32         %r50, %r49, 1;
        st.local.u32    [%rd3], %r50;
        add.s32         %r4, %r4, 1;
        add.s32         %r73, %r68, 1;

BB0_13:
        setp.lt.u32     %p8, %r28, 4;
        @%p8 bra        BB0_16;

        ld.local.u32    %r72, [%rd3];
        mul.wide.s32    %rd36, %r4, 4;
        add.s64         %rd45, %rd2, %rd36;

BB0_15:
        ld.global.u32   %r51, [%rd45];
        mul.wide.s32    %rd37, %r72, 4;
        add.s64         %rd38, %rd1, %rd37;
        st.global.u32   [%rd38], %r51;
        ld.local.u32    %r52, [%rd3];
        add.s32         %r53, %r52, 1;
        st.local.u32    [%rd3], %r53;
        ld.global.u32   %r54, [%rd45+4];
        mul.wide.s32    %rd39, %r53, 4;
        add.s64         %rd40, %rd1, %rd39;
        st.global.u32   [%rd40], %r54;
        ld.local.u32    %r55, [%rd3];
        add.s32         %r56, %r55, 1;
        st.local.u32    [%rd3], %r56;
        ld.global.u32   %r57, [%rd45+8];
        mul.wide.s32    %rd41, %r56, 4;
        add.s64         %rd42, %rd1, %rd41;
        st.global.u32   [%rd42], %r57;
        ld.local.u32    %r58, [%rd3];
        add.s32         %r59, %r58, 1;
        st.local.u32    [%rd3], %r59;
        ld.global.u32   %r60, [%rd45+12];
        mul.wide.s32    %rd43, %r59, 4;
        add.s64         %rd44, %rd1, %rd43;
        st.global.u32   [%rd44], %r60;
        ld.local.u32    %r61, [%rd3];
        add.s32         %r72, %r61, 1;
        st.local.u32    [%rd3], %r72;
        add.s64         %rd45, %rd45, 16;
        add.s32         %r73, %r73, 4;
        setp.lt.s32     %p9, %r73, %r28;
        @%p9 bra        BB0_15;

BB0_16:
        ret;
}

        // .globl       combination
.visible .entry combination(
        .param .u32 combination_param_0,
        .param .u32 combination_param_1,
        .param .u64 combination_param_2,
        .param .u64 combination_param_3,
        .param .u64 combination_param_4,
        .param .u64 combination_param_5,
        .param .u64 combination_param_6,
        .param .u64 combination_param_7,
```

```
        .param .u32 combination_param_8
)
{
        .local .align 4 .b8     __local_depot1[4];
        .reg .b64       %SP;
        .reg .b64       %SPL;
        .reg .pred      %p<4>;
        .reg .b32       %r<20>;
        .reg .b64       %rd<20>;


        mov.u64         %rd19, __local_depot1;
        cvta.local.u64  %SP, %rd19;
        ld.param.u32    %r7, [combination_param_0];
        ld.param.u32    %r8, [combination_param_1];
        ld.param.u64    %rd5, [combination_param_2];
        ld.param.u64    %rd6, [combination_param_3];
        ld.param.u64    %rd7, [combination_param_4];
        ld.param.u64    %rd8, [combination_param_5];
        ld.param.u64    %rd9, [combination_param_6];
        ld.param.u64    %rd10, [combination_param_7];
        ld.param.u32    %r9, [combination_param_8];
        add.u64         %rd11, %SP, 0;
        cvta.to.local.u64       %rd1, %rd11;
        mov.u32         %r1, %ntid.x;
        mov.u32         %r10, %ctaid.x;
        mov.u32         %r11, %tid.x;
        mad.lo.s32      %r19, %r1, %r10, %r11;
        setp.ge.s32     %p1, %r19, %r7;
        @%p1 bra        BB1_5;

        cvta.to.global.u64      %rd2, %rd8;
        cvta.to.global.u64      %rd3, %rd9;
        cvta.to.global.u64      %rd4, %rd5;
        mov.u32         %r12, %nctaid.x;
        mul.lo.s32      %r3, %r12, %r1;

