A Constraint Logic Programming Approach to Predicting the Three-Dimensional Yeast Genome

A Thesis Submitted to the

College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon

By

Kimberly MacKay

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science

176 Thorvaldson Building

110 Science Place

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

S7N 5C9

Abstract

In order for all of a cell's genetic information to fit inside its nucleus, the chromosomes must undergo extensive folding and organization. Just like in origami where the same piece of paper folded in different ways allows the paper to take on different forms and potential functions, it is possible that different genomic organizations (or architectures) are related to various nuclear functions. Until recently, it has been impossible to comprehensively investigate this relationship due to the lack of high-resolution and high-throughput techniques for identifying genomic architectures. The recent development of a technique called Hi-C, which is a derivation of chromosome conformation capture, has made it possible to detect the complete set of interactions occurring within (intra-interactions) and between (inter-interactions) chromosomes in the nucleus. Many computational methods have been proposed that use these analytical results to infer the rough three-dimensional (3D) architecture of the genome. However, the genomic architecture also impacts additional types of nuclear interactions and techniques exist that are able to capture and measure these interactions. Unfortunately, it is difficult to incorporate these additional datasets into the existing tools. To overcome this, a novel application of constraint logic programming (CLP) was used to develop a new program for the prediction of the 3D genomic architecture. The unique representation used in this program lends itself well to the future incorporation of additional genomic datasets. This thesis investigates the most efficient way to date to represent and optimally solve the constraint satisfaction problem of the 3D genome. The developed program was used to predict a 3D logical model of the fission yeast genome and the results were visualized using Cytoscape. This model was then biologically validated through literature search which verified that the prediction was able to recapitulate key documented features of the yeast genome. Future work will utilize this tool as a computational framework and extend it to incorporate additional genomic datasets and information into the prediction and visualization of the 3D genomic architecture. The development of the CLP program described here is a step towards a better understanding of the elusive relationship between the 3D structure of the genome and various nuclear functions.

Acknowledgements

I would like to thank my supervisor Dr. Anthony Kusalik and my biological collaborator Dr. Christopher Eskiw for their continued patience, guidance and encouragement over the last two years. Additionally, I would like to thank my external examiner Dr. Steve Robinson and my committee members (Dr. Anthony Kusalik, Dr. Christopher Eskiw and Dr. Ian McQuillian) for their insight and participating in my thesis defence. Funding for my Master's Degree was provided by the Department of Computer Science, the College of Graduate Studies and Research, the Government of Saskatchewan and the Natural Sciences and Engineering Research Council of Canada.

Contents

List of Tables

List of Figures

List of Abbreviations

3C	Chromosome Conformation Capture
3D	Three-Dimensional
A	Adenine
C	Cytosine
CLP	Constraint Logic Programming
CT-IC	Chromosome Territory - Interchromatin Compartment
D coefficient	Dynamics Coefficient
DNA	Deoxyribonucleic Acid
FISH	Fluorescence *in situ* Hybridization
GEO	Gene Expression Omnibus
GFD	Geocode Solver for Finite Domains
G	Guanine
H. sapiens	*Homo sapiens*
IC	Interval Constraints
ICN	Interchromatin Network
PCR	Polymerase Chain Reaction
SNP	Single Nucleotide Polymorphism
S. pombe	*Schizosaccharomyces pombe*
T	Thymine
TAD	Topologically Associating Domain

Chapter 1

Introduction

A prominent unanswered question in biology is how the three-dimensional (3D) structure of a cell's genetic information affects various nuclear functions. A variety of microscopy techniques have been used to infer the architectural organization of the genome within the nucleus [63]. Unfortunately, these techniques have not been able to capture the complete genomic structure since they are low-resolution and low-throughput. Despite this, microscopy has provided interesting and invaluable insights into genome organization. For instance, it has established that chromosomes preferentially compartmentalize into distinct territories within the nucleus (known as chromosome territories), demonstrating a non-random spatial organization of the genome [6].

It is currently unknown whether the 3D genomic organization drives various nuclear functions or vice versa. Alterations in the 3D organization of chromosome territories have been demonstrated in a wide variety of cellular processes, including differentiation [31], serum response [38], therapeutic response [39] and response to DNA damage [40]. The unique spatial organization of the genome that is seen under these different cellular conditions is hypothesized to be a crucial mechanism driving various nuclear and cellular functions. It has been theorized that this dynamic organization of the genome may be driven by the global regulation of gene expression [1]. This hypothesis suggests that groups of genes which share common regulatory mechanisms for controlling gene expression, such as pools of transcription factors, will come together in 3D space and physically interact [13, 54]. The interactions of actively transcribed genes are mediated through their association with specialized structures called transcription factories [49, 51].

The recent development of a biological technique called Hi-C [36] has allowed for the genome-wide detection of DNA regions in close 3D proximity. This proximity is often referred to as an "interaction" between the genomic regions. A multitude of computational methods have been developed that infer a crude 3D

structure of the genome using solely Hi-C results [4, 26, 34, 74]. Currently, none of these existing methods utilize a constraint logic programming approach for modelling and predicting the 3D genomic architecture. It is well known that constraints are a powerful tool for modelling many problems [28] and they have also been successfully applied to the prediction of the 3D structure for other biological components [21, 22, 24, 48]. Additionally, CLP has many advantages for representing and solving biological problems including concise programs, a close relationship between program specifications and expert knowledge and the ability to rapidly infer biological meaning from successful results. This thesis utilizes a constraint logic programming paradigm to develop an efficient 3D genome prediction program. The predicted models were biologically validated through literature search.

The long range chromatin interactions detected by Hi-C are not the only type of interactions that occur in the genome; experimental techniques exist that are able to capture additional types of nuclear interactions and information such as transcription factor binding [5], DNA-lamin interactions [47] and genomic chemical modifications (epigenetic modifications) [5, 41]. Unfortunately, these additional datasets are under-utilized by the current methods for predicting the 3D genomic architecture and the existing tools cannot be easily extended to allow for this integration. Combining these additional data sources into the prediction and visualization of the 3D genome structure will allow researchers a more comprehensive look into the interplay of various genomic factors and modifications in 3D space. Our hypothesis is that by combining these additional data sources into the prediction of the 3D genome structure, a model that is more accurate, information rich and biologically relevant can be produced.

The unique spatial organization of the genome seen under different cellular conditions is hypothesized to be a crucial mechanism driving various nuclear functions such as transcription [20] and DNA-repair [42]. Currently, it is difficult to test this theory due to the lack of computational methods that can integrate and visualize additional genomic datasets and information with the predicted 3D genome structures. The main aim of this thesis was to determine the most effective way to represent and solve the constraint satisfaction problem of the 3D genome using constraint logic programming. The structural models produced by this method were verified through literature search to determine the biological accuracy of the predicted models. Future work will focus on additional biological validation of the models (by our collaborators, Dr. Christopher

Eskiw and Dr. Troy Harkness) as well as extending the developed programs to accommodate multiple types of input datasets into the prediction and visualization of 3D genomic architectures. Overall, the method developed here is a step towards a better understanding of the relationship between 3D genomic structure and nuclear functions.

Chapter 2

Background

2.1 Genome Organization and Topology

The genome is the complete set of an organism's genetic information. It is a collection of cellular instructions required for an organism's development, growth and maintenance [8]. These instructions are encoded through repeating units of the nucleotide bases: adenine, thymine, guanine and cytosine (A, T, G and C, respectively). When these bases are covalently linked together, they form long molecular strands collectively known as deoxyribonucleic acid (DNA) [46]. In general, genomic DNA can be categorized into two main components which vary in their nuclear functions: genes (also known as coding regions) and non-coding regions. In general, genes provide the instructions for (or code for) the construction of proteins, whereas non-coding regions generally provide regulatory functions within the nucleus [46].

Different organisms can vary drastically in their genome size and the number of corresponding genomic features (such as the number of genes, chromosomes and genomic copies) [8]. For instance, it is possible for organisms to have more then one copy of their genome present in each cell. The number of genomic copies a cell contains is referred to as the organism's ploidy. Humans are considered diploid organisms since they have two genomic copies in their non-reproductive cells, whereas fission yeast are usually considered haploid organisms since they generally have only one genomic copy [8]. While the genome does include non-nuclear DNA, such as mitochondrial and chloroplast DNA, for the purpose of this thesis, references to "the genome" will only include nuclear (or nucleoid) DNA.

In bioinformatics, the genome is often abstracted as a set of strings using an alphabet of the four letters: A, T, C and G (Figure 2.1A). This is often how the results of biological experiments, like sequencing, are encoded. While informative, this type of genomic representation lacks crucial information about the natural structure

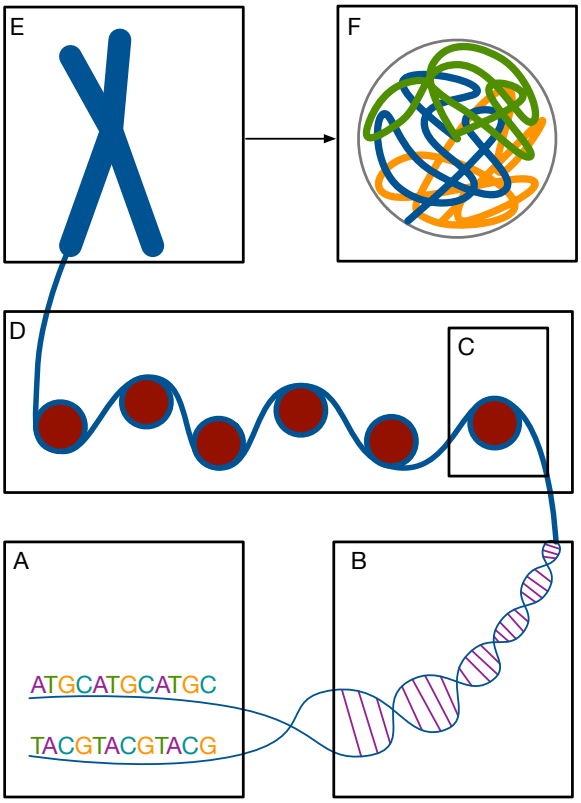

Figure 2.1: A subset of the different levels of genomic structural organization. Panel A represents the two linear DNA strands with the various nucleotides labeled as A, T, C and G (adenine, thymine, cytosine and guanine, respectively). Panel B represents a naked double stranded DNA helix, where the blue lines are the DNA backbones and the purple lines represent Watson-Crick complementary base pairs. Panel C represents a single nucleosome (DNA complexed with histone proteins). Panel D represents the "beads on a string" model of chromatin. Panel E represents a single chromosome. Panel F is a representation of the genome within the nucleus/nucleoid. The grey outline represents the nuclear boundary while the green, blue and orange lines represent individual chromosomes.

of DNA within the cell. In reality, the genome contains multiple levels of structural organization, from linear DNA strands all the way up to the 3D genomic organization. A slightly more accurate representation of cellular DNA structure is shown in Figure 2.1B. This panel depicts a simplified structure of a double helix, where two strands of DNA are connected through the interactions between Watson-Crick complementary base pairs to form a "twisted ladder". Within the cell, this ladder is then wrapped around a set of proteins called histones, forming a structure known as a nucleosome (Figure 2.1C) [70]. When multiple nucleosomes are formed on a strand of DNA, the structure is collectively known as chromatin [8]. Within the nucleus, chromatin can be found in either a relaxed structure known as euchromatin, which is typically associated with genes undergoing active transcription, or in a compacted structure known as heterochromatin, which is

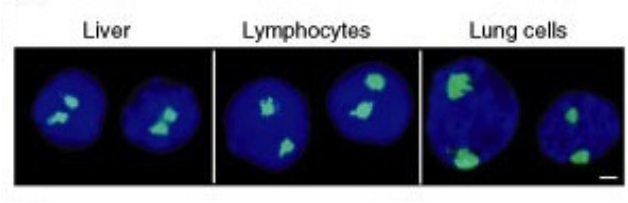

Figure 2.2: Imaging of Chromosome Territories Using Fluorescence *in situ* Hybridization. This figure depicts the nuclear localization of chromosome 5 in different *H. sapiens* tissues from reference [50]. The blue areas in the image are cellular nuclei while the green areas are the two copies of chromosome 5. Within liver cells, chromosome 5 is preferentially located within the interior of the nuclear volume, while in lymphocytes and lung cells it is found on the periphery of the nuclear volume. Specific methodologies and more details can be found in reference [50].

generally associated with genes that are not being actively transcribed [70]. The relaxed chromatin structure is often referred to as the "beads on a string" model (Figure 2.1D) while the compacted chromatin structure is often referred to as a 30 nm fibre [70]. Chromatin can then be further condensed into a structure known as the 300 nm fibre which is then packaged into chromosomes (Figure 2.1E). Many organisms have multiple chromosomes within their genome, which will interact with each other within the nuclear volume (the area within the nuclear membrane or nucleoid region — Figure 2.1F).

Recently, relatively new biological techniques have revealed key features of genomic structure and organization within and between chromosomes in the nuclear volume. These techniques have uncovered the existence of genomic structures called topologically associating domains (TADs) and chromosome territories. TADs are linear regions of DNA where intra-chromosomal interactions (DNA-DNA interactions that occur between regions of a DNA strand) occur at a higher frequency when compared to the surrounding chromosomal region [18]. They are loosely conserved structures [11] that are hypothesized to be a basic unit of genome folding [11] and are associated with gene regulatory features and transcription factories [15]. Chromosome territories are distinct, non-random areas within the nucleus that chromosomes will preferentially compartmentalize into (Figure 2.2) [6, 43, 50]. The location of chromosome territories within the nucleus is dynamic and dependent on a the cell's internal and external environment [6].

Investigations into the structure-function relationship of various macromolecules has been a fundamental aspect of molecular biology research for many years. Mainly, this research has revolved around determining the relationship between the 3D structure of proteins and their various functions. Recently, this research

area has been extended to include investigations into the structure-function relationship of the genome. The 3D structure and organization of the genome is known to be dynamic and complex. Research in this area has led to multiple hypotheses about how the genome may be organized in 3D space including the chromosome territory - interchromatin compartment (CT-IC) model [14], the lattice model [62] and the interchromatin network (ICN) model [7]. Briefly, the CT-IC model suggests that large channels exist between chromosome territories called inter-chromatin compartments. This model restricts gene transcription to occur exclusively on the periphery of chromosome territories within the inter-chromatin compartments [14]. Alternatively, the lattice model loosens this restriction by suggesting that transcription can occur both within a chromosome territory as well as on the periphery [62]. Finally, the ICN model completely removes the separation between chromosome territories and inter-chromatin compartments stating instead that the chromatin fibres can intermingle and make both intra- (within) and inter- (between) chromosomal interactions within the nuclear volume [7].

2.2 Biological Techniques for Detecting Genome Organization and Topology

Recently, the development of a technique called chromosome conformation capture (3C) [16] has allowed researchers to query if two genes (or DNA elements) are in close physical proximity. The close 3D spatial proximity of two genomic regions is often referred to as an "interaction" between the two regions, or a long-range chromatin interaction. 3C is a proximity-based ligation reaction, which uses chemical cross-linking to preserve the genome architecture (Figure 2.3). Briefly, (1) cells are fixed with formaldehyde in order to covalently cross-link genomic regions that are in close 3D proximity. (2) The cross-linked fragments are then digested with a restriction enzyme to remove the potentially large non-interacting interconnecting segments of DNA. (3) Digested fragments are ligated together. (4) The initial cross-linking is removed resulting in DNA fragments that represent the two genomic regions that form an interaction. (5) DNA fragments are purified and polymerase chain reaction (PCR) with region specific primers is used to detect interacting genomic regions. Although elegant in its design, 3C requires a priori identification of candidate regions

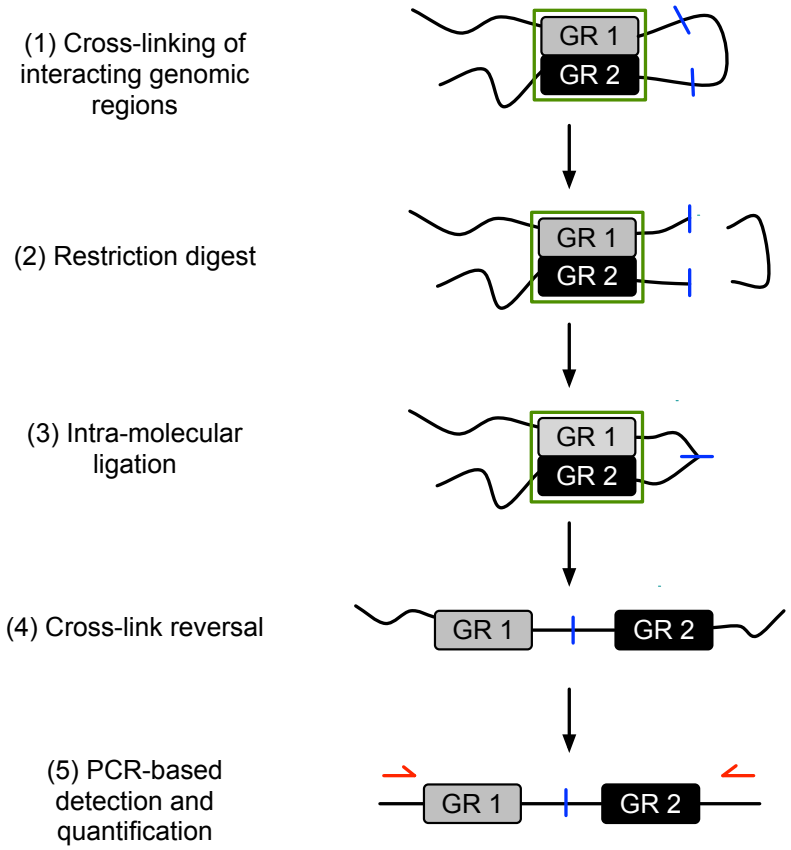

Figure 2.3: A Simplified Overview of the Chromosome Conformation Capture Protocol adapted from reference [12]. GR stands for "Genomic Region". The blue lines represent the location of a restriction enzyme cut site. The green boxes represent a pair of genomic regions being chemically cross-linked together. The red arrows represent the location and direction of a theoretical set of primers that would be required for detecting an interaction between GR 1 and GR 2.

suspected of interacting within the nucleus in order to design region specific primers that are able to identify the interaction.

Hi-C [36] is a biological technique derived from 3C that utilizes next generation sequencing technologies to identify all the DNA-DNA interactions occurring within the genome. An overview of the experimental procedure is depicted in Figure 2.4. Briefly, steps (1), (2), (4) and (5) are similar to the 3C protocol described above. Prior to step (3), the sticky ends generated through the restriction digest in step (2) are filled with biotinylated nucleotides before blunt end ligation. The biotinylated products are then purified using streptavidin beads allowing for the detection of fragments that were cut by restriction enzymes. Sequencing primers are then ligated to the ends of the purified fragments prior to library amplification. Paired-end

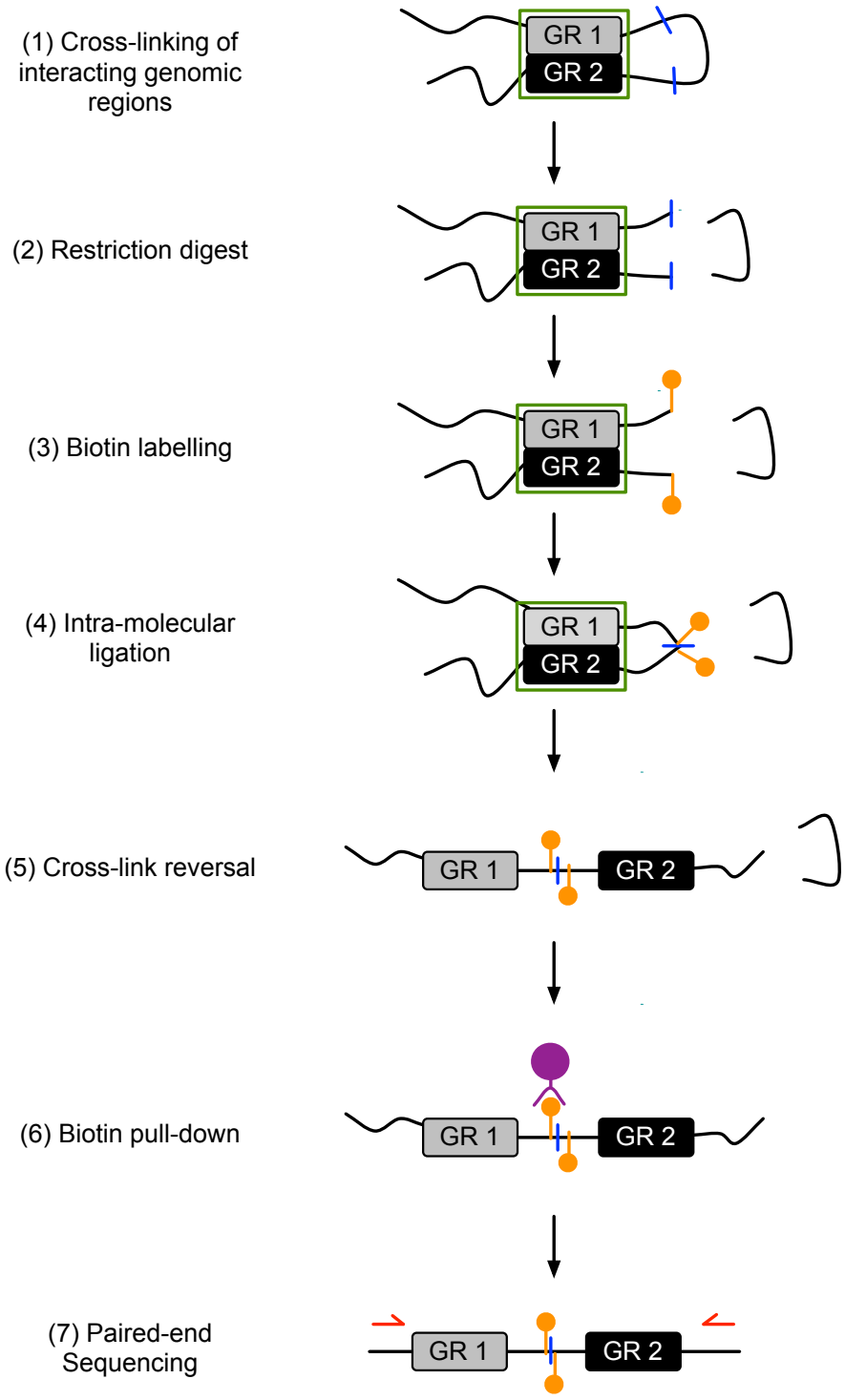

Figure 2.4: A Simplified Overview of the Hi-C Protocol adapted from reference [36]. GR stands for "Genomic Region". The blue lines represent the location of a restriction enzyme cut site. The green boxes represent a pair of genomic regions being chemically cross-linked together. The orange circles represent biotin. The purple symbol represents a streptavidin bead that can be used to purify molecules with a biotin label. The red arrows represent the primers that would be required for paired-end sequencing.

sequencing is then performed and the resultant reads are mapped to a reference genome using a Hi-C specific read mapper [2]. Since the reads generated from Hi-C can be comprised of sequences from linearly distant regions of the genome, these read mappers must be able to split and map reads to distal genomic locations. Overall, Hi-C has enabled the high-resolution detection of a set of inter- and intra-chromosomal interactions (outlined in Figure 2.5).

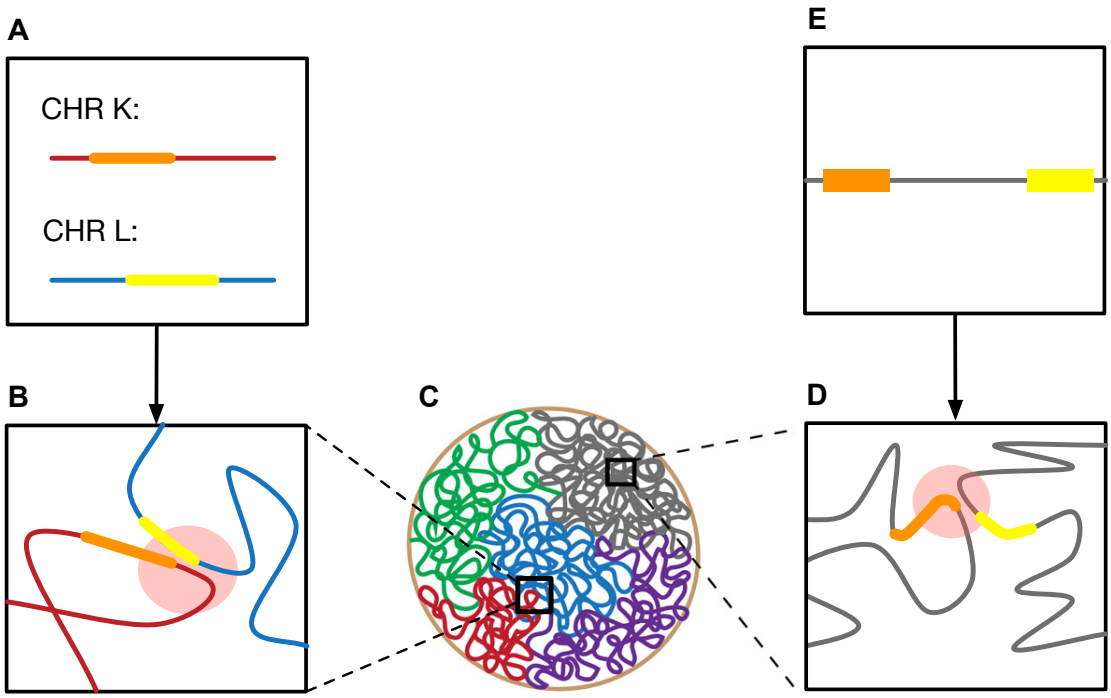

Figure 2.5: A representation of the DNA-DNA interactions that can occur within the 3D genome structure. Panels give the following representations. A: the linear locations of the genes undergoing an inter-chromosomal (trans) interaction between two hypothetical chromosomes, K and L. B: an inter-chromosomal interaction. C: the nucleus with the coloured lines representing the separate chromosomes. D: an intra-chromosomal (cis) interaction. These genes might be linearly "distant" but still have a detectable interaction in 3D space. E: the linear locations of the genes that are undergoing a 3D intra-chromosomal interaction. The orange and yellow regions in panels B, C, D and E are examples of possible gene locations. The red circles in panels B and C represent the genomic regions involved in an interaction.

Whole-genome contact maps are generated by mapping the raw sequence reads produced from Hi-C experiments. A contact map is a matrix ($\mathbf{A}$) of size $N \times N$ where N is the number of genomic "bins" (linear regions of genomic DNA) that the genome is separated into. For instance, a Hi-C experiment in yeast that is able to attain 10 kB resolution will generate 1258 genomic bins, where each bin represents roughly 10 kB of linear DNA sequence. In general, the number of genomic bins is approximately equal to the total

genome size divided by the Hi-C experimental resolution. Each cell ($A_{i,j}$) of $\mathbf{A}$ records how many times the genomic bin i was found to interact with the genomic bin j. This is often referred to as the frequency of the interaction between A_i and A_j. Systematic biases within the whole-genome contact map are removed by normalizing the interaction frequencies using a method similar to hicpipe [71]. In general, the normalized interaction frequency for a cell of the whole-genome contact map ($A_{i,j}norm$) is equal to the number of observed interactions ($A_{i,j}observed$) divided by the number of expected interactions ($A_{i,j}expected$).

Whole-genome contact maps are characteristically sparse and symmetric along the diagonal. An example of a small contact map is shown in Figure 2.6. Contact maps can be generated from a single-cell or a population of cells. Population Hi-C experiments produce whole-genome contact maps that are composed of a finite number of "interaction profiles". Each interaction profile is a matrix ($\mathbf{P}$) the same size as the whole-genome contact map. Similarly to the whole-genome contact map, the individual cells of each interaction profile ($P_{i,j}$) represent the frequency of an interaction between genomic region i and genomic region j. These frequencies will be found in the whole-genome contact map such that $\mathbf{A} = \sum_k \pi_k P_k$ where π_k is the proportion of each interaction profile matrix within the population of cells. Interaction profiles are thought to represent subpopulations of cells with unique genome conformations that potentially correspond to cells in similar phases of the cell cycle or with similar gene expression profiles. [45]. Unfortunately, it is currently impossible to deconvolute the exact set of interaction profiles that contribute to the whole-genome contact map.

Hi-C datasets that are generated from a cellular population are inherently problematic since they represent the ensemble of interactions occurring within a population of cells resulting in a heterogeneous dataset. While single cell Hi-C methodologies exist [45], they are not commonly preformed. Hi-C datasets in polyploid organisms present further difficulties since it is extremely difficult to determine which chromosome copy is participating in a specific interaction. Researchers have developed a *Mus musculus* (mouse) model which allows them to separate chromosome pairs due to unique single nucleotide polymorphisms (SNPs) found in the individual chromosome copies [73]. Recently, the knowledge of parental SNPs and haplotype phasing data from the GM12878 *Homo sapiens* (human) cell line has been used to separate chromosome pairs in human Hi-C experiments [52]. Unfortunately, this technique is only applicable to samples where parental SNPs are

	Bin1	Bin2	Bin3	Bin4	Bin5
Bin1	-	0.06	0.75	0.06	0.02
Bin2	0.06	-	0.55	0.45	0.15
Bin3	0.75	0.55	-	0.33	0.4
Bin4	0.06	0.45	0.33	-	0.08
Bin5	0.02	0.15	0.4	0.08	-

Figure 2.6: An example of a small contact map. The symmetric lower half of the contact map is indicated in light grey. The diagonal that represents "self-self" interactions is indicated in green. The genomic bin labels are shown in dark grey along both edges of the contact map. The numbers within the cell are examples of the normalized interaction frequency count.

known and is therefore not widely applicable to all human Hi-C datasets. Additionally, the positions of similar chromosome-specific SNPs have not been identified in other organisms and the process of identifying these positions is time-consuming and costly. Therefore, it would be worthwhile to develop other computational methods to deconvolute the interactions involving single cells or chromosome copies in Hi-C datasets.

2.3 Additional Biological Datasets

It is well known that long range chromatin interactions (DNA-DNA interactions) are not the only type of interactions occurring within a genome. Additional types of nuclear interactions and information (such as transcription factor binding [5], DNA-lamin interactions [47] and genomic chemical modifications (epigenetic modifications) [5, 41] can be detected using other types of biological techniques. While these experiments do not directly measure genomic interactions, they can indicate support for certain interactions from a whole-genome contact map. Unfortunately, these additional datasets are under-utilized by the existing methods for predicting the 3D structure of the genome. Because of this, current 3D genome models are incomplete and limited in the amount of biological insight that can be extracted from them. Combining additional data

sources into the prediction and visualization of the 3D genome will allow researchers a more comprehensive look into the interplay of various genomic factors and modifications in 3D space.

As mentioned previously, it has been hypothesized that genome organization may be controlled by global mechanisms controlling gene expression [1, 13, 54]. There are many different biological techniques, such as RNA-seq [65], that can detect which genes are being expressed within a population of cells. RNA-seq can provide support to specific genomic interactions by detecting genomic regions with similar gene expression states. Similarly, epigenetic datasets can be used to infer the transcriptional activity of certain genomic regions since it is well known that epigenetic marks are correlated to gene expression [27]. There are a multitude of existing biological techniques that are able to detect different types of epigenetic marks, such as: methylated DNA immunoprecipitation with high-throughput sequencing (MeDiP-seq) [67], whole-genome bisulfite sequencing [35] and chromatin immunoprecipitation with high-throughput sequencing (ChiP-seq) [29]. Additionally, bioinformatics techniques, such as motif finding algorithms, can be used to determine which regions of the genome share common docking sites for transcriptional machinery (specifically transcription factors) [3]. If these regions are being actively transcribed they should be in close 3D spatial proximity if the transcription factory hypothesis is correct. Furthermore, it has recently been suggested that codon usage biases are strongly correlated to genome organization [17]. Because of this, the results from codon usage algorithms could be incorporated into 3D genome predictions by providing support for interactions that have common codon usage biases [30].

2.4 Computational Techniques for Predicting Genome Organization

Many computational methods have been developed to infer crude 3D genomic organizations based solely on the results produced from Hi-C experiments [4, 26, 34, 56, 61, 74]. To do this, the frequencies from a whole-genome contact map are converted into a set of corresponding Euclidean distances (the straight-line distance between two points). This is known as the 3D genome reconstruction problem [57]. In general it is thought a pair of genomic regions with a higher interaction frequency will often be closer in 3D space

than a pair of genomic regions with a lower interaction frequency. Most programs then take their predicted pairwise distances and produce a realistic visualization of the 3D genome by modelling the chromatin fibre as a polymer [58].

The existing computational methods fall under two broad categories: consensus-based methods [4, 26, 74] and ensemble-based methods [26, 34, 61]. These categories are distinguished by the number of genomic models the program generates during one run. Consensus-based methods produce a single model that best represents the whole-genome contact map. This makes them well suited to single-cell Hi-C experiments and the resultant model is relatively easy to interpret [58]. But consensus-based methods lack biological accuracy when they are used to predict the genomic organization from a population of cells. This is because their predictions typically ignore the heterogenous nature of population Hi-C datasets [33]. Alternatively, ensemble-based methods produce a large set of genomic models which represent the inherent heterogeneity of genome organizations that exists within a population of cells. Although this type of technique may result in more biologically accurate predictions, it is substantially more difficult to extract relevant and novel biological information from the set of resultant models compared to the single model produced by consensus-based methods [33].

None of the existing methods are able to provide an exact solution to the 3D genome reconstruction problem. It has been suggested that all new potential solutions to this problem should encompass the following features which are uncommon in the existing methodologies:

1. New methods should be scalable to a variety of genome sizes [2].

2. New methods should have the ability to integrate and visualize multiple types of genomic datasets in addition to whole-genome contact maps [58].

3. Programs should make it easy for users to extract 3D clusters of genomic regions in order to facilitate downstream analysis [2].

Currently, many of the existing programs do not take any supplementary biological datasets into account when predicting genomic organizations. Because of this reason, the models generated from the existing methods are crude and lack crucial biological information that is required to understand the complex relationship

between the structure and function of the genome. None of the current methods utilize a constraint logic programming approach for modelling and predicting the 3D genomic architecture. The constraint logic programming tool developed here already satisfies characteristic (1). Additionally, it will be used as a framework and extended during my subsequent Ph.D. research to incorporate characteristics (2) and (3).

2.5 Constraint Logic Programming

Constraint logic programming (CLP) is an extension of the logic programming paradigm in which programs are written in the form of clauses expressing facts and rules with variables drawn from specific algebraic domains. CLP builds on logic programming by incorporating the use of constraints to limit the state space that needs to be searched to find a valid solution [48]. Additionally, the values for variable domains can be drawn from a rich set of options. For instance, in CLP over the domain of integers (CLP(I)), variables with integer domains have operations like addition and subtraction already implemented which are not available in traditional logic programming languages like Prolog. In CLP, instantiation refers to bounding variables to a set of non-variable values to define the specific domain of the variable [37]. Unlike imperative programming, where programs are a built as a set of "steps" that the computer executes one after the other (similar to a recipe), CLP programs build a knowledge representation of the problem which can then be queried [28]. CLP allows for the direct codification of relationships and constraints relating to a specific problem by specifying a set of clauses and literals which represent the problem [37].

Due to it's declarative nature, CLP is well-suited to modelling real-world problems [28]. It has been successfully used to predict other 3D biomolecular structures [24, 48] and for modelling complex molecular functions [21, 22]. These applications have been shown to produce more biologically relevant results and take less computational time when compared to competing methods [21, 22, 48]. The main advantage of this approach is that it restricts the search space of possible solutions and ensures only models that agree with the experimental data are retained [28]. It is expected that similar advantages will be achieved by applying this approach to predicting 3D genomic structure.

Many different constraint logic programming development environments exist including the libraries of SICStus Prolog [9], ECLiPSe [53] and SWI-Prolog [68]. ECLiPSe was chosen as the CLP paradigm for

this thesis because it is open source and has many attractive features (described in Section 4.1). This thesis, utilizes the `Interval Constraint (IC)` and `Geocode Solver for Finite Domains (GFD)` libraries implemented in ECLiPSe. Briefly, the `GFD` library is an interface to the Geocode constraint programming toolkit [55] where variables are defined and constrained to a finite set of possible values [53]. The `IC` library is a library which requires the variables within the program to be defined and constrained within a finite integer range [53].

Chapter 3

Research Objectives

The main objective of this thesis was to develop a CLP based program that can predict a 3D structural model of the *Schizosaccharomyces pombe (S. pombe)* genome. Specifically our hypothesis was that a CLP knowledge representation exists that will provide a solution to the 3D genome reconstruction problem without using exponential space or time. Specifically, the overall research question was: How can the heterogeneous Hi-C datasets be leveraged to yield a biologically accurate 3D model of the genome? To answer this question following sub-questions were addressed: (1) What is a scalable representation of the 3D genome reconstruction problem in CLP? (2) Given the representation developed in (1) what is an efficient method of solving the 3D genome constraint satisfaction problem? (3) What is an effective way of visualizing the models developed in (1)? (4) Do the predicted genomic models retain documented biological features?

To answer these questions, (1) a method was developed that uses constraint logic programming to predict the 3D structure of the genome from Hi-C experiments. To do this, the interaction frequencies obtained from Hi-C were converted into a set of corresponding structural constraints in a computational, knowledge representation formalism [64]. The constraints were then used as a framework to determine plausible structures using constraint satisfaction which has been successfully used to predict other 3D biological structures [24, 48]. (2) A variety of search heuristics were evaluated to determine which one among them was the most efficient at solving the 3D genome constraint satisfaction problem when using the final knowledge representation developed in (1). (3) Predicted models were transformed into a graph and visualized using Cytoscape [59]. (4) The predicted models were verified through a literature search to ensure they were biologically accurate. Further research during a subsequent Ph.D. will extend this framework to: include additional genomic datasets and information in the prediction and visualization of the 3D genome model, predict the 3D genome organizations of higher level organisms and investigate whether the integrative models support

the hypothesis that the formation of transcription factories is driving genome organization.

Chapter 4

Data and Methodology

4.1 Selecting a Constraint Logic Programming Paradigm

Many different constraint logic programming paradigms exist including the libraries of SICStus Prolog [9], ECLiPSe [53] and SWI-Prolog [68]. ECLiPSe was chosen as the CLP paradigm for this thesis primarily because it is open source and because it has many attractive features including: an active support community, the ability to be customized and a rich feature set including multiple types of constraint solvers, search strategies and heuristics [53]. The majority of these features can be interchanged within the code and tested to determine which set among them is the most efficient for a specific knowledge representation of a particular problem.

4.2 Data Acquisition

4.2.1 Synthetic Data

Synthetic datasets were used for the initial testing and development of the various knowledge representations described below. These datasets were not designed to model all of the inherent characteristics of real Hi-C datasets. For instance, the interaction frequencies in a real Hi-C datasets follow a combination of the following distributions: negative binomial, Poisson and gaussian, whereas the interaction frequencies for the synthetic datasets follow a uniform distribution (using the numbers 1 to N where N is the size of the matrix). The primary purpose of these synthetic matrices was to give an initial estimate of how a particular CLP knowledge representation would scale with increasing sizes of N. All of the testing matrices used can be found in Appendix F.

4.2.2 Hi-C Data

One Whole-genome contact map from a set of Hi-C experiments performed using *S. pombe* was downloaded from Gene Expression Omnibus (accession number: GSE56849) [44]. This *S. pombe* whole-genome contact map contained 1258 genomic bins since a 10 kB genome resolution was obtained with this Hi-C experiment. *S. pombe* (also known as fission yeast) was selected as the initial organism for the development and testing of the CLP program since it is a well studied haploid model organism [72]. It has genomic features that are similar to higher level organisms such as genes with introns and exons, RNAi capabilities and large repetitive centromeres [72]. Additionally, unlike *Saccharomyces cerevisiae* which has sixteen chromosomes, *S. pombe* has a much smaller genome containing only three chromosomes making it an ideal size for the initial program development.

4.3 Initial N-Queens Knowledge Representation

A general model of the 3D genome can be constructed from a whole-genome contact map by selecting a subset of the interactions (based on the assumption described below) in order to maximize the sum of their corresponding frequencies. Since the initial data was generated from a Hi-C experiment on haploid *S. pombe* cells, it was assumed that each region of the genome can be actively forming a Hi-C mediated interaction with only one other genomic region at any given time. Therefore, only one frequency value for each genomic bin can be selected from a whole-genome contact map to be part of the final solution. Examples of two possible solution sets are shown in Figure 4.1B (a non-optimal solution) and Figure 4.1C (the optimal solution). There are many other possible solution sets that could be extracted from the whole-genome contact map but only one optimal solution. Since each frequency corresponds to a pair of genomic regions, a model can be reconstructed from the selected frequencies by constraining the corresponding genomic regions to be in close 3D spatial proximity with their interacting partners.

Naively, a greedy heuristic could be employed that would determine this subset of frequencies by sorting and selecting the interactions with the largest frequency values. This processes would then be repeated, rejecting any frequency that involves a region of the genome that has already been selected. But, this would

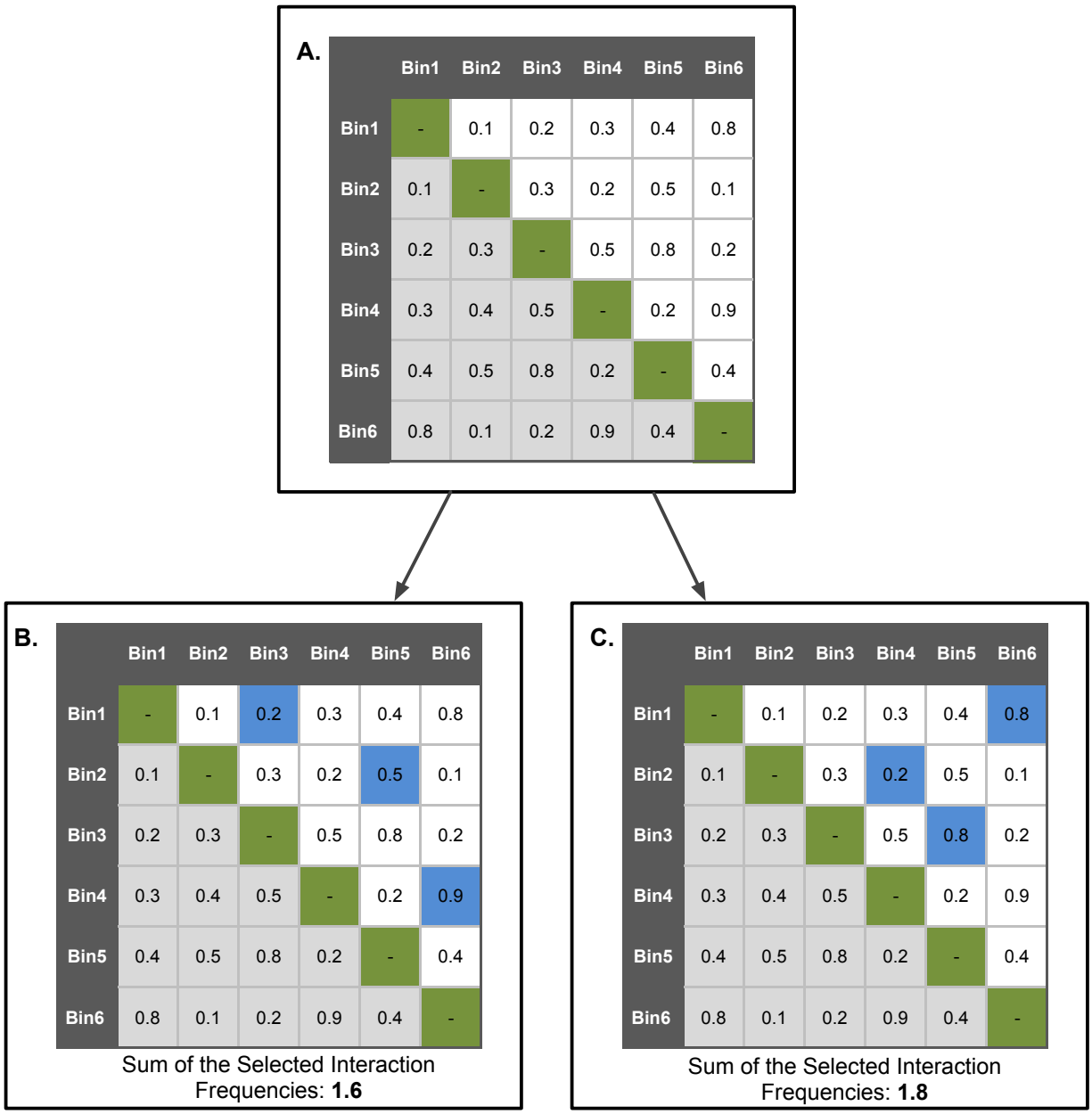

Figure 4.1: An example of two possible solutions to the 3D genome reconstruction problem in haploid cells. For all of the panels: the symmetric lower half of the contact map is indicated in light grey, the diagonal that represents "self-self" interactions is indicated in green and the genomic bin labels are represented in dark grey. For panels B and C: the blue boxes represent the subset of frequencies that could be selected as possible solutions.

fail to take into account the situation when lower frequencies, which were rejected by selecting a higher frequency interaction, would actually result in a greater maximum value for the sum of selected frequencies within the solution matrix. An example of this can be seen in Figure 4.1 where panel A is the initial whole-genome contact map and panels B and C represent two possible solution matrices with different overall frequency sums. Specifically, Figure 4.1B follows the greedy heuristic described above which results in a selected frequency sum of 1.6. The optimal solution is shown in Figure 4.1C results in a selected frequency sum of 1.8. By definition the 3D genome reconstruction problem is an optimization problem because there are multiple, possible correct solutions. However, what is desired is not just any solution, but a solution that simultaneously maximizes the sum of the frequency values from the chosen interactions. These types of problems have been shown to be well-suited for constraint logic programming.

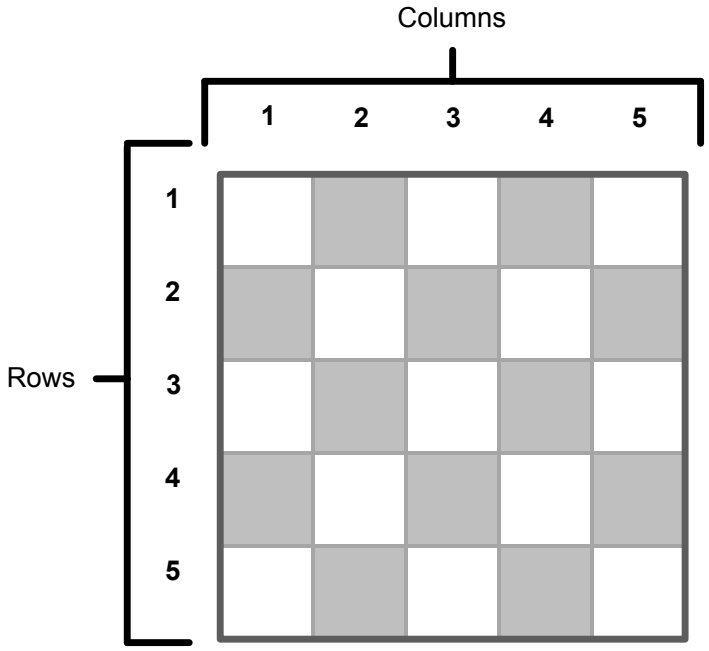

Figure 4.2: An example an empty N-Queens board. Row and column labels are found on the outer edge of the board. The white and grey squares within the board represent positions where queens could be placed.

The 3D genome reconstruction problem can be loosely modelled after a canonical problem in computer science known as the N-Queens problem. Briefly, the N-Queens problem asks how N queens can be placed on an ordinary $N \times N$ chess board so that none of the queens can attack each other within one move. An example of an empty N-Queens board is depicted in Figure 4.2. While this problem is computationally very

difficult, it is easy to represent and solve with CLP. By modelling the relationships and constraints in this problem, CLP is able to efficiently determine the optimal solution for the N-Queens problem within a few minutes, when N is less than or equal to 10,000 (data not shown). In this representation of the N-Queens problem, each square on the board is represented by a logical variable that can be bound to a value like 'Q' if the queen is located in that square in the solution set. Similarly, a knowledge representation of the 3D genome reconstruction problem can be produced where each logic variable represents a cell from the contact map that can be bound to the corresponding frequency value if the interaction is selected to be a part of the final solution set.

One of the main differences between the N-Queens problem and the 3D genome reconstruction problem is that the latter has a cost associated with each possible solution which is related to the preference for one solution over another. In this case, the cost is the sum of the selected interaction frequencies and the optimal solution will have the maximum cost. Additionally, the 3D genome reconstruction problem does not need to model the N-Queens "diagonal constraint". The removal of this constraint makes the 3D genome reconstruction problem a relaxed version of N-Queens, that could be deemed the "N-Rooks" problem. The goal of the CLP program is to determine which subset of frequencies should be selected from the whole-genome contact map in order to maximize the overall frequency summation of the solution set without violating any of the defined relationships and constraints. The specific relationships and constraints for the 3D genome reconstruction problem are defined below.