BB1_2:
        add.s32         %r13, %r19, %r9;
        mul.wide.s32    %rd12, %r13, 4;
        add.s64         %rd13, %rd4, %rd12;
        ld.global.u32   %r14, [%rd13];
        mul.lo.s32      %r5, %r19, %r8;
        mul.wide.s32    %rd14, %r5, 4;
        add.s64         %rd15, %rd3, %rd14;
        st.global.u32   [%rd15], %r14;
        mov.u32         %r15, 0;
        st.local.u32    [%rd1], %r15;
        setp.eq.s32     %p2, %r19, 0;
        @%p2 bra        BB1_4;

        add.s32         %r16, %r19, -1;
        mul.wide.s32    %rd16, %r16, 4;
        add.s64         %rd17, %rd2, %rd16;
        ld.global.u32   %r17, [%rd17];
        st.local.u32    [%rd1], %r17;

BB1_4:
        // Callseq Start 1
        {
        .reg .b32 temp_param_reg;
        // <end>}
        .param .b32 param0;
        st.param.b32    [param0+0], %r19;
        .param .b32 param1;
        st.param.b32    [param1+0], %r8;
        .param .b64 param2;
        st.param.b64    [param2+0], %rd9;
        .param .b32 param3;
        st.param.b32    [param3+0], %r15;
        .param .b64 param4;
        st.param.b64    [param4+0], %rd5;
        .param .b64 param5;
        st.param.b64    [param5+0], %rd6;
        .param .b64 param6;
        st.param.b64    [param6+0], %rd7;
        .param .b64 param7;
        st.param.b64    [param7+0], %rd10;
        .param .b64 param8;
        st.param.b64    [param8+0], %rd11;
        .param .b32 param9;
        st.param.b32    [param9+0], %r5;
        call.uni
        recursive,
        (
        param0,
        param1,
        param2,
        param3,
        param4,
        param5,
        param6,
```

```
        param7,
        param8,
        param9
        );

        //{
        }// Callseq End 1
        add.s32         %r19, %r3, %r19;
        setp.lt.s32     %p3, %r19, %r7;
        @%p3 bra        BB1_2;

BB1_5:
        ret;
}
```

XI. Acknowledgement

First and foremost, I would like to thank God Almighty for giving me the strength, wisdom, opportunities, and resources to help me succeed. I owe everything to Him.

To my parents and family, thank you for being very supportive and for believing that I can do anything. Thank you for reassuring me that everything will be okay and for giving me inspiration to finish my SP successfully. I know you always asked if I'm graduating with honors and I was so afraid I wouldn't be able to pass your expectations, but I want to thank you because I used this as an inspiration and motivation to finish my SP and to graduate on time. Thank you, and because of this, I dedicate my work to you.

I also want to acknowledge my friends and my blockmates. Thank you for inspiring me to work hard for my SP. Thank you for telling me not to give up. Thank you for being there whenever I needed help. Thank you for giving me hugs and emotional support when I feel like won't be able to do it. Thank you for being that safe space I run to whenever I doubt myself. Surviving 4 years in UP wasn't a smooth ride, but thank you for riding it with me.

I also want to thank the DPSM Chemistry professors who patiently taught and answered my questions about genes and gene clusters. Thank you for being there during the early stages of developing my SP, I would not have fully understood these concepts without your help.

To Sir Jeff Aborot and to DOST-ASTI, thank you for being so accomodating. I am truly grateful for our meetings and sessions; I know you are busy so I really want to thank you sir for being available whenever I have questions. Learning the concepts behind parallel programming was hard given the time constraints, but I was able to pull it off just fine, thanks to your undying support and help.

To Sir Marvin Ignacio, thank you for lending your machine for the demo of my tool.

To my adviser and greatest motivator, Sir Geoff Solano, thank you for believing in me. Thank you sir for inspiring me to work hard and for teaching me to have a mindset of creating something that would eventually help the people. Thank you for being the best adviser, for making time for us, for being an emotional and academic support in completing this SP, and for providing guidance and life advice.

And to everyone who has helped and supported me along the way, thank you.