An ECLiPSe program was developed to represent and solve the 3D genome reconstruction problem with CLP(IC) (A part of the complete program for a 5×5 sub-matrix of the yeast whole-genome contact map is shown in Program A.1). The complete program can be found at `https://github.com/kimmackay/msc_research/tree/master/initial_clp`. Initially, it was developed in a way that allowed users to hard-code multiple genomes within the program source code. Each cell of the input whole-genome contact map was represented by an individual variable within the program. The domains of the variables were defined to either be zero, which represented the interaction not being selected, or the corresponding interaction frequency value from the whole-genome contact map. Since the genomic bins can be easily mapped to their corresponding chromosomes, this relationship was not explicitly represented within the program.

The relaxed N-Queens constraints were modelled through the use of the global `atleast/3` constraint from the `ic_global` library. Specifically, the program's logic stated that for every set of interaction cells which correspond to a particular genomic bin there must be at least $N - 1$ variables bound to value of zero. In order to find the optimal subset of interactions, `minimize/2` from the `branch_and_bound` library was used to maximize the sum of the selected frequencies in the solution set (this was done by minimizing the negative sum of the frequencies since no maximize function was defined in this library — Program A.1 line 80). Finally, the selected interactions were displayed output as a text representation of a numeric-valued matrix. The user can determine a solution to the 3D genome reconstruction problem by invoking the `solve/1` predicate with the appropriate input parameter (for example: `solve(1).`). While this strategy was effective and efficient (to be described in Section 5.1), it was unable to solve genome size problems due to limits on the number of logical variables that the ECLiPSe internal dictionary can store. To overcome this, the *S. pombe* whole-genome contact map had to be compressed by a factor of three in order to reduce the number of variables required to represent and solve the problem while utilizing this knowledge representation. The value of this compression was determined by trial and error and would have to be determined independently for each new genome added to this representation.

4.3.1 Automation

A variety of Perl scripts (Appendix B) were developed to assist in the generation of the ECLiPSe program based on a whole-genome contact map. The final workflow is depicted in Figure 4.3. Briefly, there are four main sections:

A) Initial Inputs: these are the inputs that the developed Perl scripts expect to receive from the user. The "normalized Hi-C interaction matrix" input can be generated by processing the raw sequence reads obtained from a Hi-C experiment or public repository such as GEO with software similar to HiCUP [69] (a pipeline used for normalizing and mapping the initial raw sequence reads) and HOMER [25] (a program used for the generation of a whole-genome contact map). The "number of genomic bins to compress" input is used to specify the number of adjacent genomic bins that need to be combined in order to reduce the number of variables in the ECLiPSe program; in the case of *S. pombe* it is three. The

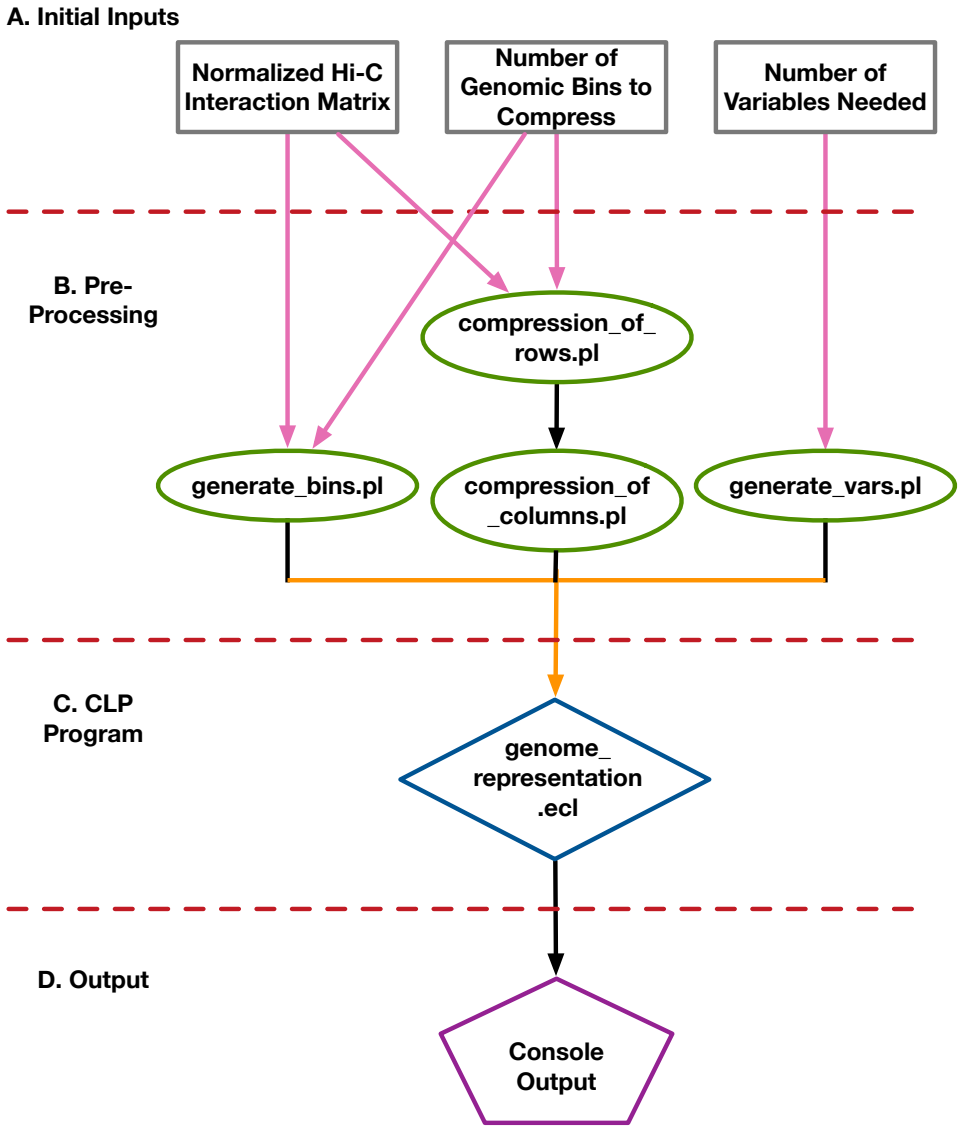

Figure 4.3: The initial workflow for producing a logical model of the three-dimensional yeast genome. Grey boxes represent required user input. Green circles represent the developed Perl programs. The blue diamond represents the developed ECLiPSe program and the purple pentagon represents the final output of the CLP program. The pink arrows represent expected user input. The black arrows show the flow of program input/output within the workflow while the orange lines represent the manual combination of the pre-processing outputs that are necessary to produce the custom ECLiPSe program. The dashed red lines separate the programs into four general subsections.

"number of variables needed" input is used to specify the size of one edge of the interaction matrix (ie. the user would enter 1258 for a 1258×1258 matrix). This value will be used to automatically generate the large number of variables needed to represent the problem.

B) Pre-Processing Programs: these are the programs that will format the data to be the correct size and syntax for use in the ECLiPSe program (Program B.1, Program B.2 and Program B.3). There is also a method (Program B.4) that will generate a matrix of N by N (where N is specified through the "number of variables needed" input) variables which are needed to represent the data. The output of these programs must be manually combined by the user in order to generate the custom ECLiPSe program.

C) The CLP Program: this is the program that determines the most probable interactions occurring in the genome based on the defined relationships and constraints.

D) Output: this section represents the output given by the CLP program. Essentially, the CLP program will print out a matrix of size $N \times N$ that shows the interaction frequencies selected by the program.

4.4 Minimal N-Queens Knowledge Representation

In an attempt to overcome the size constraint of ECLiPSe's internal dictionary, a new knowledge representation was implemented based on a minimal N-Queens solution. Essentially, this representation collapses the $N \times N$ chess board into a list of size N (Figure 4.4). In this representation the logic variables represent queens in the solution set. The positioning of these queens is encoded through the list position and the value each variable is bound to (for the row and column positions, respectively).

Ideally, the minimal N-Queens representation could be mapped to the 3D genome reconstruction problem by having a list of size N where the logical variables represent the selected interaction frequencies in the solution set. In practice, two lists of size N are required in order to completely represent and store the relevant data associated with an interaction frequency (one list for frequency values and the other for the corresponding column values — Figure 4.5). This representation reduced the required variable storage to $2N$ (as opposed to the N^2 storage requirement in the initial N-Queens representation). The position of

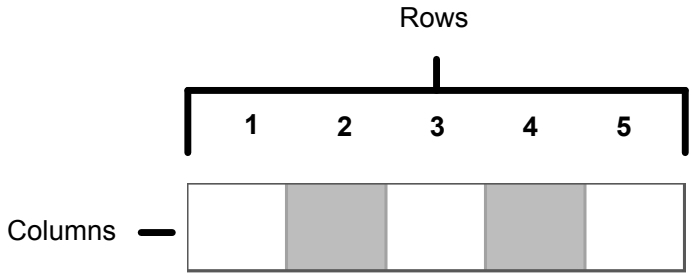

Figure 4.4: An example an empty N-Queens board using a minimal knowledge representation. Row positions for queens in the solution set are found on the top of the list. The column position of a queen in the solution will correspond to the value placed in the square.

the variable within either list was used to encode the corresponding row of the whole-genome contact map, whereas the value of the variable in the "column list" was used to encode the corresponding column of the whole-genome contact map for a particular selected frequency value. A partial implementation of this program for a 5×5 sub-matrix of the yeast whole-genome contact map is shown in Program A.2. The complete program can be found at `https://github.com/kimmackay/msc_research/tree/master/minimal_clp`.

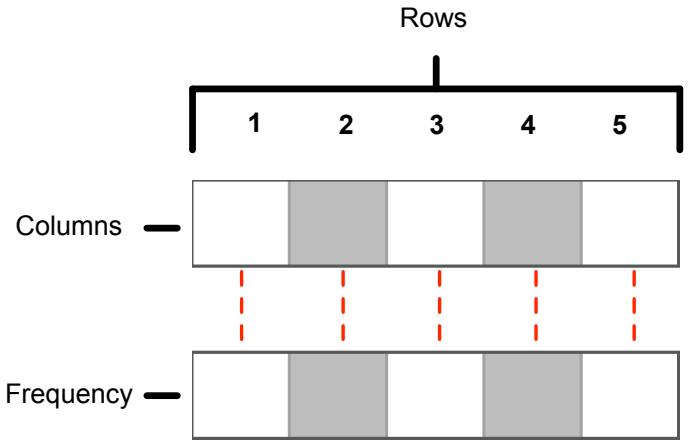

Figure 4.5: A visualization of the N-Queens based, minimal knowledge representation for the 3D genome reconstruction problem. Row positions for the selected interactions are found on the top of the list. The column position of a selected interaction will correspond to the value placed in the square of the "Columns" list. The corresponding interaction frequency for a specific row, column pair will be in the same square the column value was found in. This relationship between the squares from the two lists are indicated with a dashed red line.

The relaxed N-Queens constraints were modelled through the use of a newly defined `alldifferent_except/1` constraint (lines 7 to 33) which was developed to ensure all of the variables within the column list are pairwise different from each other, or zero. Additionally, the program uses constraint imposition (lines 64 to 98) with

values from the whole-genome contact map to encode valid column, frequency pairs. In order to find the optimal subset of interactions, `minimize/2` from the `branch_and_bound` library was used in conjunction with `search/6` from the `gfd` library (line 109). Finally, the selected interactions are displayed in the form of two numeric lists and can be extracted for downstream visualization. The user can determine a solution to the 3D genome reconstruction problem by invoking the `maximize/2` predicate with output parameters that will be bound to values that represent the selected interactions (i.e. `maximize(R, F)`. A CLP program was generated and tested with a variety of synthetic whole-genome contact maps in order to determine the scalability of this representation (to be described in Section 5.2). While this representation was able to overcome the problem discovered in the initial N-Queens knowledge representation (as will be shown in the results section) its runtime was exponential based on the the number of rows or columns (N) in the whole-genome contact map.

4.5 Improving the Program Runtime

This section describes the divide and conquer approach that was used to improve the runtime for the minimal N-Queens representation as well as the minimal, non-redundant N-Queens representation (described below in Section 4.6).

4.5.1 Divide and Conquer

Division

In order to improve the efficiency and scalability of the program, the yeast whole-genome contact map was divided into six subproblems that can be run in parallel. These subproblems represent a natural partition of the whole-genome contact map. The location of these subproblems within the whole-genome contact map are identified in Figure 4.6. Each subproblem corresponds either to interactions within a specific chromosome (intra-interactions), or to a subset of the interactions between 2 of the 3 chromosomes (inter-interactions). Since the set of genomic bins that correspond to a particular chromosome was known ahead of time, the separation of these subproblems was provided as program input to generate the six CLP programs.

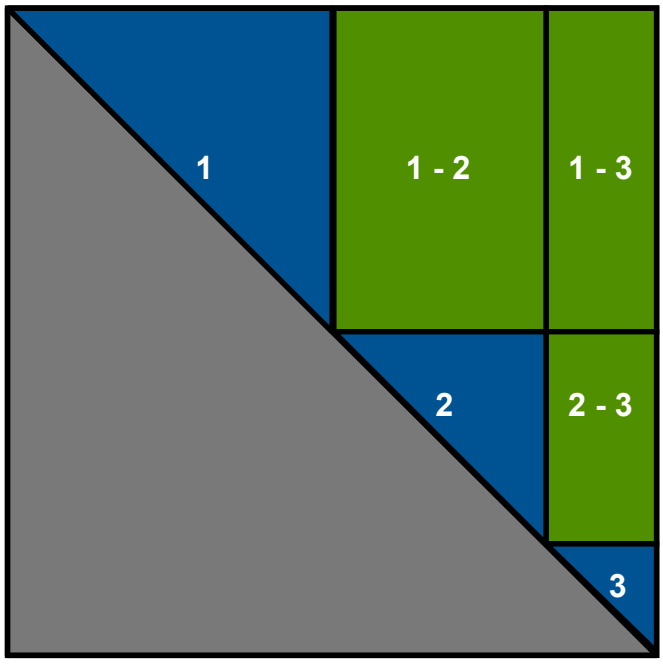

Figure 4.6: Identification of subproblems within the yeast contact map. The large grey triangle represents the portion of the contact map that does not need to be processed since all contact maps are mirrored along the diagonal. The blue triangles represent the subsections of the contact map that correspond to intra-chromosomal interactions, while the green squares represent the subsections of the contact map that correspond to the inter-chromosomal interactions. The labels on the blue and green areas represent the chromosome(s) involved in the interactions within that subsection of the contact map.

Local Conquer

In order to solve entire yeast 3D genome reconstruction problem, ECLiPSe programs that correspond to the following subproblems were run independently: (1) chromosome 1 intra-interactions, (2) chromosome 2 intra-interactions, (3) chromosome 3 intra-interactions, (4) chromosome 1 - chromosome 2 inter-interactions, (5) chromosome 1 - chromosome 3 inter-interactions and (6) chromosome 2 - chromosome 3 inter-interactions. Similarly to Section 4.4, the user can determine a solution to the 3D genome reconstruction problem by invoking the `maximize/3` predicate with parameters that correspond to the desired output file names (i.e. `maximize(‘‘RowFileName" , ‘‘FreqFileName" , ‘‘NonZeroRowsFileName"`). The complete programs for the six subproblems using the minimal N-Queens knowledge representation can be found at `https://github.com/kimmackay/msc_research/tree/master/minimal_clp/subproblems`.

Recombination/Merging

The results from each subproblem were then combined together to reconstruct the entire 3D logical model of the *S. pombe* genomic architecture. Specifically, to help account for the instances when a single genomic region is involved in more then one solution interaction a "dynamics coefficient" (abbreviated the D coefficient) was defined. The D coefficient is calculated by scanning all of the result files and counting how many interactions a specific region is making across the subproblem solution sets. The more interactions a genomic region is involved in, the higher it's corresponding D coefficient is. For instance, if genomic bin 1 was found to be forming an interaction in subproblem (1) and subproblem (2) it would have a D coefficient of 2, whereas if it was found to be only forming an interaction in subproblem (2) it would have an associated D coefficient of 1. This allows the program to encode some of the flexibility and dynamics of genome organization into the logical model. Additionally, it allows the model to keep all the information associated with each subproblem's optimal solution instead of having to exclude interactions that involve genomic regions already selected as part of the solution set.

4.6 Minimal, Non-Redundant N-Queens Knowledge Representation

In an attempt to further decrease the program's runtime, redundant frequency and column values along with zero value frequencies were removed from the variable domains resulting in the minimal, non-redundant version of the knowledge representation. This knowledge representation utilizes the same structure described in Section 4.4 . For the sake of brevity, two partial program's are shown in Program A.3 (representative of an intra-interaction CLP program) and Program A.4 (representative of an inter-interaction CLP program). The complete programs for each of the six subproblems can be found at `https://github.com/kimmackay/msc_research/tree/master/minimal_nonredunant_clp)`. The user can determine a solution to the 3D genome reconstruction problem by invoking the `maximize/3` predicate with input parameters that correspond to the desired names of the output files the selected interactions (i.e. `maximize(‘‘RowFileName" , ‘‘FreqFileName" , ‘‘NonZeroRowsFileName"`).

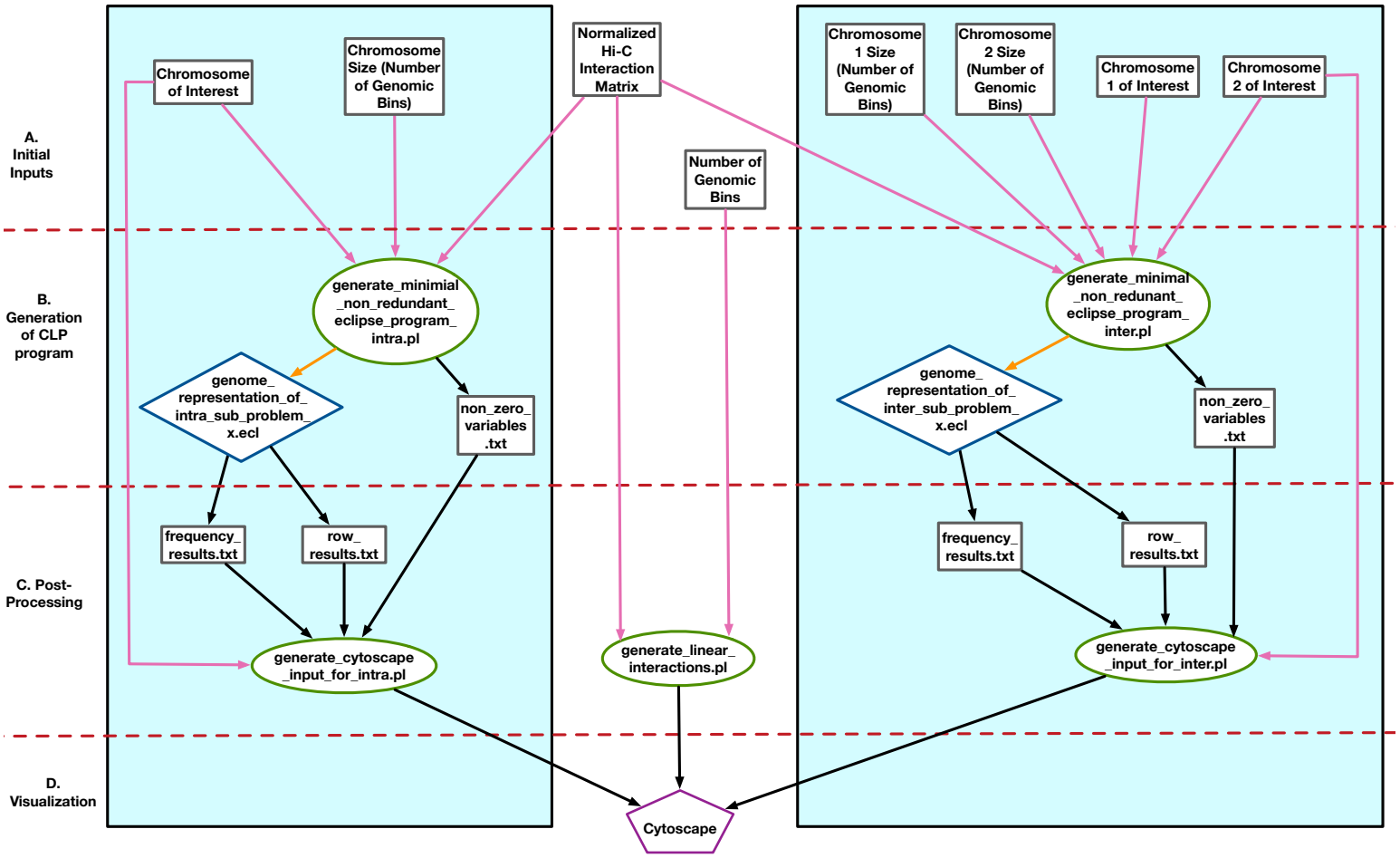

Figure 4.7: The workflow for producing a logical model of the three-dimensional yeast genome using a minimal N-Queens representation. Grey boxes represent program input and outputs. The green circles represent the developed Perl programs. Blue diamonds represent the ECLiPSe programs produced by the Perl programs and the purple pentagon represents the visualization of the predicted model with Cytoscape. The pink arrows represent expected user input. The black arrows show the flow of program input/output within the workflow while the orange arrows represent the automatic generation of the custom ECLiPSe program. The dashed red lines separate the programs into 4 general sub-sections. The light blue panels are areas of the workflow that will be repeated multiple times in order to accommodate all the intra- and inter-interaction subproblems.

A variety of Perl scripts (Appendix C) were developed to automate the generation of a minimal non-redundant ECLiPSe program based on a whole-genome contact map. The overall workflow is depicted in Figure 4.7. Briefly, there are four main sections of the workflow:

A) Initial Inputs: these are the inputs that the developed Perl programs expect to receive from the user. The "normalized Hi-C interaction matrix" input can be generated by processing the raw sequence reads obtained from a Hi-C experiment or public repository such as GEO with software similar to HiCUP [69] and HOMER [25]. Additionally, each program requires the user to specify which chromosome(s) subproblem the program should generate and the size of the corresponding chromosome(s).

B) Generation of CLP Program: the Programs C.1 and C.2 generate the custom ECLiPSe program for a specific subproblem and whole-genome contact map, as well as a text file that lists the non-zero genomic bins which will be used to generate the network.

C) Post-Processing: the programs in this step (Program D.1, Program D.2 and Program D.3) utilize the output generated by the programs in step B to generate a network file that can be used as input to Cytoscape.

D) Visualization: this step visualizes the results of the CLP program in Cytoscape.

4.7 Visualization

The results from the CLP program were converted into a network using Program D.1 or Program D.2 (Appendix D) and visualized with Cytoscape. Each edge in the network was weighted using the novel relative distance measure: the average D coefficient (defined above) between the two genomic regions divided by their corresponding interaction frequency from the whole-genome contact map. It is commonly accepted that the distance between two genomic regions in 3D space is inversely proportional to the associated interaction frequency from the whole-genome contact map. For instance when genomic region a (assuming a D coefficient of 1) is selected to be forming an interaction with genomic region b (assuming a D coefficient of 1) at a frequency of 14, the relative distance measure will be 0.071 i.e. $(\frac{(1+1)/2)}{14})$, suggesting they have a very close 3D spatial proximity. On the contrary, if genomic region c (assuming a D coefficient of 7) is selected to be

forming an interaction with genomic region d (assuming a D coefficient of 5) at a frequency of 2, the relative relative distance measure will be 4.75 i.e. $(\frac{(7+5)/2)}{2})$. An edge-weighted spring-embedded layout was then applied to the network where the edge weight was the newly defined relative distance measure. The nodes in the network were then coloured according to their chromosome number.

4.8 Evaluation and Validation

4.8.1 Computational Evaluation

Each CLP program that was generated in each knowledge representation (including the program's generated for the synthetic datasets) was compiled and run independently three times. Program runtimes were measured in the form of elapsed runtime. The program runtimes (elapsed time) from each of the three runs were averaged to produce the average program runtime for each of the programs described above as well as all of the testing and heuristic optimization instances. Programs were run on a compute server which has 20TB RAID external storage, a dual 8-core CPU (Intel Xeon E5-2670) and 384GB RAM. An example of the command line input and output is shown in Appendix E.1.

4.8.2 Computational Feature Extraction

Program features (such as the number of variables, the number of values in a variable domain and the number of constraint implications) were extracted using a bash command similar to `grep -c '<feature specific text>' <input file>`. The results from this command were then normalized based on the number of total variables used in the `<input file>` knowledge representation.

4.8.3 Biological Validation

A literature search was performed to identify a set of documented genomic features (such as 3D centromere and telomere clustering) that should be present in the predicted 3D yeast genome. Nodes in the Cytoscape visualization were then coloured to quickly and easily identify whether or not the predicted model contained these identified genomic features.

Chapter 5

Results and Discussion

5.1 Initial N-Queens Knowledge Representation

The initial CLP(IC) program was developed using the modified N-Queens knowledge representation described in Section 4.3. As mentioned previously, the whole-genome contact map was represented as a $N \times N$ matrix, where N corresponds to the number of genomic bins along one edge of the contact map. Constraints were defined using the `atleast/3` constraint from the `ic_global` library to ensure that each genomic bin could participate in only one Hi-C mediated interaction within the final solution set. A subset of the program was described in Appendix A.1 and the complete program can be found at `https://github.com/kimmackay/msc_research/tree/master/initial_clp`. Various important representation features along with the average program runtimes are listed in Table 5.1. The "Number of Frequency Values in the Variable Domains"

Table 5.1: Features of the Initial N-Queens Knowledge Representation. The top row lists the names of the individual subproblems. The "Testing" problem corresponds to the 5×5 synthetic contact map from Appendix F.1. The "Condensed Genome" problem corresponds to the representation that was developed by combining adjacent genomic bins. The "Complete Genome" problem corresponds to the representation of the entire 1258×1258 yeast whole-genome contact map. '-' indicates that there was no runtime information available.

Program Feature	Testing	Condensed Genome	Complete Genome
Number of Genomic Bins	5	420	1258
Number of Variables	25	176400	1582564
Number of Frequency Values in the Variable Domains	2	2	2
Average Program Runtime (Elapsed Time — seconds)	0.3	0.43	–

row has a constant value of '2' across all the problems since each variable domain is instantiated to a list containing the corresponding frequency value from the whole-genome contact map and zero. These values represent the corresponding interaction being either selected as a member of the solution set, or rejected from the solution set (respectively). Using this knowledge representation, ECLiPSe was able to find the optimal and correct solution for the initial testing matrix (Appendix F.1) in 0.30 seconds.

Once the initial testing and development with the synthetic data was completed, the same knowledge representation was used to generate a CLP(IC) program for the entire *S. pombe* whole-genome contact map. Unfortunately, this program would not compile because the number of variables required to represent the contact map (1,582,564) exceeded the total memory of ECLiPSe's internal dictionary. Since there was no way of manually increasing the dictionary size, this was overcome by combining adjacent genomic bins into larger regions to achieve a lower number of variables. For the yeast whole-genome contact map, every set of three adjacent bins had to be combined into one larger genomic bin in order for the program to compile. This genomic compression resulted in a smaller whole-genome contact map of 420×420, but it should be noted that this process reduced the overall resolution of the data from 10 kB to 30 kB.

While computationally efficient, this knowledge representation is not scalable to higher order organisms with larger genome sizes since the space utilization complexity is $O(N^2)$. Any attempts to use this knowledge representation for the comprehensive prediction of larger genomic architectures would result in a drastic decrease in the experimental resolution due to the size restrictions imposed by ECLiPSe's internal dictionary. However, since the average program runtimes achieved with this representation are remarkably low, this program could still be utilized for organisms with small genomes. Alternatively, it could also be used to gain a high-level overview of the genomic organization of a large genome (at a low resolution). Other prediction programs (such as the method developed by Trieu and Cheng [61]) could then be used to predict the high-resolution local genomic organization of specific regions within the genome.

5.2 Minimal N-Queens Knowledge Representation

In an attempt to maintain the experimental resolution and overcome the size restriction of ECLiPSe's internal dictionary, a new CLP(GFD) program was developed based on the minimal N-Queens knowledge

representation described in Section 4.4. Briefly, instead of representing each cell of the whole-genome contact map, two lists of size N were used which held only the frequency and column values selected to be part of the final solution set. Constraints were defined using the `alldifferent_except/1` constraint (developed as a part of this thesis) on the column list to ensure that each genomic bin was involved in only one Hi-C mediated interaction within the solution set. Specifically, the `alldifferent_except/1` constraint ensured that the value of each variable within the parameter list was pairwise different from every other variable or zero. A subset of the program is described in Appendix A.2 and the complete program can be found at `https://github.com/kimmackay/msc_research/tree/master/minimal_clp`. Unfortunately, the CLP(GFD) program that was generated for the entire yeast whole-genome contact map was not able to find a "ground" solution after a week of continuous runtime and was therefore terminated. However when the section of the program which imposes constraints to define the valid column, frequency pairs (the "constraint imposition" section) was removed and the program only had to identify the row/column pairs that did not violate the defined relationships, correct solutions were found in less then four seconds when N was less than or equal to 1500. This suggests that the current method of constraint imposition causes a substantial increase in program runtime.

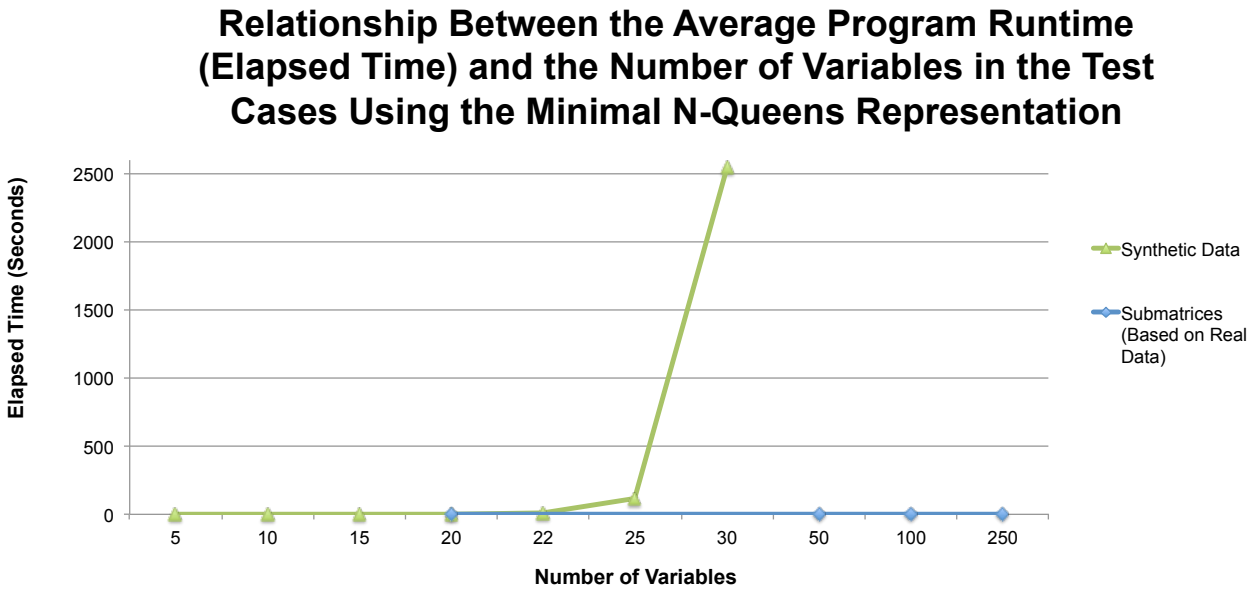

Figure 5.1: The relationship between the average program runtime (seconds) and the number of variables is shown in green for the test data and blue for sub-matrices extracted from real data.

To investigate the relationship between the number of variables (N) and the average program runtime when using this representation, the corresponding CLP(GFD) programs for the seven synthetic matrices listed in Appendix F were generated and run. In each instance, ECLiPSe was able to determine the optimal and correct solution for the corresponding subproblem. Unfortunately, as the number of variables grew, the average runtime of the corresponding program increased exponentially, suggesting that the runtime was dependent on the size of N. However, when a subset of data from the *S. pombe* whole-genome contact map was used, the corresponding average program runtime was no longer dependent on N (Figure 5.1). It is possible that this disassociation of dependency on N is seen because representations based on Hi-C data contain a handful of variables that are already ground to a zero value. This would result in a decreased overall runtime by reducing the area of the solution space which would need to be searched in order to find the optimal solution.

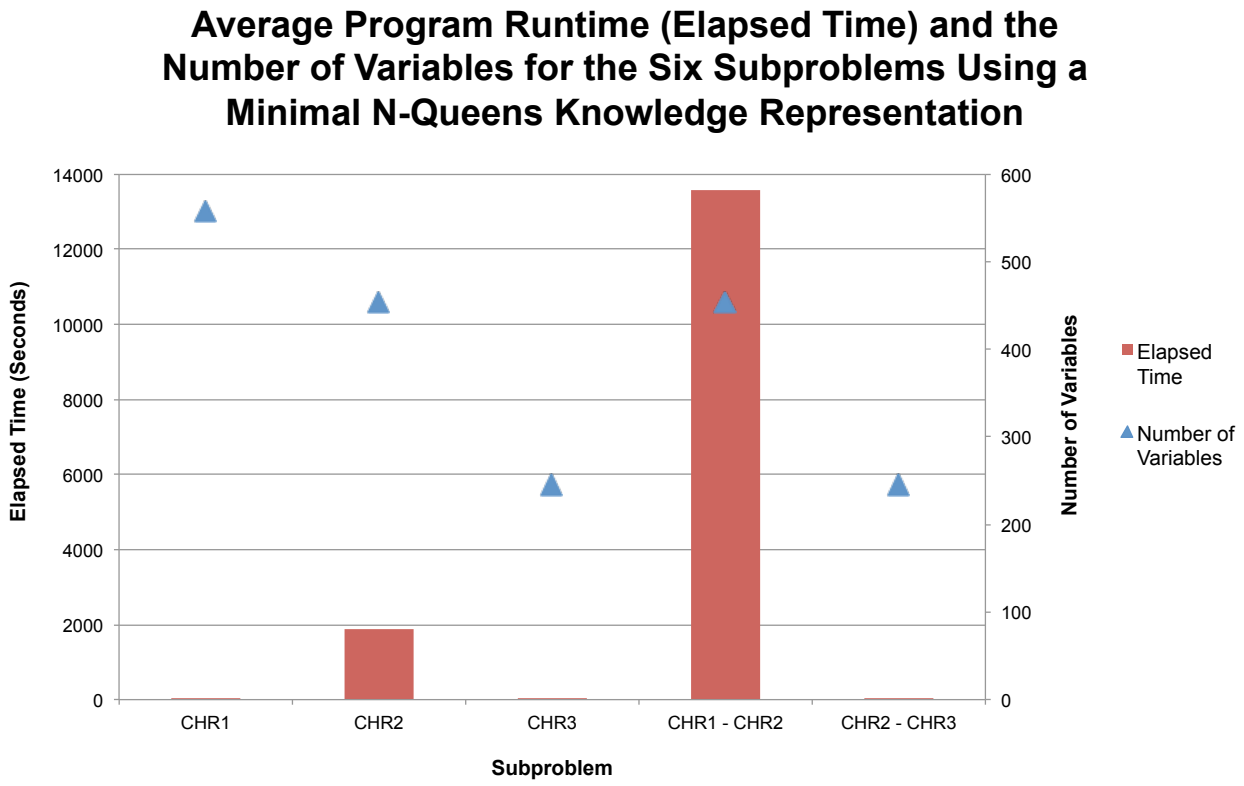

Figure 5.2: The average program runtime in seconds is shown in red and the values can be found along the left y-axis. The number of variables for each subproblem is indicated with a blue triangle and the values can be found along the right y-axis. Labels for the subproblems are listed along the x-axis.

Table 5.2: Features of the Minimal N-Queens Knowledge Representation. The top row lists the names of the individual subproblems. "CHR1" represents the chromosome 1 intra-interaction subproblem. "CHR2" represents the chromosome 2 intra-interaction subproblem. "CHR3" represents the chromosome 3 intra-interaction subproblem. "CHR1 - CHR2" represents the subproblem for chromosome 1 and chromosome 2 inter-interactions. "CHR1 - CHR3" represents the subproblem for chromosome 1 and chromosome 3 inter-interactions. "CHR2 - CHR3" represents the subproblem for chromosome 2 and chromosome 3 inter-interactions.

Program Feature	**CHR1**	**CHR2**	**CHR3**	**CHR1 - CHR2**	**CHR1 - CHR3**	**CHR2 - CHR3**
Number of Variables	558	454	246	454	246	246
Number of Zero Variables	13	15	13	347	192	167
Total Number of Frequency Values in the Variable Domains	8803	6989	3697	771	363	359
Normalized Number of Frequency Values in the Variable Domains	15.78	15.39	15.03	1.70	1.48	1.46
Total Number of Column Values in Variable Domains	156519	103739	30627	253332	137268	111,684
Normalized Number of Column Values in the Variable Domains	280.50	228.50	124.50	558	558	454
Total Number of Constraint Impositions	156519	103739	30627	253786	137514	111930
Normalized Number of Constraint Impositions(Based on the Number of Variables)	280.50	228.50	124.50	599.00	559.01	455.00
Average Program Runtime (Elapsed Time — seconds)	19.89	1889.7	6.63	13585.27	249.0	50.05

Once the initial testing and development was completed on the synthetic matricies and sub-matrices based on real Hi-C data, the same knowledge representation was used to generate the six CLP(GFD) subproblems identified in Section 4.5.1. The complete program for each subproblem can be found at `https://github.com/kimmackay/msc_research/tree/master/minimal_clp/subproblems`. The various important representation features along with the average program runtimes are listed in Table 5.2. Similarly to the testing Hi-C sub-matrices described above, Figure 5.2 demonstrates that the average program runtimes for the six subproblems are not dependent on N. However, it should be noted that the chromosome 1 - chromosome 2 inter-interaction subproblem had a drastically larger runtime when compared to the rest of the subproblems.

A number of program features were investigated to determine why the chromosome 1 - chromosome 2 inter-interaction subproblem had a significantly larger average runtime compared to the rest of the subproblems. These features included: the number of zero variables, the normalized number of values in the domain of the column variables, the normalized number of values in the domain of the frequency variables and the normalized number of constraint impositions within the program. In the case of these features, "normalization" refers to the division of the total number of instances of a specific feature by the number of variables used in the program. Specific values for each of these metrics can be found in Table 5.2. Unfortunately none of these measures were directly related to the average runtimes of the subproblem programs. But, since this representation without constraint imposition was able to correctly and efficiently solve problems for N up to 1500, the current method of constraint imposition likely was responsible for the increased runtime seen in certain subproblems. Specifically, it is possible that constraint impositions with duplicated frequency values were responsible for the dramatic increase in runtime since the program would have to spend extra time searching the solution space of potentially equivalent solutions (in terms of the summed frequency). This is demonstrated in the following code snippet from the chromosome 1 - chromosome 2 inter-interaction subproblem.

```
%% A Sub-Set of Constraint Impositions for Row559
((Chr1_Chr2_Row559 #= 0) and (Chr1_Chr2_Freq559 #= 0)) or
((Chr1_Chr2_Row559 #= 1) and (Chr1_Chr2_Freq559 #= 0)) or
((Chr1_Chr2_Row559 #= 2) and (Chr1_Chr2_Freq559 #= 0)) or
((Chr1_Chr2_Row559 #= 3) and (Chr1_Chr2_Freq559 #= 0)) or
((Chr1_Chr2_Row559 #= 4) and (Chr1_Chr2_Freq559 #= 0)) or
((Chr1_Chr2_Row559 #= 5) and (Chr1_Chr2_Freq559 #= 0)) or
((Chr1_Chr2_Row559 #= 6) and (Chr1_Chr2_Freq559 #= 0)) or
```

```
    ((Chr1_Chr2_Row559 #= 7) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 8) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 9) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 10) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 11) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 12) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 13) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 14) and (Chr1_Chr2_Freq559 #= 0)) or
    ((Chr1_Chr2_Row559 #= 15) and (Chr1_Chr2_Freq559 #= 0)) or
    ...
```

5.3 Minimal, Non-Redundant N-Queens Knowledge Representation

In order to further improve the efficiency and scalability of the CLP(GFD) subprograms, zero-valued variables and their corresponding constraint impositions were removed from the knowledge representation. Additionally, redundant values from the remaining variable domains were removed. Each CLP(GFD) program was developed based on the minimal, non-redundant N-Queens knowledge representation described in Section 4.5. As mentioned previously, this representation utilized a "divide and conquer" approach to efficiently solve the entire yeast 3D genome reconstruction problem. The overall problem was divided into 6 subproblems that represent either the intra-interactions of a specific chromosome or the inter-interactions between 2 of the 3 chromosomes. The results from each subproblem were then merged together using the method described in Section 4.5.1. A novel term, referred to as the D coefficient (Section 4.5.1), was defined to account for genomic regions that were part of the solution set for multiple subproblems. This allowed the program to encode some of the flexibility and dynamics of genome organization into the logical model. The D coefficient is similar to the B factor (also known as the temperature factor or the Debye-Waller factor) generated in protein X-Ray Crystallography experiments [32]. Briefly, the B factor encodes the degree of uncertainty associated with the computed atomic positions in 3D space [32].

Similarly to the minimal N-Queens knowledge representation described in the above section, the constraints were defined using the `alldifferent_except/1` constraint (developed as a part of thesis) to ensure that each genomic bin can only be actively forming one Hi-C mediated interaction in the final solution set. A subset of the program for an intra-interaction subproblem is described in Appendix A.3 and a subset of the program for the an inter-interaction subproblem is described in Appendix A.4. The complete set of

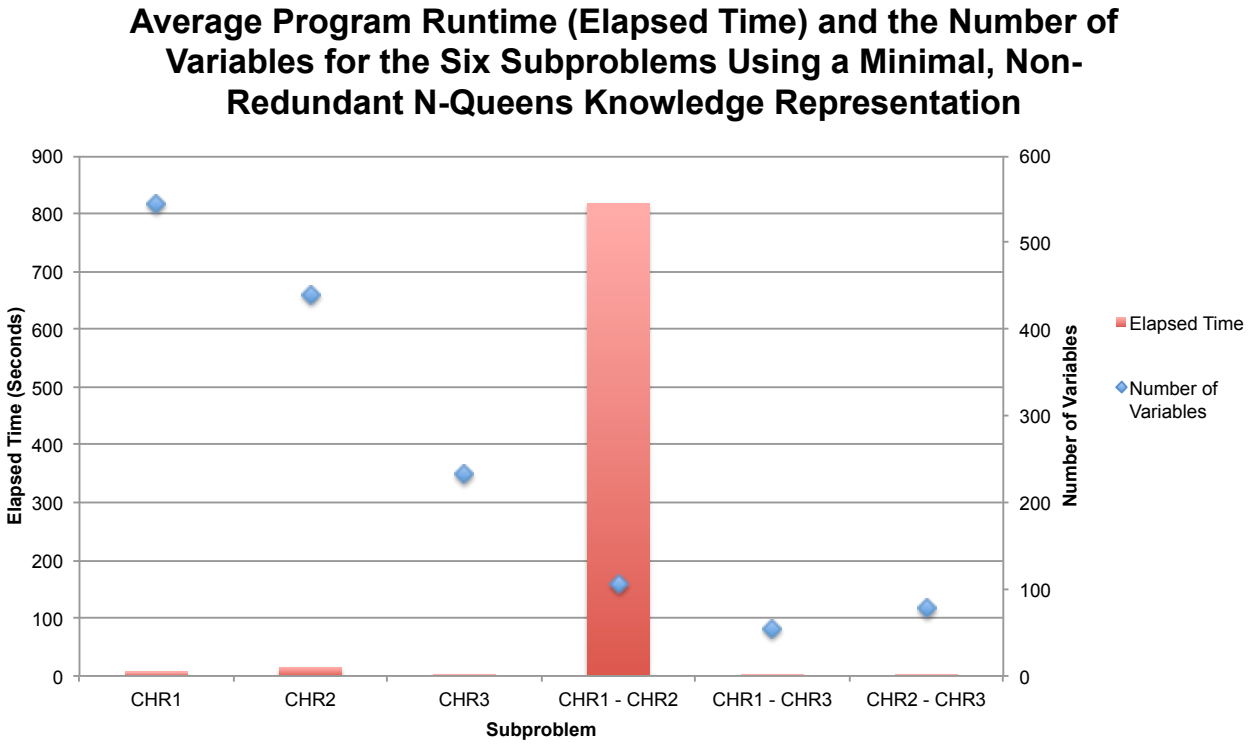

Figure 5.3: The relationship between the average program runtime and the number of variables is shown in blue. Inter-interaction subproblems are indicated with a red dot while intra-interaction subproblems are indicated with a green dot.

programs representing all of the six subproblems can be found at `https://github.com/kimmackay/msc_research/tree/master/minimal_nonredunant_clp`. The various important representation features along with the average program runtimes are listed in Table 5.3. In the case of these features, "normalization" refers to the total number of times a specific feature was found throughout the program divided by the number of variables used in the programs knowledge representation. Similarly to the minimal N-Queens representation above, the average program runtime was not dependent on N. Additionally, the removal of redundant and zero values from variable domains decreased the program runtime by an average of 84.73 % when compared to the minimal N-Queens representation in Section 5.2. Figure 5.3 depicts the relationship between average program runtime and the number of variables. The "divide and conquer" strategy applied here enables the program to be scalable to much larger genomes, since the sub-programs generated with this knowledge representation can be run in parallel.

Even though this representation was able to drastically reduce the average program runtime of the chro-

Table 5.3: Features of the Minimal, Non-Redundant Knowledge Representation. The column headings are the same as the headings described in Table 5.2.

Program Feature	CHR1	CHR2	CHR3	CHR1 - CHR2	CHR1 - CHR3	CHR2 - CHR3
Number of Non-Zero Variables	545	439	233	107	54	79
Total Number of Frequency Values in the Variable Domains	8790	6974	3684	182	171	288
Normalized Number of Frequency Values in the Variable Domains	16.13	15.89	15.81	1.70	3.17	3.65
Total Number of Column Values in Variable Domains	33090	26494	14246	856	1246	2053
Normalized Number of Column Values in the Variable Domains	60.72	60.35	61.14	8.00	23.07	26.99
Total Number of Constraint Impositions	33090	26494	14246	856	1246	2053
Normalized Number of Constraint Impositions (Based on the Number of Variables)	60.72	60.35	61.14	8.00	23.07	25.99
Average Program Runtime (Elapsed Time — seconds)	6.96	15.06	0.54	817.80	0.34	0.28

mosome 1 - chromosome 2 inter-interaction subproblem (compared to the minimal N-Queens representation described above), it still had a significantly higher runtime when compared to the other subproblems with this representation. A number of program features were investigated to determine why the chromosome 1 - chromosome 2 inter-interaction subproblem still had a significantly larger average runtime compared to the rest of the subproblems. These features included: the number of non-zero variables, the normalized number of values in the domain of the column variables, the normalized number of values in the domain of the frequency variables and the normalized number of constraint impositions within the program. Specific values for each of these metrics can be found in Table 5.3. Unfortunately none of these measures were directly related to the average program runtimes of the subproblems. It is possible that the relatively high runtime seen in the chromosome 1 - chromosome 2 subproblem is due to the issues with constraint imposition described above. Future work will focus on modifying the constraint imposition section to decrease the runtime in subproblems similar to the chromosome 1 - chromosome 2 subproblem.

5.4 Evaluation of Various Heuristics

Testing was performed to determine which library and variable selection method strategy should be used in the final knowledge representation. The `GFD` library was selected for the implementation of the knowledge representations described in Section 4.4 and 4.5 because it gave ground solutions for problems with much larger N values in the initial testing (data not shown). A subset of the available variable selection options for `search/6` from the `GFD` library were tested to determine which one produced the lowest runtime for the 3D genome reconstruction problem (Table 5.4). A program using each option was complied and run independently three times to produce an average program runtime. Specifically, the synthetic 22×22 contact map was used for this testing.

The "anti_occurence" strategy had the best performance in the subset of variable selection methods tested and was selected as the variable selection method in the 3D genome reconstruction problem. Briefly, the "anti_occurence" strategy selects the variable with the lowest number of attached propagators. Future work will be done to determine whether or not different variable selection methods could perform better in certain subproblems (such as the chromosome 1 - chromosome 2 inter-interaction problem).

Table 5.4: The Effect of Different Variable Selection Methods on Average Program Runtime (Elapsed Time) and Average Compile Time. '-' indicates that there was no runtime information available.

Variable Selection Method	Number of Variables	Average Compile Time (seconds)	Average Runtime (Elapsed Time — seconds)
input_order	22	0.72	11.39
first_fail	22	0.72	11.14
occurence	22	0.71	–
anti_first_fail	22	0.72	–
smallest	22	0.73	11.98
largest	22	0.74	–
anti_occurence	22	0.73	10.21
most_constrained	22	0.73	12.11
random	22	0.76	–

5.5 Visualization

A Cytoscape network was generated based on the output from Program D.1, Program D.2 and Program D.3. Once the network was constructed, the individual genomic bins were coloured either according to their corresponding chromosome or in a way that highlighted a particular genomic feature. An edge-weighted spring embedded layout (where the edge weight was defined as the novel distance metric developed in Subsection 4.7) was applied to the network which generated the images in Figure 5.4 and Figure 5.5B. Initial visual inspection of the generated images suggests that the novel distance metric could have some biological merit based on the distribution of the chromosomes and the overall shape of the predicted genome structure within the images.

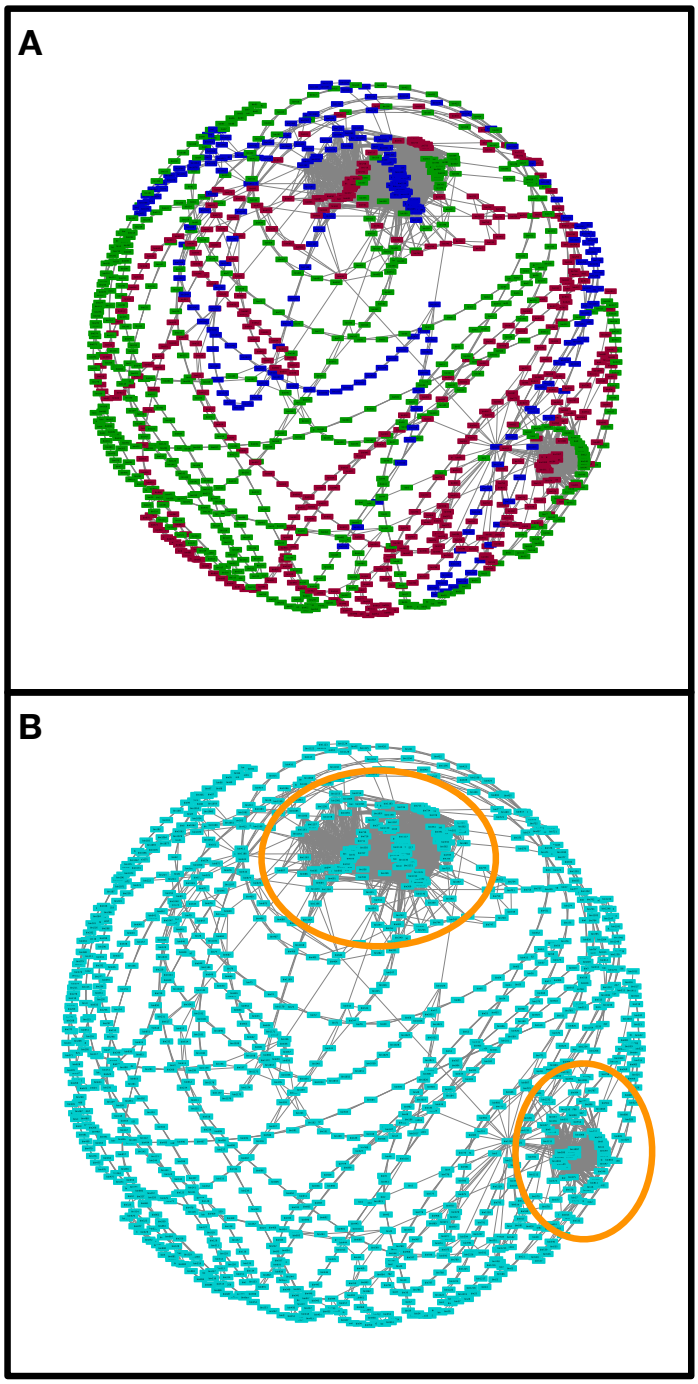

Figure 5.4: Visualization of the 3D yeast logical model using Cytoscape. In Panel A, the rectangles representing the genomic bins are coloured in the following way: chromosome 1 is green, chromosome 2 is red, chromosome 3 is blue. Grey lines indicate an interaction between a pair of genomic regions. In Panel B: the grey lines and coloured rectangles have the same meaning as in panel A. All of the genomic bins are colours in a light blue. The orange circles indicate the location of two potential clusters of transcription factories within the predicted model.

5.6 Biological Validation

One of the most well documented features of *S. pombe* genomic organization is the 3D clustering of centromeres and telomeres within the nuclear volume [10, 23, 66]. Specifically, fluorescence *in situ* hybridization (FISH) experiments on haploid *S. pombe* cells were used by Chikashige *et al.* to determine the number (two centromere clusters and one telomere cluster) and location of these clusters within the genome [10]. In order to determine whether the predicted yeast model was able to recapitulate these features, the genomic bins corresponding to centromeres and telomeres were coloured in the Cytoscape visualization. Figure 5.5 provides a visual comparison of these these clusters from the FISH experiments [10] and the predicted genomic model. The number and location of these clusters appears to be conserved in the predicted model suggesting that a reasonable amount of biological accuracy was achieved using the minimal, non-redundant subproblem based N-Queens representation described in Section 4.5.

Furthermore, the mapping and analysis of *S. pombe* Hi-C results has unveiled the presence of transcription factories within the genomic organization of these organisms [60]. It is possible that the two clusters of highly interacting regions within the logical model (Figure 5.4B — circled in orange) could represent clusters of transcription factories within the predicted genomic structure. Unfortunately, the Hi-C dataset that this model was built from does not have any associated gene expression data which would allow us to explicitly confirm this. Future work will focus on further validating this model with gene expression datasets from different studies in the same yeast strain. Additionally, genes located within the two clusters of transcription factories will be extracted and motif-finding algorithms will be used in an attempt to separate these clusters into their individual potential transcription factories.

5.7 Comparison to Existing Methods for Solving the 3D Genome Reconstruction Problem

In general, the existing methods that can be used to predict the 3D genomic structure from Hi-C data can be categorized according to the hierarchy depicted in Figure 5.6. Specifically the programs developed here can be

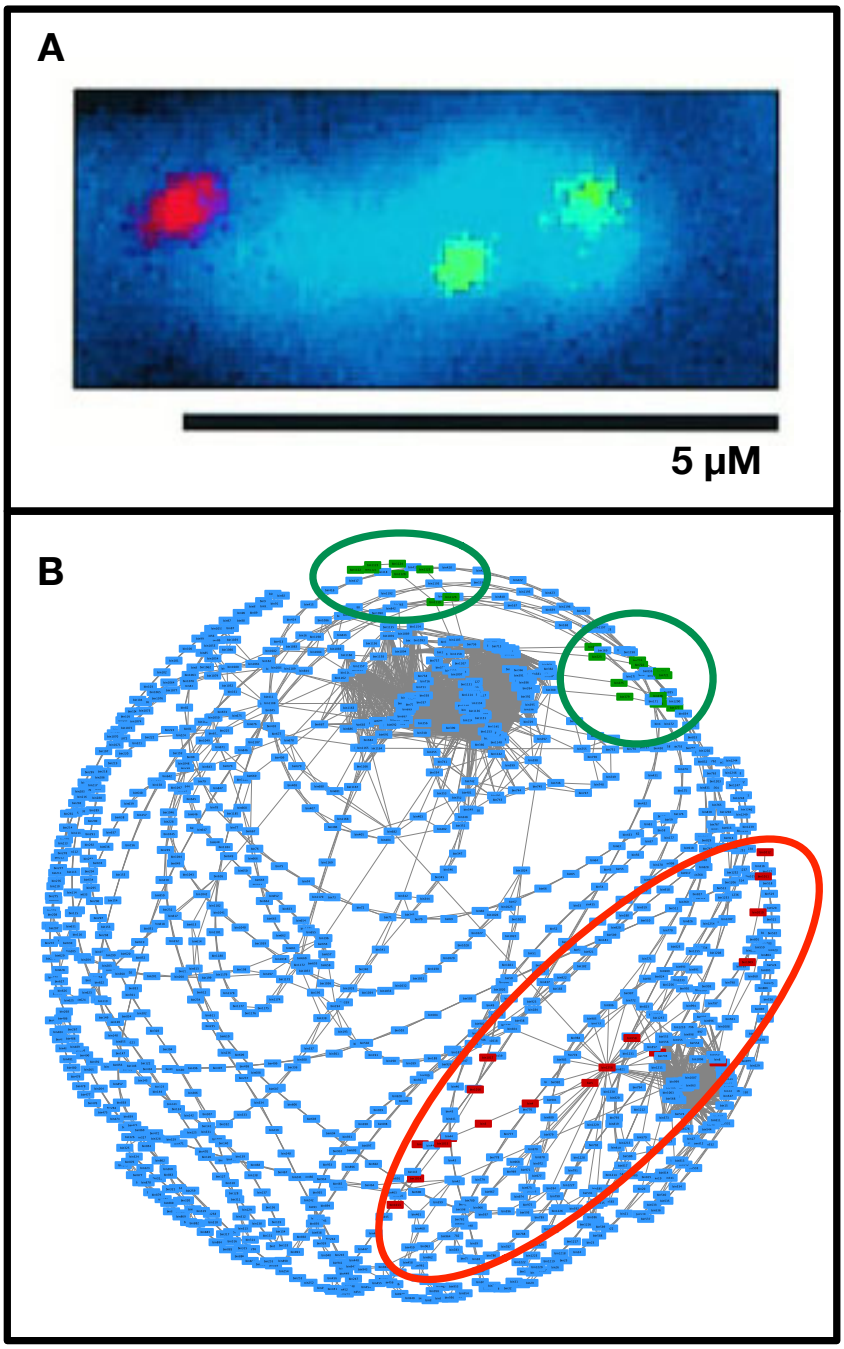

Figure 5.5: Visualization of the Nuclear Localization of Telomeres and Centromeres in *S. pombe*. In both panels: the telomeres are represented by red signals, the centromeres are represented by green signals and the nuclear DNA is represented in blue. Panel A is the results of fluorescence *in situ* hybridization performed on a haploid *S. pombe* cell [10]. Panel B is the Cytoscape visualization of the logical model with the genomic bins coloured according to the scheme above. The green circles encapsulate centromere clusters while the red circle encapsulates the telomere cluster.

classified as consensus-based, optimization solutions that utilize a complete search to find the optimal logical model of the 3D genome. Currently, none of the existing methods have be used to predict the 3D genomic organization of the *S. pombe* genome. Additionally, most of the existing methods are only able to predict the 3D structures for a portion of the genome (typically one chromosome or one region of a chromosome) whereas the programs using the minimal, non-redundant N-queens based knowledge representation developed here are able to predict a logical model for the entire *S. pombe* genome.

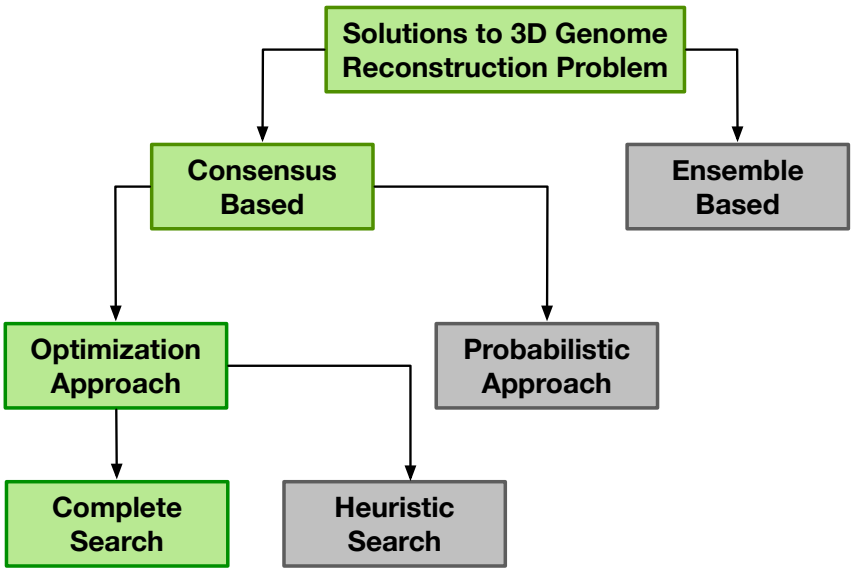

Figure 5.6: Hierarchy of existing computational methods for predicting 3D genomic structure. Green boxes represent the categories that the developed programs would fall under. Grey boxes indicate categories that do not apply to the developed programs. Black arrows indicate the hierarchal relationship between the categories.

The programs developed here are the only consensus based methods that are able to find the optimal solution to the 3D genome reconstruction problem. The method developed by Duan *et. al* is an example of a consensus based, optimization approach that utilizes a heuristic search to converge on a 3D genomic structure (Figure 5.7A) [19]. One of the main downfalls of this method is that it cannot be used in species other then yeast because it hard codes telomere and centromere 3D clustering into the prediction process. The method developed by Nagano *et. al* is an example of a consensus based, probabilistic approach that uses simulated annealing to predict the structure of the X chromosome form single-cell Hi-C data (from male mouse cells) [45]. An example of the X chromosome structure (for two of the single cells) is shown in Figure

5.7B. Finally, BACH is an example of an ensemble method that uses Markov Chain Monte Carlo sampling to predict the structure of a TAD from Hi-C data [26]. Examples of two TAD structures from the solution set are shown in Figure 5.7C.

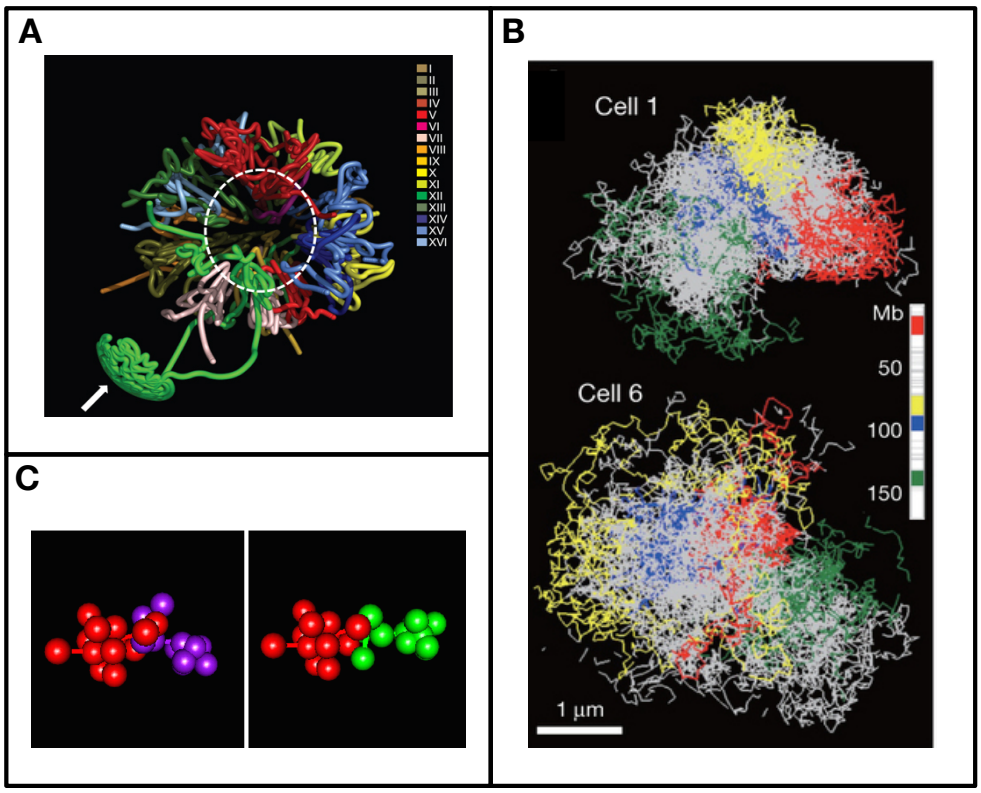

Figure 5.7: Results of existing computational methods for predicting 3D genomic structure. Panel A shows the 3D *Saccharomyces cerevisiae* genomic model produced from the method developed by Duan *et. al* [19]. Each individual chromosome is coloured differently based on the legend on the right hand side of the image. Panel B shows the 3D X chromosome models produced from the method developed by Nagano *et. al* [45]. The linear regions of the chromosome are coloured the same in each model to highlight the differences in positioning between five arbitrary regions (grey, green, blue, yellow, red). Panel B depicts the 3D visualization of two predicted TAD structures from the set of TAD structures produced by the BACH method [26].

5.8 Comparison to CLP Methods for 3D Protein Structure Prediction

Constraint logic programming has been used to predict the 3D genomic structure of other biomolecules such as proteins. Dal Pal *et al.* formulated the protein structure prediction problem as an optimization problem

in CLP where the goal was to minimize the free energy of the predicted structure [48]. A logical model of the predicted protein structure was built by using known secondary structures and the presence of disulphide bridges as constraints [48]. The logical model was then converted into a realistic "all atoms" model and a comparison of the predicted and known structure was preformed producing Figure 5.8 [48]. Overall, this method developed by Dal Pal *et al.* was proven to work satisfactorily for small proteins.

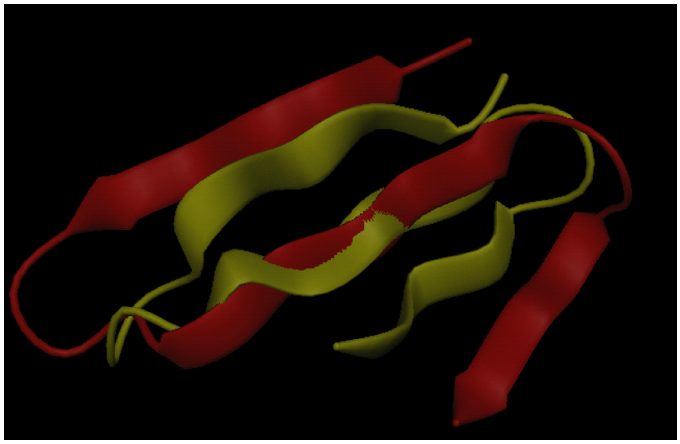

Figure 5.8: Comparison of known and predicted structures for the WW domain from Dal Pal *et al.*. The known structure is shown in yellow and the "all atom" model of the predicted structure is shown in red.

5.9 Future Work

The set of prediction programs using the minimal, non-redundant N-Queens knowledge representation developed as a part of this thesis will be utilized as a computational framework which will be extended and enhanced during my subsequent Ph.D. research. Specifically, these programs will be extended to incorporate a variety of additional genomic datasets and information into the prediction process. The unique knowledge representation utilized in this computational framework will allow these additional datasets to be naturally incorporated into the prediction of the 3D genome. For instance, each genomic bin could have an associated list of variables representing the genes found within that bin and their corresponding gene expression values. Additionally, the large body of existing microscopy images will be investigated to determine if it can be codified and incorporated into the prediction program. Biological and computational evaluation will be done

to determine if the integration of these additional datasets increases the biological accuracy of the predicted models. New methods of concurrently visualizing the 3D genome structure along with the additional genomic datasets will be investigated. It is hypothesized that the concurrent visualization of 3D genomic structures along with additional genomic datasets would allow researchers a more comprehensive look into the interplay of various genomic factors and modifications in 3D space. To ensure the developed programs are scalable to a wide variety of organisms, they will be used to predict the 3D genomic structure of higher level polyploid genomes. Machine-learning techniques will then be used in attempt to deconvolute the heterogenous Hi-C interactions occurring on separate chromosome copies in polyploid organisms. Finally, the predicted models will be evaluated to determine whether or not they support the transcription factory hypothesis of genome structure formation.

Chapter 6

Conclusion

The unique spatial organization of the genome seen under different cellular conditions is hypothesized to be a crucial mechanism driving various nuclear functions. Currently, it has been extremely difficult to comprehensively investigate this relationship due to the lack of high-resolution and high-throughput techniques for identifying the 3D genomic architecture. The recent development of the biological technique Hi-C has made it possible to detect the complete set of interactions occurring within (intra-interactions) and between (inter-interactions) chromosomes in the nucleus. While a handful of computational methods have been developed that utilize the results from Hi-C to predict a crude 3D structure of the genome, the existing methods cannot be easily extended to incorporate additional types of genomic datasets and information that might impact the overall 3D structure. Additionally, none of the existing methods utilize a constraint logic programming approach to predict the 3D genomic structure. One of the main objectives of this thesis was to determine an effective way of solving the constraint satisfaction problem of the 3D genome reconstruction problem. The developed programs were used to predict a 3D logical model of the yeast genome and the results were visualized using Cytoscape. The predicted yeast model was biologically verified through literature search to ensure the developed CLP prediction programs were able to recapitulate key features of the yeast genome. Overall, the developed computational workflow was an effective method for predicting the 3D structure of the yeast genome from Hi-C data in a reasonable amount of time. This thesis presented a way around the computational complexity problems encountered by using a minimal N-Queens knowledge representation with a "divide and conquer" approach. Additionally, the solution strategy developed here lends itself to additional speed improvements due to the potential for running the defined subproblems in parallel. Furthermore, a novel distance metric was defined (the D coefficient) which allowed a level of positional uncertainty to be encoded into the genomic model for the first time. The CLP programs developed

in this thesis will be utilized as a computational framework and extended in my subsequent Ph.D. research to allow for the incorporation of additional genomic datasets in the prediction and visualization of the 3D genome. Overall, the method developed here will be a step towards a better understanding of how the 3D structure of the genome impacts various nuclear functions.

References

[1] Ferhat Ay, Evelien M. Bunnik, Nelle Varoquaux, Sebastiaan M. Bol, Jacques Prudhomme, Jean-Philippe Vert, William Stafford Noble, and Karine G. Le Roch. Three-dimensional modeling of the *P. falciparum* genome during the erythrocytic cycle reveals a strong connection between genome architecture and gene expression. *Genome Research*, 24:974–988, March 2014.

[2] Ferhat Ay and William S. Noble. Analysis methods for studying the 3D architecture of the genome. *Genome Biology*, 16(1):1–15, September 2015.

[3] Timothy L Bailey, Mikael Boden, Fabian A Buske, Martin Frith, Charles E Grant, Luca Clementi, Jingyuan Ren, Wilfred W Li, and William S Noble. MEME SUITE: tools for motif discovery and searching. *Nucleic Acids Research*, 37:W202–W208, May 2009.

[4] Davide Baù and Marc A Marti-Renom. Genome structure determination via 3C-based data integration by the integrative modeling platform. *Methods*, 58(3):300–306, November 2012.

[5] Ronnie Blecher-Gonen, Zohar Barnett-Itzhaki, Diego Jaitin, Daniela Amann-Zalcenstein, David Lara-Astiaso, and Ido Amit. High-throughput chromatin immunoprecipitation for genome-wide mapping of *in vivo* protein-DNA interactions and epigenomic states. *Nature Protocols*, 8:539–554, February 2013.

[6] Andreas Bolzer, Gregor Kreth, Irina Solovei, Daniela Koehler, Kaan Saracoglu, Christine Fauth, Stefan Müller, Roland Eils, Christoph Cremer, Michael R Speicher, and Thomas Cremer. Three-Dimensional maps of all chromosomes in human male fibroblast nuclei and prometaphase rosettes. *PLoS Biology*, 3(5):e157, 2005.

[7] Miguel R Branco and Ana Pombo. Intermingling of chromosome territories in interphase suggests role in translocations and transcription-dependent associations. *PLoS Biology*, 4(5):e138, May 2006.

[8] Robert J Brooker, Eric P Widmaier, Linda E Graham, and Peter D Stiling. *Biology*. McGraw-Hill, first edition, 2008.

[9] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. *Programming Languages: Implementations, Logics, and Programs*, 1292:191–206, 1997.

[10] Yuji Chikashige, DaQiao Ding, Yoshiyuki Imai, Masayuki Yamamoto, Tokuko Haraguchi, and Yasushi Hiraoka. Meiotic nuclear reorganization: switching the position of centromeres and telomeres in the fission yeast schizosaccharomyces pombe. *The EMBO Journal*, 16(1):193–202, 1997.

[11] Filippo Ciabrelli and Giacomo Cavalli. Chromatin-driven behavior of topologically associating domains. *Journal of Molecular Biology*, 427(3):608–625, February 2015.

[12] Nathan F Cop and Peter Fraser. Chromosome conformation capture. *Cold Spring Harbor Protocols*, 2009(2):pdb.prot5137, 2009.

[13] Nathan F Cope, Peter Fraser, and Christopher H Eskiw. The yin and yang of chromatin spatial organization. *Genome Biology*, 11:204, 2010.

[14] T. Cremer, A. Kurz, R. Zirbel, S. Dietzel, B. Rinke, E. Schröck, MR Speicher, U. Mathieu, A. Jauch, P. Emmerich, H. Scherthan, T. Ried, C. Cremer, and P. Lichter. Role of chromosome territories in the functional compartmentalization of the cell nucleus. *Cold Spring Harbor Symposia on Quantitative Biology*, 58:777–792, 1993.

[15] Job Dekker, Marc A Marti-Renom, and Leonid A Mirny. Exploring the three-dimensional organization of genomes: interpreting chromatin interaction data. *Nature Reviews Genetics*, 14:390–403, 2013.

[16] Job Dekker, Karsten Rippe, Martijn Dekker, and Nancy Kleckner. Capturing chromosome conformation. *Science*, 295(5558):1306–1311, February 2002.

[17] Alon Diament and Ron Y. Pinterand Tamir Tuller. Three-dimensional eukaryotic genomic organization is strongly correlated with codon usage expression and function. *Nature Communications*, 5(5876), December 2014.

[18] Jesse R Dixon, Siddarth Selvaraj, Feng Yue, Audrey Kim, Yan Li, Yin Shen, Ming Hu, Jun S Liu, and Bing Ren. Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature*, 485(7398):376–380, 2012.

[19] Zhijun Duan, Mirela Andronescu, Kevin Schutz, Sean McIlwain, Yoo Jung Kim, Choli Lee, Jay Shendure, Stanley Fields, C. Anthony Blau, and William S. Noble. A three-dimensional model of the yeast genome. *Nature*, 465:363–367, 2010.

[20] Christopher H Eskiw, Nathan F Cope, Ieuan Clay, Stefan Schoenfelder, Takashi Nagano, and Peter Fraser. Transcription factories and nuclear organization of the genome. *Cold Spring Harbor Symposia on Quantitative Biology*, 75:501–506, April 2010.

[21] Damien Eveillard, Delphone Ropers, Hidde de Jong, Christiane Branlant, and Alexander Bockmayr. A multi-scale constraint programming model of alternative splicing regulation. *Computational Systems Biology*, 325(1):3–24, 2004.

[22] Eric Fanchon, Fabien Corblin, Laurent Trilling, Bastien Hermant, and Danielle Gulino. Modeling the molecular network controlling adhesion between human endothelial cells: Inference and simulation using constraint logic programming. In Vincent Danos and Vincent Schachter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin Heidelberg, 2005.

[23] Hironori Funabiki, Iain Hagan, Satoru Uzawa, and Mitsuhiro Yanagida. Cell cycle-dependent specific positioning and clustering of centromeres and telomeres in fission yeast. *Journal of Cell Biology*, 121:961–976, June 1993.

[24] Juan Antonio Garcia-Martin, Ivan Dotu, Javier Fernandez-Chamorro, Gloria Lozano, Jorge Ramajo, Encarnacion Martinez-Salas, and Peter Clote. RNAiFold2T: Constraint programming design of thermo-IRES switches. *Bioinformatics*, 32(12):i360–i368, June 2016.

[25] Sven Heinz, Christopher Benner, Nathanael Spann, Eric Bertolino, Yin C Lin, Peter Laslo, Jason X Cheng, Cornelis Murre, Harinder Singh, and Christopher K Glass. Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities. *Molecular Cell*, 38(4):576–589, May 2010.

[26] Ming Hu, Ke Deng, Zhaohui Qin, Jesse Dixon, Siddarth Selvaraj, Jennifer Fang, Bing Ren, and Jun S. Liu. Bayesian inference of spatial organizations of chromosomes. *PLOS Computational Biology*, 9(1):e1002893, January 2013.

[27] Rudolf Jaenisch and Adrian Bird. Epigenetic regulation of gene expression: how the genome integrates intrinsic and environmental signals. *Nature Genetics*, 33:245–254, 2003.

[28] Joxan Jaffar and Michael J Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19(20):503–581, 1994.

[29] David S Johnson, Ali Mortazavi, Richard M Myers, and Barbara Wold. Genome-wide mapping of in vivo protein-DNA interactions. *Science*, 316(5830):1497–1502, 2007.

[30] Morten Kloster and Chao Tang. SCUMBLE: a method for systematic and accurate detection of codon usage bias by maximum likelihood estimation. *Nucleic Acids Research*, 36(11):3819–3827, June 2008.

[31] Masahiko Kuroda, Hideyuki Tanabe, Keiichi Yoshida, Kosuke Oikawa, Akira Saito, Tomoharu Kiyuna, Hiroshi Mizusawa, and Kiyoshi Mukai. Alteration of chromosome positioning during adipocyte differentiation. *Journal of Cell Science*, 117:5897–5903, November 2004.

[32] Antonija Kuzmanic, Navraj S. Pannu, and Bojan Zagrovic. X-ray refinement significantly underestimates the level of microscopic heterogeneity in biomolecular crystals. *Nature Communications*, 5(3220), 2014.

[33] Bryan R Lajoie, Job Dekker, and Noam Kaplan. The hitchhiker's guide to Hi-C analysis: Practical guidelines. *Methods*, 72:65–75, January 2015.

[34] Annick Lesne, Julien Riposo, Paul Roger, Axel Cournac, and Julien Mozziconacci. 3D genome reconstruction from chromosomal contacts. *Nature Methods*, 11:1141–1143, March 2014.

[35] Yuanyuan Li and Trygve O Tollefsbol. DNA methylation detection: Bisulfite genomic sequencing analysis. *Methods in Molecular Biology*, 791:11–21, 2011.

[36] Erez Lieberman-Aiden, Nynke L van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R Lajoie, Peter J Sabo, Michael O Dorschner, Richard Sandstrom, Bradley Bernstein, M A Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A Mirny, Eric S Lander, and Job Dekker. Comprehensive mapping of long range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, October 2009.

[37] Kim Marriott and Peter J Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[38] Ishita S Mehta, Manelle Amira, Amanda J Harvey, and Joanna M Bridger. Rapid chromosome territory relocation by nuclear motor activity in response to serum removal in primary human fibroblasts. *Genome Biology*, 11(1):R5, January 2010.

[39] Ishita S Mehta, Christopher H Eskiw, Halime D Arican, Ian R Kill, and Joanna M Bridger. Farnesyltransferase inhibitor treatment restores chromosome territory positions and active chromosome dynamics in Hutchinson-Gilford progeria syndrome cells. *Genome Biology*, 12(8):R74, August 2011.

[40] Ishita S Mehta, Mugdha Kulashreshtha, Sandeep Chakraborty, Ullas Kolthur-Seetharam, and Basuthkar J Rao. Chromosome territories reposition during DNA damage-repair response. *Genome Biology*, 14(12):R135, December 2013.

[41] Alexander Meissner, Tarjei S. Mikkelsen, Hongcang Gu, Marius Wernig, Jacob Hanna, Andrey Sivachenko, Xiaolan Zhang, Bradley E Bernstein, Chad Nusbaum, David B Jaffe, Andreas Gnirke, Rudolf Jaenisch, and Eric S Lander. Genome-scale DNA methylation maps of pluripotent and differentiated cells. *Nature*, 454:766–770, August 2008.

[42] Tom Misteli. Beyond the sequence: Cellular organization of genome function. *Cell*, 128(4):787–800, February 2007.

[43] Tom Misteli. Chromosome territories: The arrangement of chromosomes in the nucleus. *Nature Education*, 1(1):167, 2008.

[44] Takeshi Mizuguchi, Geoffrey Fudenberg, Sameet Mehta, Jon-Matthew Belton, Nitika Taneja, Hernan Diego Folco, Peter FitzGerald, Job Dekker, Leonid Mirny, Jemima Barrowman, and Shiv IS Grewal. Cohesin-dependent globules and heterochromatin shape 3D genome architecture in S. pombe. *Nature*, 516(7531):432–435, December 2014.

[45] Takashi Nagano, Yaniv Lubling, Tim J Stevens, Stefan Schoenfelder, Eitan Yaffe, Wendy Dean, Ernest D Laue, Amos Tanay, and Peter Fraser. Single-cell Hi-C reveals cell-to-cell variability in chromosome structure. *Nature*, 502:59–64, October 2013.

[46] David L Nelson and Micheal M Cox. *Lehninger Principles of Biochemistry*. W.H. Freeman and Company, fifth edition, 2008.

[47] Valerio Orlando. Mapping chromosomal proteins in vivo by formaldehyde-crosslinked-chromatin immunoprecipitation. *Trends in Biochemical Sciences*, 25(3):99–104, March 2000.

[48] Alessandro Dal Palù, Agostino Dovier, and Federico Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5(186), December 2004.

[49] Argyris Papantonis and Peter R Cook. Transcription factories: genome organization and gene regulation. *Chemical Reviews*, 113(11):8683–8705, April 2013.

[50] Luis A Parada, Philip G McQueen, and Tom Misteli. Tissue-specific spatial organization of genomes. *Genome Biology*, 5:R44, June 2004.

[51] Ana Pombo, Paula Cuello, Wouter Schul, JongBok Yoon, Robert G Roeder, Peter R Cook, and Shona Murphy. Regional and temporal specialization in the nucleus: a transcriptionallyactive nuclear domain rich in PTF, Oct1 and PIKA antigens associates with specific chromosomes early in the cell cycle. *The EMBO Journal*, 17(6):1768–1778, 1998.

[52] Suhas SP Rao, Miriam H Huntley, Neva C Durand, Elena K Stamenova, Ivan D Bochkov, James T Robinson, Adrian L Sanborn, Ido Machol, Arina D Omer, Eric S Lander, and Erez Lieberman Aiden. 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1680, 2014.

[53] Joachim Schimpf and Kish Shen. ECLiPSe - from LP to CLP. *Theory and Practice of Logic Programming*, 12:127–156, 2011.

[54] Stefan Schoenfelder, Tom Sexton, Lyubomira Chakalova, Nathan F Cope, Alice Horton, Simon Andrews, Sreenivasulu Kurukuti, Jennifer A Mitchell, David Umlauf, Daniela S Dimitrova, Christopher H Eskiw, Yanquan Luo, Chia-Lin Wei, Yijun Ruan, James J Bieker, and Peter Fraser. Preferential associations between co-regulated genes reveal a transcriptional interactome in erythroid cells. *Nature Genetics*, 42(1):53–61, January 2010.

[55] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. *Modeling and Programming with Gecode*, 2015.

[56] Eran Segal, Michael Shapira, Aviv Regev, Dana Pe'er, David Botstein, Daphne Koller1, and Nir Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature Genetics*, 34:166–176, May 2003.

[57] Mark R Segal and Henrik L Bengtsson. Reconstruction of 3D genome architecture via a two-stage algorithm. *BMC Bioinformatics*, 16(373), November 2015.

[58] François Serra, Marco Di Stefano andYannick G. Spill, Yasmina Cuartero, Michael Goodstadt, Davide Baù, and Marc A. Marti-Renom. Restraint-based three-dimensional modeling of genomes and genomic domains. *FEBS Letters*, 589(20):2987–2995, May 2015.

[59] Michael E Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, and Trey Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, February 2011.

[60] Hideki Tanizawa, Osamu Iwasaki, Atsunari Tanaka, Joseph R Capizzi, Priyankara Wickramasinghe, Mihee Lee, Zhiyan Fu, and Ken ichi Noma. Mapping of long-range associations throughout the fission yeast genome reveals global genome organization linked to transcriptional regulation. *Nucleic Acids Research*, 38(22):8164–8177, October 2010.

[61] Tuan Trieu and Jianlin Cheng. Large-scale reconstruction of 3D structures of human chromosomes from chromosomal contact data. *Nucleic Acids Research*, 42(7):e52, January 2014.

[62] A.E. Visser, R. Eils, A. Jauch, G. Little, P.J.M. Bakker, T. Cremer, and J.A. Aten. Spatial distributions of early and late replicating chromatin in interphase chromosome territories. *Experimental Cell Research*, 243(2):398–407, September 1998.

[63] Susanne Voelter-Mahlknecht, Stephan Letzel, and Ulrich Mahlknecht. Fluorescence in situ hybridization and chromosomal organization of the human sirtuin 7 gene. *International Journal of Oncology*, 28(4):899–908, 2006.

[64] Mark Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996.

[65] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, January 2009.

[66] Quan wen Jin, Edgar Trelles-Sticken, Harry Scherthan, and Josef Loidl. Yeast nuclei display prominent centromere clustering that is reduced in nondividing cells and in meiotic prophase. *Journal of Cell Biology*, 141(1):21–29, April 1998.

[67] Yu-I Weng, Tim H-M Huang, and Pearlly S Yan. Methylated DNA immunoprecipitation and microarray-based analysis: detection of DNA methylation in breast cancer cell lines. *Molecular Endocrinology*, 590:165–176, 2009.

[68] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjorn Lager. SWI-prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

[69] Steven Wingett, Philip Ewels, Mayra Furlan-Magaril, Takashi Nagano, Stefan Schoenfelder, Peter Fraser, and Simon Andrews. HiCUP: pipeline for mapping and processing Hi-C data. *F1000Research*, 4:1310, 2015.

[70] Christopher L Woodcock and Rajarshi P Ghosh. Chromatin higher-order structure and dynamics. *Cold Spring Harbor Perspectives in Biology*, 2(5):a000596, May 2010.

[71] Eitan Yaffe and Amos Tanay. Probabilistic modeling of Hi-C contact maps eliminates systematic biases to characterize global chromosomal architecture. *Nature Genetics*, 43:1059–1065, 2011.

[72] Mitsuhiro Yanagida. The model unicellular eukaryote, Schizosaccharomyces pombe. *Genome Biology*, 3(3):comment2003.1 – comment2003.4, February 2002.

[73] Chih yu Chen, Quaid Morris, and Jennifer A Mitchell. Enhancer identification in mouse embryonic stem cells using integrative modeling of chromatin and genomic features. *BMC Genomics*, 13:152, 2012.

[74] ZhiZhuo Zhang, Guoliang Li, Kim-Chuan Toh, and Wing-Kin Sung. 3D chromosome modeling with semi-definite programming and Hi-C data. *Journal of Computational Biology*, 20(11):831–846, November 2013.

Appendix A

CLP Knowledge Representations of the Three-Dimensional Genome Reconstruction Problem

A.1 Overview of the Initial CLP Knowledge Representation

Program A.1 A Subset of the Initial CLP Program with a Complete N-Queens Knowledge Representation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load the relevant libraries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- lib(ic).
:- lib(branch_and_bound).
:- import atleast/3 from ic_global.
:- import sumlist/2 from ic_global.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The clause that should be called within
%% eclipse to determine the interactions that
%% are most likely occurring
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
solve(GenomeName) :-
    %% load the frequency values for a specific
    %% genome
    genome(GenomeName, Freqs),

    %% select the interactions that maximize the
    %% sum of the selected frequencies
    maximize(Freqs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% A clause that will select the maximum sub-
%% set of frequencies which satisfy the constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maximize(Freqs) :-
    %% get the size of the frequency matrix
    dim(Freqs, [N,N]),

    %% define the variables to represent a
    %% complete N-Queens board (NxN)
    %% generated by the perl programs
    %% note: these are 2D "arrays"
    Rows = [
        [V1_1, V1_2, V1_3, V1_4, V1_5],
        [V2_1, V2_2, V2_3, V2_4, V2_5],
        [V3_1, V3_2, V3_3, V3_4, V3_5],
        [V4_1, V4_2, V4_3, V4_4, V4_5],
        [V5_1, V5_2, V5_3, V5_4, V5_5]
    ],

    Cols = [
        [V1_1, V2_1, V3_1, V4_1, V5_1],
        [V1_2, V2_2, V3_2, V4_2, V5_2],
        [V1_3, V2_3, V3_3, V4_3, V5_3],
        [V1_4, V2_4, V3_4, V4_4, V5_4],
        [V1_5, V2_5, V3_5, V4_5, V5_5]
    ],

    %% a list of all the solution variables
    SlnVars = [
        V1_1, V1_2, V1_3, V1_4, V1_5,
        V2_1, V2_2, V2_3, V2_4, V2_5,
        V3_1, V3_2, V3_3, V3_4, V3_5,
        V4_1, V4_2, V4_3, V4_4, V4_5,
        V5_1, V5_2, V5_3, V5_4, V5_5
    ],

    %% define the variable domains, based on
    %% the frequency matrix
```

```
    ( for(I,1,N), param(Freqs,N,Rows) do
       ( for(J,1,N), param(Freqs,I,Rows) do
          Rows[I,J] :: [0, Freqs[I,J]]
       )
    ),

    %% apply constraints
    %% in this case, for each row and column
    %% only 1 variable can have a non-zero
    %% frequency value
    Num is N-1,
    ( for(I,1,N), param(Rows,Cols,N,Num) do
       atleast(Num, Rows[I, 1..N], 0),
       atleast(Num, Cols[I,1..N], 0)
    ),

    %% search the solution space for the optimal
    %% solution
    Cost is -sum(SlnVars),
    minimize(search(SlnVars, 0, smallest, indomain, complete, []), Cost),

    %% display the results
    printInteractions(Rows).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% prints all the selected interactions
%% based on the input variable Rows
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
printInteractions(Rows) :-
    %% get the dimensions of the frequency matrix
    dim(Rows, [N,N]),

    %% get the labels for the S. pombe bins
    spombeBins(BinNames),

    %% print out the value of each row and column
    ( for(I,1,N), param(Rows,BinNames,N) do
       ( for(J,1,N), param(Rows,BinNames,I) do
          X is BinNames[I],
          Y is BinNames[J],
          Z is Rows[I,J],

          %% if the value is zero: write an empty
          %% string
          ( Z #= 0 -> write("");
             %% else display the interaction
             printf("␣%s␣interacts␣%s\n", [X, Y]) )
       ), nl
    ), nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DATASET SPECIFIC INFORMATION - generated by
%% the perl programs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% bin labels for: S. pombe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
spombeBins([bin1000000, bin1000001, bin1000002,
          bin1000003, bin1000004, bin1000005]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% representation of a subset of the
%% whole-genome contact map for the S. pombe
%% 999a genome (GSM1379427) only the upper
%% triangle is explicitly defined since the
%% matrix is symmetrical along the diagonal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
genome(1, [
    [4748, 4100, 3383, 2104, 1979],
    [_, 5872, 4293, 2330, 2039],
    [_, _, 5378, 2352, 2275],
    [_, _, _, 3202, 2814],
    [_, _, _, _, 4121],
].
```

A.2 Overview of the Minimal CLP Knowledge Representation

Program A.2 A Subset of the Initial CLP Program with a Minimal N-Queens Knowledge Representation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load the relevant libraries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- lib(gfd).
:- import minimize/2 from branch_and_bound.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Based on the alldifferent/1 global constraint -
%%    ensures each variable is pairwise different
%% from each other or has a value of zero
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alldifferent_except(Vars) :-
   % get the length of the input list
   length(Vars, N),

   % loop through all pairs of elements to check
   % if they are different
   ( for(I,1,N), param(Vars, N) do
      ( for(J, I, N), param(Vars, I) do

         element(I, Vars, X),
         element(J, Vars, Y),

         % if it isn't the same element, or an element
         % with a 0 value
         (I #\= J) ->
            % constrain to be pairwise different or have
            % one (or both) equal zero
            X #\= Y or (X #= 0 or Y #= 0)
         ;
            true
         )
      ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The clause that the user should invoke
%% to get the program to run. This clause will
%% determine the interactions that
%% are most likely occurring by selecting the maximum
%% subset of frequencies that satisfy the constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maximize(Rows, Freqs) :-
   %% define N variables to represent a
   %% complete N-Queens board (NxN)
   Rows = [Row1, Row2, Row3, Row4, Row5],

   Freqs = [Freq1, Freq2, Freq3, Freq4, Freq5],

   %% define the row domains
   Row1 :: [0..1],
   Row2 :: [0..2],
   Row3 :: [0..3],
   Row4 :: [0..4],
   Row5 :: [0..5],

   %% define the variable domains, based on
   %% the frequency matrix
   Freq1 :: [0],
   Freq2 :: [0],
   Freq3 :: [0, 47],
   Freq4 :: [0, 41, 58],
   Freq5 :: [0, 33, 42, 53],

   %% apply constraints to the data to ensure
   %% they are valid 2-tuple pairs based on the
   %% biological data. note: (0,0) is when we
   %% choose to not select anything for that
   %% column row pair

   % Column1 Constraints
   ((Row1 #= 0) and (Freq1 #= 0)) or
   ((Row1 #= 1) and (Freq1 #= 0)),

   % Column2 Constraints
   ((Row2 #= 0) and (Freq2 #= 0)) or
```

```
    ((Row2 #= 1) and (Freq2 #= 0)) or
    ((Row2 #= 2) and (Freq2 #= 0)),

    % Column3 Constraints
    ((Row3 #= 0) and (Freq3 #= 0)) or
    ((Row3 #= 1) and (Freq3 #= 0)) or
    ((Row3 #= 2) and (Freq3 #= 47)) or
    ((Row3 #= 3) and (Freq3 #= 0)),

    % Column4 Constraints
    ((Row4 #= 0) and (Freq4 #= 0)) or
    ((Row4 #= 1) and (Freq4 #= 47)) or
    ((Row4 #= 2) and (Freq4 #= 58)) or
    ((Row4 #= 3) and (Freq4 #= 0)) or
    ((Row4 #= 4) and (Freq4 #= 0)),

    % Column5 Constraints
    ((Row5 #= 0) and (Freq5 #= 0)) or
    ((Row5 #= 1) and (Freq5 #= 33)) or
    ((Row5 #= 2) and (Freq5 #= 42)) or
    ((Row5 #= 3) and (Freq5 #= 0)) or
    ((Row5 #= 4) and (Freq5 #= 0)) or
    ((Row5 #= 5) and (Freq5 #= 53)),

    %% additional constraints
    %% all row domains should be different (or zero)
    %% since we only want 1 region to make 1 interaction
    %% with another region
    alldifferent_except(Rows),

    %% search the solution space for the optimal
    %% solution
    Cost #= -sum(Freqs),
    minimize(search(Freqs, 0, input_order, indomain_max, complete, []), Cost).
```

A.3 Overview of the CLP knowledge Representation for a Intra-Interaction Subproblem

Program A.3 A Subset of the Refined CLP Program with a Minimal N-Queens Knowledge Representation for Detecting Intra-Interactions for a Subset of Chromosome 1.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load the relevant libraries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- lib(gfd).
:- import minimize/2 from branch_and_bound.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Based on the alldifferent/1 global constraint -
%%    ensures each variable is pairwise different
%% from each other or has a value of zero
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alldifferent_except(Vars) :-
   % get the length of the input list
   length(Vars, N),

   % loop through all pairs of elements to check
   % if they are different
   ( for(I,1,N), param(Vars, N) do
      ( for(J, I, N), param(Vars, I) do

         element(I, Vars, X),
         element(J, Vars, Y),

         % if it isn't the same element, or an element
         % with a 0 value
         (I #\= J) ->
            % constrain to be pairwise different or have
            % one (or both) equal zero
            X #\= Y or (X #= 0 or Y #= 0)
         ;
            true
         )
      ).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The clause that the user should invoke
%% to get the program to run. This clause will
%% determine the interactions that
%% are most likely occurring by selecting the maximum
%% subset of frequencies that satisfy the constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maximize(RowFile, FreqFile, Non_Zero_Rows) :-

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%    Define
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %% specify the Rows for the intra-interaction submatricies
   Non_Zero_Rows = [Chr1_Row8, Chr1_Row9, Chr1_Row10,
      Chr1_Row11, Chr1_Row12, Chr1_Row13],

   %% specify the Frequency for the intra-interaction submatricies
   Freqs = [Chr1_Freq8, Chr1_Freq9, Chr1_Freq10,
      Chr1_Freq11, Chr1_Freq12, Chr1_Freq13],

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%    Representation of the Genome
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %% define the Row domains for the intra-interations
   Chr1_Row8 :: [0, 6],
   Chr1_Row9 :: [0, 6, 7],
   Chr1_Row10 :: [0, 6, 7, 8],
   Chr1_Row11 :: [0, 6, 7, 8, 9],
   Chr1_Row12 :: [0, 6, 7, 8, 9, 10],
   Chr1_Row13 :: [0, 6, 7, 8, 9, 10, 11],


   %% define the Freq domains for the intra-interations
   %% this is based on the Hi-C interaction matrix
   Chr1_Freq8 :: [0, 47],
   Chr1_Freq9 :: [0, 41, 58],
   Chr1_Freq10 :: [0, 33, 42, 53],
   Chr1_Freq11 :: [0, 21, 23, 32],
   Chr1_Freq12 :: [0, 19, 20, 22, 28, 41],
   Chr1_Freq13 :: [0, 19, 21, 24, 30, 37, 40],

   %% apply constraints to the data to ensure they are
   %% valid 2-tuple pairs based on the biological data.
   %% note: (0,0) is when we choose to not select
   %% anything for that Column-row pair
   %% Column 8 Constraints
   ((Chr1_Row8 #= 6) and (Chr1_Freq8 #= 47)) or
   ((Chr1_Row8 #= 0) and (Chr1_Freq8 #= 0)),

   %% Column 9 Constraints
   ((Chr1_Row9 #= 6) and (Chr1_Freq9 #= 41)) or
   ((Chr1_Row9 #= 7) and (Chr1_Freq9 #= 58)) or
   ((Chr1_Row9 #= 0) and (Chr1_Freq9 #= 0)),

   %% Column 10 Constraints
   ((Chr1_Row10 #= 6) and (Chr1_Freq10 #= 33)) or
   ((Chr1_Row10 #= 7) and (Chr1_Freq10 #= 42)) or
   ((Chr1_Row10 #= 8) and (Chr1_Freq10 #= 53)) or
   ((Chr1_Row10 #= 0) and (Chr1_Freq10 #= 0)),

   %% Column 11 Constraints
   ((Chr1_Row11 #= 6) and (Chr1_Freq11 #= 21)) or
   ((Chr1_Row11 #= 7) and (Chr1_Freq11 #= 23)) or
   ((Chr1_Row11 #= 8) and (Chr1_Freq11 #= 23)) or
   ((Chr1_Row11 #= 9) and (Chr1_Freq11 #= 32)) or
   ((Chr1_Row11 #= 0) and (Chr1_Freq11 #= 0)),

   %% Column 12 Constraints
   ((Chr1_Row12 #= 10) and (Chr1_Freq12 #= 41)) or
   ((Chr1_Row12 #= 6) and (Chr1_Freq12 #= 19)) or
   ((Chr1_Row12 #= 7) and (Chr1_Freq12 #= 20)) or
   ((Chr1_Row12 #= 8) and (Chr1_Freq12 #= 22)) or
   ((Chr1_Row12 #= 9) and (Chr1_Freq12 #= 28)) or
   ((Chr1_Row12 #= 0) and (Chr1_Freq12 #= 0)),

   %% Column 13 Constraints
   ((Chr1_Row13 #= 10) and (Chr1_Freq13 #= 37)) or
   ((Chr1_Row13 #= 11) and (Chr1_Freq13 #= 40)) or
```

```
    ((Chr1_Row13 #= 6) and (Chr1_Freq13 #= 19)) or
    ((Chr1_Row13 #= 7) and (Chr1_Freq13 #= 21)) or
    ((Chr1_Row13 #= 8) and (Chr1_Freq13 #= 24)) or
    ((Chr1_Row13 #= 9) and (Chr1_Freq13 #= 30)) or
    ((Chr1_Row13 #= 0) and (Chr1_Freq13 #= 0)),

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%    Additional Constraints
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% all the Output values will be different -
    %% want them to correspond to which Row
    %% for each has a non-zero value, multiple zeros are allowed
    alldifferent_except(Non_Zero_Rows),
    atmost(543, Non_Zero_Rows, 0),

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%    Optimize
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % heuristics could be improved to make it faster
    %%  maximize the interaction frequency from the tuple
    Cost #= -sum(Freqs),
    minimize(search(Freqs, 0, input_order, indomain_max, complete, [])
       , Cost),

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%    Output the results
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% PRINT THE FREQUENCIES
    open(FreqFile, 'write', FREQ_OUT),
    %%list the frequencies
    (foreach(X,Freqs),
       param(FREQ_OUT) do
          get_domain_as_list(X, DomList),
             (foreach(Y,DomList),
                param(FREQ_OUT) do
                   write(FREQ_OUT, Y),
                   write(FREQ_OUT, ' ')
             ),
          write(FREQ_OUT, "\n")
    ),
    close(FREQ_OUT),

    %% PRINT THE ROWS
    %% list the potential rows
    open(RowFile, 'write', ROW_OUT),
    (foreach(X,Non_Zero_Rows),
       param(ROW_OUT) do
          get_domain_as_list(X, DomList),
             (foreach(Y,DomList),
                param(ROW_OUT) do
                   write(ROW_OUT, Y),
                   write(ROW_OUT, ' ')
             ),
             write(ROW_OUT, "\n")
    ),
    close(ROW_OUT).
```

A.4 Overview of the CLP knowledge Representation for a Inter-Interaction Subproblem

Program A.4 A Subset of the Refined CLP Program with a Minimal N-Queens Knowledge Representation for Detecting Inter-Interactions Between Chromosome 1 and Chromosome 2.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load the relevant libraries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- lib(gfd).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Based on the alldifferent/1 global constraint -
%%    ensures each variable is pairwise different
%% from each other or has a value of zero
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alldifferent_except(Vars) :-
   % get the length of the input list
   length(Vars, N),

   % loop through all pairs of elements to check
   % if they are different
   ( for(I,1,N), param(Vars, N) do
      ( for(J, I, N), param(Vars, I) do

         element(I, Vars, X),
         element(J, Vars, Y),

         % if it isn't the same element, or an element
         % with a 0 value
         (I #\= J) ->
            % constrain to be pairwise different or have
            % one (or both) equal zero
            X #\= Y or (X #= 0 or Y #= 0)
         ;
            true
         )
      ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The clause that the user should invoke
%% to get the program to run. This clause will
%% determine the interactions that
%% are most likely occurring by selecting the maximum
%% subset of frequencies that satisfy the constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maximize(RowFile, FreqFile, Non_Zero_Rows) :-

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%     Define
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %% specify the Rows for the inter-interaction submatricies
   Non_Zero_Rows = [Chr1_Chr2_Row562, Chr1_Chr2_Row563,
      Chr1_Chr2_Row564, Chr1_Chr2_Row565, Chr1_Chr2_Row566],

   %% specify the Frequency for the inter-interaction submatricies
   Freqs = [Chr1_Chr2_Freq562, Chr1_Chr2_Freq563,
      Chr1_Chr2_Freq564, Chr1_Chr2_Freq565, Chr1_Chr2_Freq566],

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%     Representation of the Genome
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %% define the Row domains for the inter-interations
   %% these should be 0..N where N is the number of columns in the submatrix
   %% chromosome1 - chromosome 2
   Chr1_Chr2_Row562 :: [0, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 462,
                  545, 546, 547, 548, 549, 550, 551, 552, 553,
                  554, 555, 556, 557],
   Chr1_Chr2_Row563 :: [0, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 543,
                  544, 545, 546, 547, 548, 549, 550, 551, 552,
                  553, 554, 555, 556, 557],
   Chr1_Chr2_Row564 :: [0, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
                  544, 545, 546, 547, 548, 549, 550, 551, 552,
                  553, 554, 555, 556, 557],
   Chr1_Chr2_Row565 :: [0, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
                  544, 545, 546, 547, 548, 549, 550, 551, 552,
                  553, 554, 555, 556, 557],
   Chr1_Chr2_Row566 :: [0, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
                  545, 546, 547, 548, 549, 550, 551, 552, 553,
                  554, 555, 556, 557],

   %% define the Freq domains for the inter-interations
   %% this is based on the Hi-C interaction matrix
   Chr1_Chr2_Freq562 :: [0, 3, 4, 6, 5, 1, 2],
   Chr1_Chr2_Freq563 :: [0, 4, 5, 6, 3, 1, 2],
   Chr1_Chr2_Freq564 :: [0, 4, 5, 2, 1, 3],
   Chr1_Chr2_Freq565 :: [0, 4, 6, 5, 1, 2, 3],
   Chr1_Chr2_Freq566 :: [0, 3, 4, 5, 1, 2],

   %% apply constraints to the data to ensure they are
   %% valid 2-tuple pairs based on the
   %% biological data. note: (0,0) is when we choose
   %% to not select anything for that
   %% Column row pair
```

```
    %% Column562 Constraints
    ((Chr1_Chr2_Row562 #= 10) and (Chr1_Chr2_Freq562 #= 3)) or
    ((Chr1_Chr2_Row562 #= 11) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 12) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 13) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 14) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 15) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 462) and (Chr1_Chr2_Freq562 #= 3)) or
    ((Chr1_Chr2_Row562 #= 545) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 546) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 547) and (Chr1_Chr2_Freq562 #= 1)) or
    ((Chr1_Chr2_Row562 #= 548) and (Chr1_Chr2_Freq562 #= 2)) or
    ((Chr1_Chr2_Row562 #= 549) and (Chr1_Chr2_Freq562 #= 2)) or
    ((Chr1_Chr2_Row562 #= 550) and (Chr1_Chr2_Freq562 #= 3)) or
    ((Chr1_Chr2_Row562 #= 551) and (Chr1_Chr2_Freq562 #= 4)) or
    ((Chr1_Chr2_Row562 #= 552) and (Chr1_Chr2_Freq562 #= 5)) or
    ((Chr1_Chr2_Row562 #= 553) and (Chr1_Chr2_Freq562 #= 6)) or
    ((Chr1_Chr2_Row562 #= 554) and (Chr1_Chr2_Freq562 #= 5)) or
    ((Chr1_Chr2_Row562 #= 555) and (Chr1_Chr2_Freq562 #= 3)) or
    ((Chr1_Chr2_Row562 #= 556) and (Chr1_Chr2_Freq562 #= 4)) or
    ((Chr1_Chr2_Row562 #= 557) and (Chr1_Chr2_Freq562 #= 4)) or
    ((Chr1_Chr2_Row562 #= 6) and (Chr1_Chr2_Freq562 #= 3)) or
    ((Chr1_Chr2_Row562 #= 7) and (Chr1_Chr2_Freq562 #= 4)) or
    ((Chr1_Chr2_Row562 #= 8) and (Chr1_Chr2_Freq562 #= 6)) or
    ((Chr1_Chr2_Row562 #= 9) and (Chr1_Chr2_Freq562 #= 5)) or
    ((Chr1_Chr2_Freq562 #= 0) and (Chr1_Chr2_Row562 #= 0)),

    %% Column563 Constraints
    ((Chr1_Chr2_Row563 #= 10) and (Chr1_Chr2_Freq563 #= 3)) or
    ((Chr1_Chr2_Row563 #= 11) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 12) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 13) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 14) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 15) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 543) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 544) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 545) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 546) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 547) and (Chr1_Chr2_Freq563 #= 1)) or
    ((Chr1_Chr2_Row563 #= 548) and (Chr1_Chr2_Freq563 #= 2)) or
    ((Chr1_Chr2_Row563 #= 549) and (Chr1_Chr2_Freq563 #= 2)) or
    ((Chr1_Chr2_Row563 #= 550) and (Chr1_Chr2_Freq563 #= 3)) or
    ((Chr1_Chr2_Row563 #= 551) and (Chr1_Chr2_Freq563 #= 5)) or
    ((Chr1_Chr2_Row563 #= 552) and (Chr1_Chr2_Freq563 #= 5)) or
    ((Chr1_Chr2_Row563 #= 553) and (Chr1_Chr2_Freq563 #= 6)) or
    ((Chr1_Chr2_Row563 #= 554) and (Chr1_Chr2_Freq563 #= 5)) or
    ((Chr1_Chr2_Row563 #= 555) and (Chr1_Chr2_Freq563 #= 4)) or
    ((Chr1_Chr2_Row563 #= 556) and (Chr1_Chr2_Freq563 #= 4)) or
    ((Chr1_Chr2_Row563 #= 557) and (Chr1_Chr2_Freq563 #= 3)) or
    ((Chr1_Chr2_Row563 #= 6) and (Chr1_Chr2_Freq563 #= 4)) or
    ((Chr1_Chr2_Row563 #= 7) and (Chr1_Chr2_Freq563 #= 5)) or
    ((Chr1_Chr2_Row563 #= 8) and (Chr1_Chr2_Freq563 #= 6)) or
    ((Chr1_Chr2_Row563 #= 9) and (Chr1_Chr2_Freq563 #= 6)) or
    ((Chr1_Chr2_Freq563 #= 0) and (Chr1_Chr2_Row563 #= 0)),

    %% Column564 Constraints
    ((Chr1_Chr2_Row564 #= 10) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Row564 #= 11) and (Chr1_Chr2_Freq564 #= 2)) or
    ((Chr1_Chr2_Row564 #= 12) and (Chr1_Chr2_Freq564 #= 2)) or
    ((Chr1_Chr2_Row564 #= 13) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 14) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 15) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 16) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 544) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 545) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 546) and (Chr1_Chr2_Freq564 #= 1)) or
    ((Chr1_Chr2_Row564 #= 547) and (Chr1_Chr2_Freq564 #= 2)) or
    ((Chr1_Chr2_Row564 #= 548) and (Chr1_Chr2_Freq564 #= 2)) or
    ((Chr1_Chr2_Row564 #= 549) and (Chr1_Chr2_Freq564 #= 2)) or
    ((Chr1_Chr2_Row564 #= 550) and (Chr1_Chr2_Freq564 #= 3)) or
    ((Chr1_Chr2_Row564 #= 551) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Row564 #= 552) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Row564 #= 553) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Row564 #= 554) and (Chr1_Chr2_Freq564 #= 3)) or
    ((Chr1_Chr2_Row564 #= 555) and (Chr1_Chr2_Freq564 #= 3)) or
    ((Chr1_Chr2_Row564 #= 556) and (Chr1_Chr2_Freq564 #= 2)) or
    ((Chr1_Chr2_Row564 #= 557) and (Chr1_Chr2_Freq564 #= 3)) or
    ((Chr1_Chr2_Row564 #= 6) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Row564 #= 7) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Row564 #= 8) and (Chr1_Chr2_Freq564 #= 5)) or
```

```
    ((Chr1_Chr2_Row564 #= 9) and (Chr1_Chr2_Freq564 #= 4)) or
    ((Chr1_Chr2_Freq564 #= 0) and (Chr1_Chr2_Row564 #= 0)),

    %% Column565 Constraints
    ((Chr1_Chr2_Row565 #= 10) and (Chr1_Chr2_Freq565 #= 4)) or
    ((Chr1_Chr2_Row565 #= 11) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 12) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 13) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 14) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 15) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 16) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 544) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 545) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 546) and (Chr1_Chr2_Freq565 #= 1)) or
    ((Chr1_Chr2_Row565 #= 547) and (Chr1_Chr2_Freq565 #= 2)) or
    ((Chr1_Chr2_Row565 #= 548) and (Chr1_Chr2_Freq565 #= 2)) or
    ((Chr1_Chr2_Row565 #= 549) and (Chr1_Chr2_Freq565 #= 2)) or
    ((Chr1_Chr2_Row565 #= 550) and (Chr1_Chr2_Freq565 #= 3)) or
    ((Chr1_Chr2_Row565 #= 551) and (Chr1_Chr2_Freq565 #= 5)) or
    ((Chr1_Chr2_Row565 #= 552) and (Chr1_Chr2_Freq565 #= 5)) or
    ((Chr1_Chr2_Row565 #= 553) and (Chr1_Chr2_Freq565 #= 4)) or
    ((Chr1_Chr2_Row565 #= 554) and (Chr1_Chr2_Freq565 #= 4)) or
    ((Chr1_Chr2_Row565 #= 555) and (Chr1_Chr2_Freq565 #= 3)) or
    ((Chr1_Chr2_Row565 #= 556) and (Chr1_Chr2_Freq565 #= 3)) or
    ((Chr1_Chr2_Row565 #= 557) and (Chr1_Chr2_Freq565 #= 2)) or
    ((Chr1_Chr2_Row565 #= 6) and (Chr1_Chr2_Freq565 #= 4)) or
    ((Chr1_Chr2_Row565 #= 7) and (Chr1_Chr2_Freq565 #= 4)) or
    ((Chr1_Chr2_Row565 #= 8) and (Chr1_Chr2_Freq565 #= 6)) or
    ((Chr1_Chr2_Row565 #= 9) and (Chr1_Chr2_Freq565 #= 5)) or
    ((Chr1_Chr2_Freq565 #= 0) and (Chr1_Chr2_Row565 #= 0)),

    %% Column556 Constraints
    ((Chr1_Chr2_Row566 #= 10) and (Chr1_Chr2_Freq566 #= 3)) or
    ((Chr1_Chr2_Row566 #= 11) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 12) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 13) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 14) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 15) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 16) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 545) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 546) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 547) and (Chr1_Chr2_Freq566 #= 1)) or
    ((Chr1_Chr2_Row566 #= 548) and (Chr1_Chr2_Freq566 #= 2)) or
    ((Chr1_Chr2_Row566 #= 549) and (Chr1_Chr2_Freq566 #= 2)) or
    ((Chr1_Chr2_Row566 #= 550) and (Chr1_Chr2_Freq566 #= 3)) or
    ((Chr1_Chr2_Row566 #= 551) and (Chr1_Chr2_Freq566 #= 4)) or
    ((Chr1_Chr2_Row566 #= 552) and (Chr1_Chr2_Freq566 #= 4)) or
    ((Chr1_Chr2_Row566 #= 553) and (Chr1_Chr2_Freq566 #= 4)) or
    ((Chr1_Chr2_Row566 #= 554) and (Chr1_Chr2_Freq566 #= 3)) or
    ((Chr1_Chr2_Row566 #= 555) and (Chr1_Chr2_Freq566 #= 2)) or
    ((Chr1_Chr2_Row566 #= 556) and (Chr1_Chr2_Freq566 #= 2)) or
    ((Chr1_Chr2_Row566 #= 557) and (Chr1_Chr2_Freq566 #= 2)) or
    ((Chr1_Chr2_Row566 #= 6) and (Chr1_Chr2_Freq566 #= 3)) or
    ((Chr1_Chr2_Row566 #= 7) and (Chr1_Chr2_Freq566 #= 4)) or
    ((Chr1_Chr2_Row566 #= 8) and (Chr1_Chr2_Freq566 #= 5)) or
    ((Chr1_Chr2_Row566 #= 9) and (Chr1_Chr2_Freq566 #= 5)) or
    ((Chr1_Chr2_Freq566 #= 0) and (Chr1_Chr2_Row566 #= 0)),

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%    Additional Constraints
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% all the Output values will be different - want them to correspond
    %% to which Row for each has a non-zero value, multiple zeros
    %% are allowed
    alldifferent_except(Non_Zero_Rows),
    atmost(49, Non_Zero_Rows, 0),

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%    Optimize
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%  maximize the interaction frequency from the tuple
    Cost #= -sum(Freqs),
    minimize(search(Freqs, 0, input_order, indomain_max, complete, [])
       , Cost),

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%    Output the results
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% OUTPUT THE FREQUENCY VALUES
    open(FreqFile, 'write', FREQ_OUT),
```

```
    %%list the frequencies
254 (foreach(X,Freqs),
        param(FREQ_OUT) do
256         get_domain_as_list(X, DomList),
                %% for each value in the domain of
258             %% X: print it out
                (foreach(Y,DomList),
260                 param(FREQ_OUT) do
                        write(FREQ_OUT, Y),
262                     write(FREQ_OUT, '␣')
                ),
264         write(FREQ_OUT, "\n")
    ),
266 close(FREQ_OUT),

268 %% OUTPUT THE ROW VALUES
    open(RowFile, 'write', ROW_OUT),
270 (foreach(X,Non_Zero_Rows),
        param(ROW_OUT) do
272         get_domain_as_list(X, DomList),
                %% for each value in the domain of
274             %% X: print it out
                (foreach(Y,DomList),
276                 param(ROW_OUT) do
                        write(ROW_OUT, Y),
278                     write(ROW_OUT, '␣')
                ),
280             write(ROW_OUT, "\n")
    ),
282 close(ROW_OUT).
```

Appendix B

Perl Programs for the Automation of Portions of the Original CLP Program

B.1 compression_of_rows.pl

```
#!/usr/bin/perl
## generates the CLP representation of the interaction frequencies
## argument 1: corrected HiC matrix file
## argument 2: the number of bins you want to condense together
## Kimberly MacKay April 21, 2015

use strict;
use warnings;

## check to ensure two arguments was passed in
die "ERROR: must pass in two argumnets." if @ARGV != 2;

my $HiC_file = $ARGV[0];
my $combine_num = $ARGV[1];

## open the files
open HIC, "$HiC_file" or die "ERROR: $HiC_file could not be opened.";
chomp(my @matrix = <HIC>);
close HIC;

my @adjusted_line_chr1;
my @adjusted_line_chr2;
my @adjusted_line_chr3;

##print "genome(1, [](\n";

##for each line in the matrix
for(my $i = 1; $i <= $#matrix; $i++)
{
   my @line = split /\t/, $matrix[$i];

   ##for each other column, check to see if it is a NA (a non-interaction)
   for(my $j = 1; $j <= $#line; $j++)
   {
      ## adjust the values and re-store it in the line
      ## if it is in chr 1
      if($j < 559)
      {
         ## if it corresponds to the interaction between the same bin
         if($j == $i)
         {
            $adjusted_line_chr1[$j] = 0;
         }
         ## get rid of half the matrix (it is symmerical)
         elsif($j < $i)
         {
            $adjusted_line_chr1[$j] = 0;
         }
         elsif(($line[$j] =~ /^NA$/) || ($line[$j] =~ /^0$/) )
         {
            $adjusted_line_chr1[$j] = 0;
         }
         ## adjust the interaction value so it can be minimized with
         ## interval constraints
         else
         {
            my $temp = $line[$j] * -1000000000;
            $adjusted_line_chr1[$j] = $temp;
         }
      }
      ## if it is in chr 2
      elsif($j < 1013)
      {
         ## if it corresponds to the interaction between the same bin
```

```
            if($j == $i)
            {
               $adjusted_line_chr2[$j] = 0;
            }
            ## get rid of half the matrix (it is symmerical)
            elsif($j < $i)
            {
               $adjusted_line_chr2[$j] = 0;
            }
            elsif(($line[$j] =~ /^NA$/) || ($line[$j] =~ /^0$/) )
            {
               $adjusted_line_chr2[$j] = 0;
            }
            ## adjust the interaction value so it can be minimized with
            ## interval constraints
            else
            {
               my $temp = $line[$j] * -1000000000;
               $adjusted_line_chr2[$j] = $temp;
            }
         }
         ## if it is in chr 3
         elsif($j < 1259)
         {
            ## if it corresponds to the interaction between the same bin
            if($j == $i)
            {
               $adjusted_line_chr3[$j] = 0;
            }
            ## get rid of half the matrix (it is symmerical)
            elsif($j < $i)
            {
               $adjusted_line_chr3[$j] = 0;
            }
            elsif(($line[$j] =~ /^NA$/) || ($line[$j] =~ /^0$/) )
            {
               $adjusted_line_chr3[$j] = 0;
            }
            ## adjust the interaction value so it can be minimized with
            ## interval constraints
            else
            {
               my $temp = $line[$j] * -1000000000;
               $adjusted_line_chr3[$j] = $temp;
            }
         }
      }

      ## loop through each of the chromosome arrays and print out the
      ## average of the N adjacent elements
      for(my $k = 1; $k <= $#adjusted_line_chr1; $k = $k + $combine_num)
      {
         my $avg = 0;
         my $count = 0;
         for(my $l = 0;
            $l < $combine_num && defined($adjusted_line_chr1[$k+$l]);
            $l++)
         {
            $avg = $avg + $adjusted_line_chr1[$k+$l];
            $count ++;
         }
         $avg = $avg/$count;

         print "$avg\t";
      }

      for(my $k = 559; $k <= $#adjusted_line_chr2; $k = $k + $combine_num)
      {
         my $avg = 0;
         my $count = 0;
         for(my $l = 0;
            $l < $combine_num && defined($adjusted_line_chr2[$k+$l]);
            $l++)
         {
            $avg = $avg + $adjusted_line_chr2[$k+$l];
            $count++;
         }
         $avg = $avg/$count;

         print "$avg\t";
      }
```

```
146
    for(my $k = 1013; $k <= $#adjusted_line_chr3; $k = $k + $combine_num)
148 {
        my $avg = 0;
150     my $count = 0;
        for(my $l = 0;
152         $l < $combine_num && defined($adjusted_line_chr3[$k+$l]);
            $l++)
154     {
            $avg = $avg + $adjusted_line_chr3[$k+$l];
156         $count++;
        }
158     $avg = $avg/$count;

160     print "$avg\t";
    }
162 print "\n";
}
```

B.2 compression_of_columns.pl

```
1  #!/usr/bin/perl
   ## generates the CLP representation of the interaction frequencies
3  ## argument 1: corrected HiC matrix file
   ## argument 2: the number of bins you want to condense together
5  ## Kimberly MacKay April 21, 2015

7  use strict;
   use warnings;
9
   ## check to ensure two arguments was passed in
11 die "ERROR: must pass in two argumnets." if @ARGV != 2;

13 my $HiC_file = $ARGV[0];
   my $combine_num = $ARGV[1];
15
   ## open the files
17 open HIC, "$HiC_file" or die "ERROR: $HiC_file could not be opened.";
   chomp(my @matrix = <HIC>);
19 close HIC;

21 my @adjusted_line_chr1;
   my $avg = 0;
23 my $count = 0;
   my @line = split /\t/, $matrix[1];
25
   print "genome(1,,[](";
27
   ## for each line in the file
29 for(my $i = 1; $i <= $#matrix; $i=$i+$combine_num-1)
   {
31    ## for each element in the line
      for(my $l = 0; $l <= $#line; $l++)
33    {
         $avg = 0;
35       $count = 0;

37       ## average 35 of the elements
         for(my $j = $i; $j < $i + $combine_num ; $j++)
39       {
            if(defined($matrix[$j]))
41          {
               my @line1 = split /\t/, $matrix[$j];
43             $avg = $avg + $line1[$l];
               $count++;
45          }
         }
47
            $avg = $avg/$count;
49          $adjusted_line_chr1[$l] = $avg;

51    }
      ## print out the new line
53    print "\n\t[](";
      for(my $k = 0; $k <= $#adjusted_line_chr1; $k++)
55    {
         if($k > 0 && $k <= $#adjusted_line_chr1)
```

```
57        {
              print ",␣";
59        }

61        printf("%d", $adjusted_line_chr1[$k]);
      }
63    print "),";
  }
65
  print "\n)).";
```

B.3 generate_bins.pl

```
1  #!/usr/bin/perl
   ## Will generate the CLP representation of all the genomic bins from the HiC Data
3  ## argument 1: corrected HiC matrix file downloaded from GEO
   ## argument 2: the number of bins you want to condense together
5  ## Kimberly MacKay April 22, 2015

7  use strict;
   use warnings;
9
   ## check to ensuretwo arguments was passed in
11 die "ERROR:␣must␣pass␣in␣two␣argumnets." if @ARGV != 2;

13 ## grab the command line arguments
   my $HiC_file = $ARGV[0];
15 my $combine_num = $ARGV[1];

17 ## open the file
   open HIC, "$HiC_file" or die "ERROR:␣$HiC_file␣could␣not␣be␣opened.";
19 chomp(my @matrix = <HIC>);
   close HIC;
21
   my ($bin_name, $bin_chr, $bin_start, $bin_stop);
23
   my @bins = split /\t/, $matrix[0];
25 my $num = 1;
   my $combined_bin = "";
27 my $chr_prev = 1;

29 ## print out the beginning of the representation
   print "spombeBins([](";
31
   ## for each of the bins combined the appropriate number and then
33 ## print out the representation
   for(my $i = 0; $i <= $#bins; $i++)
35 {
      ## check wether or not you have already accumulated
37    ## enough bins to condense them
      if($num > $combine_num)
39    {
         print "\"".$combined_bin."\",␣";
41       $num = 1;
         $combined_bin = "";
43    }
      ## extract the relavent bin information
45    if($bins[$i] =~ /(bin\d+)\|.*\|chr(\d+):(\d+)-(\d+)/)
      {
47       $bin_name = $1;
         $bin_chr = $2;
49       $bin_start = $3;
         $bin_stop = $4;
51    }
      ## check to make sure subsequent bins you want to condense
53    ## together are on the same chromosome
      if($bin_chr == $chr_prev)
55    {
         if($num > 1)
57       {
            $combined_bin = $combined_bin ."_". $bin_name;
59       }
         else
61       {
            $combined_bin = $bin_name;
63       }
         $num++;
```

```
65      }
        ## if the subsequent bins are not on the same chromosome
67      ## if there are other bins that are currently stored
        ## print out the other bins before accumulating bins on the new chromosome
69      elsif($num > 1)
        {
71          print "\"".$combined_bin."\",␣";
            $num = 1;
73          print "\nspombeBins([](";
            $combined_bin = $bin_name;
75          $num++;
        }
77      ## if all the bins on the previous chromosome have already been printed out
        ## just continue on accumulating bins
79      elsif($num == 1)
        {
81          $num = 1;
            $combined_bin = $bin_name;
83          $num++;
        }
85      $chr_prev = $bin_chr;
    }
87
    ## print the remainder of the representation
89      print "\"".$combined_bin."\"";

91
    print ")).";
```

B.4 generate_vars.pl

```
    #!/usr/bin/perl
2   ## will generate the vars needed for the CLP eclispe program
    ## argument 1: the N or a N by N matrix
4   ## Kimberly MacKay April 25, 2015

6   use strict;
    use warnings;
8
    ## check to ensure one argument was passed in
10  die "ERROR:␣must␣pass␣in␣one␣argumnet." if @ARGV != 1;

12  my $num = $ARGV[0];

14
    print "Rows␣=␣[](\n";
16
    for(my $i = 1; $i <= $num; $i++)
18  {
        print "\t[](";
20      for(my $j = 1; $j <= $num; $j++)
        {
22          if($j > 1)
            {
24              print ",␣"
            }
26          print "V".$i."_".$j;
        }
28      print "),\n";
    }
30
    print "),\n";
32
    print "Cols␣=␣[](\n";
34
    for(my $i = 1; $i <= $num; $i++)
36  {
        print "\t[](";
38      for(my $j = 1; $j <= $num; $j++)
        {
40          if($j > 1)
            {
42              print ",␣"
            }
44          print "V".$j."_".$i;
        }
46      print "),\n";
```

```
}

print "),\n";

print "SlnVars = [](\n";

for(my $i = 1; $i <= $num; $i++)
{
   print "\t";
   for(my $j = 1; $j <= $num; $j++)
   {
      print "V".$i."_".$j.", ";
   }
   print "\n";
}

print "),\n";
```

Appendix C

Perl Programs for the Automated Generation of the Minimal, Non-Redundant CLP Program

C.1 generate_minimal_non_redundant_eclipse_program_intra.pl

```
#!/usr/bin/perl
## will generate the vars needed for the CLP eclispe program
## argument 1: the size of the matrix
## argument 2: the interaction matrix
## argument 3:the chromsome of interst
## argument 4: chr start
## argument 5: chr stop
## argument 6: chr size

## Kimberly MacKay June 10, 2016

use strict;
use warnings;
use List::MoreUtils 'uniq';

## check to ensure five arguments was passed in
die "ERROR: must pass in six argumnets." if @ARGV != 6;

my $num_variables = $ARGV[0];
my $HiC_file = $ARGV[1];
my $chr = $ARGV[2];
my $chr_start = $ARGV[3];
my $chr_stop = $ARGV[4];
my $chr_size = $ARGV[5];

my $scale = 1000;

##############################################################################
## parse the interaction matrix
##############################################################################

## open the interaction matrix file
open HIC, "$HiC_file" or die "ERROR: $HiC_file could not be opened.";
chomp(my @matrix_file = <HIC>);
close HIC;

my @frequencies;

## for each line after the header line
for(my $row = 1; $row <= $#matrix_file; $row++)
{
   ## split the line
   my @matrix_line = split /\t/, $matrix_file[$row];

   ## loop through the entire file to extract the frequencies
   ## note: we only have to extract one half of the matrix since it is symmetric
   ## along the diagonal
   for(my $col = 1; $col <= $row; $col++)
   {
      ## adjusts NA's to 0's
      if($matrix_line[$col] =~ "NA")
      {
         $frequencies[$row][$col] = 0;
      }
      else
      {
         ## trying a smaller integer size to see if it improves speed
         $frequencies[$row][$col] = int($matrix_line[$col]*$scale);
      }
   }
}

## loop through the submatrix to determine the non-zero rows
my %non_zero;
```

```
my @non_zero_row_index;
my %only_zero_rows;

for(my $row = $chr_start; $row <= $chr_stop; $row++)
{
   my $only_zero = 1;

   for(my $col = $chr_start; $col <= $row; $col++)
   {
      ##check to see if it is non-zero
      if($frequencies[$row][$col] != 0)
      {
         $only_zero = 0;
         $non_zero{$row}{$col} = $col;
      }
   }

   if($only_zero)
   {
      $only_zero_rows{$row} = $row;
   }
}

@non_zero_row_index = (sort {$a <=> $b} keys %non_zero);

## print out the non-zero rows
my $out_filename = "../chr".$chr."_non_zero_bins.txt";
open OUT, '>', "$out_filename" or die "ERROR: $out_filename could not be opened.";

for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
   print OUT "$non_zero_row_index[$i]\n";
}

close OUT;

#########################################################################
## printing the program
#########################################################################

print ":- lib(gfd).\n\n";
#print "import minimize/2 from lib(branch_and_bound).";

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "%%    maximize the interactions that are occurring based on\n";
print "%%    resultant frequency table\n";
print "%%    Output is the output paramater which will store the list of cells\n";
print "%%    to keep for each Rowumn\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

print "maximize(File, Non_Zero_Rows) :-\n\n";

#############################################################################
##     Print the variables based on N
#############################################################################
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%    Define\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";


## print Row variables
print "\t%% specify the Rows for the intra-interaction submatricies";

print "\n\tNon_Zero_Rows = [";
for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
   ## account for the last variable, it won't have a comma
   if($i == $#non_zero_row_index)
   {
      print "Chr".$chr."_Row".$non_zero_row_index[$i]."],";
   }
   else
   {
      print "Chr".$chr."_Row".$non_zero_row_index[$i].", ";
   }
}

print "\n";


```

```
## print frequency variables
print "\n\n\t%% specify the Frequency for the intra-interaction submatricies";

print "\n\tFreqs = [";
for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
   ## account for the last variable, it won't have a comma
   if($i == $#non_zero_row_index)
   {
      print "Chr".$chr."_Freq".$non_zero_row_index[$i]."],";
   }
   else
   {
      print "Chr".$chr."_Freq".$non_zero_row_index[$i].", ";
   }
}
print "\n";

#########################################################################
##    Print the genome representation
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%    Representation of the Genome\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

## print Row domains
print "\t% define the Row domains for the intra-interations";

## intra-interaction Row domains

print "\n\t% chromosome".$chr."\n";
for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
   if($i == $#non_zero_row_index)
   {
      print "\tChr".$chr."_Row".$non_zero_row_index[$i].
         " :: [0,".$chr_start."..".$non_zero_row_index[$i]."],\n";
   }
   else
   {
      print "\tChr".$chr."_Row".$non_zero_row_index[$i].
         " :: [0,".$chr_start."..".$non_zero_row_index[$i]."],\n";
   }
}

############################
## print frequency domains
print "\n\n\t% define the Freq domains for the intra-interations\n";
print "\t% this is based on the Hi-C interaction matrix";

print "\n\t% chromosome".$chr."\n";

#my %values


for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
   my @domain;
   my @unique_values;

   for(my $col = $chr_start; $col <= $non_zero_row_index[$i]; $col++)
   {

      ##check to see if it is non-zero
      if($frequencies[$non_zero_row_index[$i]][$col] != 0)
      {
         push @domain, $frequencies[$non_zero_row_index[$i]][$col];
      }
   }

   @unique_values = uniq @domain;

   print "\tChr".$chr."_Freq".$non_zero_row_index[$i]." :: [0";

   for(my $u = 0; $u <= $#unique_values; $u++)
   {
      print ", ".$unique_values[$u];
   }
   print "],\n";
```

```
}
print "\n";


##############################
## print Row constraints
print "\t% apply constraints to the data to ensure they".
      " are valid 2-tuple pairs based on the \n";
print "\t% biological data. note: (0,0) is when we choose".
      " to not select anything for that\n";
print "\t% Column row pair\n";

print "\n\t% INTRA-INTERACTIONS\n";

print "\n\t% chromosome".$chr."\n";

for(my $row = $chr_start; $row <= $chr_stop; $row++)
{
   ## if non-zero values exist
   if(defined $non_zero{$row})
   {
      print "\n\t% Column ".$row." Constraints\n";

      ## first print the non-zero options
      for my $nz_col (sort keys $non_zero{$row})
      {

         print "\t((Chr".$chr."_Row".$row." #= ".$nz_col.") and (Chr".
            $chr."_Freq".$row." #= ".$frequencies[$row][$nz_col].")) or \n";
      }
      ## print the "0" case
      print "\t((Chr".$chr."_Row".$row." #= 0) and (Chr".$chr."_Freq"
         .$row." #= 0)),\n";
   }
}
print "\n";



#########################################################################
##    Print the additional constraints
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%% Additional Constraints\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

print "\t% all the Output values will be different -want them to".
   "correspond to which row\n";
print "\t% for each has a non-zero value, multiple zeros are allowed\n";

print "\talldifferent(Non_Zero_Rows),\n";
print "\tatmost(".($#non_zero_row_index-1).", Non_Zero_Rows, 0),\n";
print "\n";


#########################################################################
##    Print the optimization
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%% Optimize\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

print "\t% heuristics could be improved to make it faster\n";
print "\t%% maximize the interaction frequency from the tuple\n";
print "\tCost #= -sum(Freqs),\n";
#print "\tminimize(\n";
print "\t\tsearch(Freqs, 0, input_order, indomain_max, bb_min(Cost), []),\n";
#print "\t\tCost\n";
#print "\t).\n";

#########################################################################
##    Output the results
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%% Output the results\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";
```

```
print "\topen(File,␣'write',␣OUT),\n";
print "\t(foreach(X,Non_Zero_Rows),\n";
print "\t\tparam(OUT)␣do\n";
print "\t\t\tget_domain_as_list(X,␣DomList),\n";
print "\t\t\t\t(foreach(Y,DomList),\n";
print "\t\t\t\t\tparam(OUT)␣do\n";
print "\t\t\t\t\t\twrite(OUT,␣Y),\n";
print "\t\t\t\t\t\twrite(OUT,␣'␣')\n";
print "\t\t\t\t),\n";
print "\t\t\t\twrite(OUT,␣\"\\n\")\n";
print "\t),\n";
print "\tclose(OUT).\n";
```

C.2 generate_minimal_non_redundant_eclipse_program_inter.pl

```
#!/usr/bin/perl
## will generate the vars needed for the CLP eclispe program
## argument 1: the size of the matrix
## argument 2: the interaction matrix
## argument 3: chr1
## argument 4: chr1 start
## argument 5: chr1 stop
## argument 6: chr2
## argument 7: chr2 start
## argument 8: chr2 stop
## argument 9: chr2 size

## Kimberly MacKay June 13, 2016

use strict;
use warnings;
use List::MoreUtils 'uniq';

## check to ensure two arguments was passed in
die "ERROR:␣must␣pass␣in␣nine␣argumnets." if @ARGV != 9;

my $num_variables = $ARGV[0];
my $HiC_file = $ARGV[1];

my $chr1 = $ARGV[2];
my $chr1_start = $ARGV[3];
my $chr1_stop = $ARGV[4];

my $chr2 = $ARGV[5];
my $chr2_start = $ARGV[6];
my $chr2_stop = $ARGV[7];
my $chr2_size = $ARGV[8];

my $scale = 500;


#########################################################################
## parse the interaction matrix
#########################################################################

## open the interaction matrix file
open HIC, "$HiC_file" or die "ERROR:␣$HiC_file␣could␣not␣be␣opened.";
chomp(my @matrix_file = <HIC>);
close HIC;

my @frequencies;

## for each line after the header line
for(my $row = 1; $row <= $#matrix_file; $row++)
{
   ## split the line
   my @matrix_line = split /\t/, $matrix_file[$row];

   ## loop through the entire file to extract the frequencies
   ## note: we only have to extract one half of the matrix since it is symmetric
   ## along the diagonal
   for(my $col = 1; $col <= $row; $col++)
   {
      ## adjusts NA's to 0's
      if($matrix_line[$col] =~ "NA")
      {
```

```
            $frequencies[$row][$col] = 0;
        }
        else
        {
            ## trying a smaller integer size to see if it improves speed
            $frequencies[$row][$col] = int($matrix_line[$col]*$scale);
        }
    }
}


## loop through the submatrix to determine the non-zero rows
my %non_zero;
my @non_zero_row_index;
my %only_zero_rows;

for(my $row = $chr2_start; $row <= $chr2_stop; $row++)
{
    my $only_zero = 1;

    for(my $col = $chr1_start; $col <= $chr1_stop; $col++)
    {
        ##check to see if it is non-zero
        if($frequencies[$row][$col] != 0)
        {
            $only_zero = 0;
            $non_zero{$row}{$col} = $col;
        }
    }

    if($only_zero)
    {
        $only_zero_rows{$row} = $row;
    }
}

@non_zero_row_index = (sort {$a <=> $b} keys %non_zero);

## print out the non-zero rows
my $out_filename = "../chr".$chr1."_chr".$chr2."_non_zero_bins.txt";
open OUT, '>', "$out_filename" or die "ERROR: $out_filename could not be opened.";

for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
    print OUT "$non_zero_row_index[$i]\n";
}

close OUT;


##############################################################################
## printing the program
##############################################################################

print ":- lib(gfd).\n\n";

print "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "%%    maximize the interactions that are occurring based on\n";
print "%%    resultant frequency table\n";
print "%%    Output is the output paramater which will store the list of cells\n";
print "%%    to keep for each Rowumn\n";
print "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";

print "maximize(File, Non_Zero_Rows) :-\n\n";

##############################################################################
##    Print the variables based on N
##############################################################################
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%    Define\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";


## print Row variables

print "\t%% specify the Rows for the inter-interaction submatricies";

print "\n\tNon_Zero_Rows = [";
for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
    if($i == $#non_zero_row_index)
```

```
    {
        print "Chr".$chr1."_Chr".$chr2."_Row".$non_zero_row_index[$i]."],";
    }
    else
    {
        print "Chr".$chr1."_Chr".$chr2."_Row".$non_zero_row_index[$i]."], ";
    }
}

## print frequency variables
print "\n\n\t%% specify the Frequency for the inter-interaction submatricies";
print "\n\tFreqs = [";

for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
    if($i == $#non_zero_row_index)
    {
        print "Chr".$chr1."_Chr".$chr2."_Freq".$non_zero_row_index[$i]."],";
    }
    else
    {
        print "Chr".$chr1."_Chr".$chr2."_Freq".$non_zero_row_index[$i]."], ";
    }
}
print "\n";


#########################################################################
##    Print the genome representation
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%    Representation of the Genome\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

## inter-interaction Row domains
print "\t% define the Row domains for the inter-interations\n";
print "\t%these should be 0..N where N is the number of columns in the submatrix";

print "\n\t% chromosome".$chr1." - chromosome ".$chr2."\n";

for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
    if($i == $#non_zero_row_index)
    {
        print "\tChr".$chr1."_Chr".$chr2."_Row".$non_zero_row_index[$i].
              " : : [0,".$chr1_start."..".$chr1_stop."],\n";
    }
    else
    {
        print "\tChr".$chr1."_Chr".$chr2."_Row".$non_zero_row_index[$i].
              " : : [0,".$chr1_start."..".$chr1_stop."],\n";
    }
}
print "\n";


############################
## print frequency domains

## inter-interaction frequency domains
print "\t% define the Freq domains for the inter-interations\n";
print "\t% this is based on the Hi-C interaction matrix\n";

print "\n\t% chromosome".$chr1." - chromosome ".$chr2."\n";

for(my $i = 0; $i <= $#non_zero_row_index; $i++)
{
    my @domain;
    my @unique_values;

    for(my $col = $chr1_start; $col <= $chr1_stop; $col++)
    {
        ##check to see if it is non-zero
        if($frequencies[$non_zero_row_index[$i]][$col] != 0)
        {
            push @domain, $frequencies[$non_zero_row_index[$i]][$col];
        }
    }

    @unique_values = uniq @domain;
```

```
   print "\tChr".$chr1."_Chr".$chr2."_Freq".$non_zero_row_index[$i]."␣::␣[0";
   for(my $u = 0; $u <= $#unique_values; $u++)
   {
      print ",␣".$unique_values[$u];
   }
   print "],\n";

}
print "\n";

## print Row constraints
print "\t%␣apply␣constraints␣to␣the␣data␣to␣ensure␣they␣".
   "are␣valid␣2-tuple␣pairs␣based␣on␣the␣\n";
print "\t%␣biological␣data.␣note:␣(0,0)␣is␣when␣we␣choose␣".
   "to␣not␣select␣anything␣for␣that\n";
print "\t%␣Column␣row␣pair\n";

print "\n\t%␣INTER-INTERACTIONS\n";
print "\n\t%␣chromosome".$chr1."␣-␣chromosome␣".$chr2."\n";
my $vars = $chr1_start;
for(my $row = $chr2_start; $row <= $chr2_stop; $row++)
{
   ## if non-zero values exist
   if(defined $non_zero{$row})
   {
      print "\n\t%Column".$vars."␣Constraints\n";

      ## first print the non-zero options
      for my $nz_col (sort keys $non_zero{$row})
      {
         print "\t((Chr".$chr1."_Chr".$chr2."_Row".$row."␣#=␣".
            $non_zero{$row}{$nz_col}.")␣and␣(Chr".$chr1."_Chr".$chr2."_Freq".
            $row."␣#=␣".$frequencies[$row][$non_zero{$row}{$nz_col}."))␣or␣\n";

      }


      # print the "0" case
      print "\t((Chr".$chr1."_Chr".$chr2."_Freq".$row."␣#=␣0)␣and␣(Chr".
         $chr1."_Chr".$chr2."_Row".$row."␣#=␣0)),\n";
   }
}
print "\n";

#########################################################################
##    Print the additional constraints
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%␣␣␣␣Additional␣Constraints\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

print "\t%␣all␣the␣Output␣values␣will␣be␣different␣-␣"
   ."want␣them␣to␣correspond␣to␣which␣Rowumn\n";
print "\t%␣for␣each␣has␣a␣non-zero␣value,␣multiple␣zeros␣are␣allowed\n";

## for each chromosome
print "\talldifferent(Non_Zero_Rows),\n";
print "\tatmost(".($#non_zero_row_index-1).",␣Non_Zero_Rows,␣0),\n";

print "\n";


#########################################################################
##    Print the optimization
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%␣␣␣␣Optimize\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

print "\t%␣heuristics␣could␣be␣improved␣to␣make␣it␣faster\n";
print "\t%%␣␣maximize␣the␣interaction␣frequency␣from␣the␣tuple\n";
print "\tCost␣#=␣-sum(Freqs),\n";
#print "\tminimize(\n";
print "\t\tsearch(Freqs,␣0,␣input_order,␣indomain_max,␣bb_min(Cost),␣[]).\n";
#print "\t\tCost\n";
#print "\t).\n";

#########################################################################
```

```
##     Output the results
#########################################################################

print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
print "\t%%␣␣␣␣Output␣the␣results\n";
print "\t%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

print "\topen(File,␣'write',␣OUT),\n";
print "\t(foreach(X,Non_Zero_Rows),\n";
print "\t\tparam(OUT)␣do\n";
print "\t\t\tget_domain_as_list(X,␣DomList),\n";
print "\t\t\t\t(foreach(Y,DomList),\n";
print "\t\t\t\t\tparam(OUT)␣do\n";
print "\t\t\t\t\t\twrite(OUT,␣Y),\n";
print "\t\t\t\t\t\twrite(OUT,␣'␣')\n";
print "\t\t\t\t),\n";
print "\t\t\t\twrite(OUT,␣\"\\n\")\n";
print "\t),\n";
print "\tclose(OUT).\n";
```

Appendix D

Perl Programs for the Automation of Cytoscape Input Based on the Minimal, Non-Redundat CLP Program Results

D.1 generate_cytoscape_input_for_intra.pl

```
#!/usr/bin/perl
## will generate the input need for cytoscape analysis
## argument 1: the file containing the row output from eclipse
## argument 2: the file containing a list of non-zero variables
## Kimberly MacKay June 22, 2016

use strict;
use warnings;

## check to ensure four arguments were passed in
die "ERROR: must pass in four argumnets." if @ARGV != 4;

## grab the input arguments
my $clp_row_results_file = $ARGV[0];
my $clp_freq_results_file = $ARGV[1];
my $non_zero_vars_file = $ARGV[2];
my $chr = $ARGV[3];

#################################################
## parse the results file
#################################################
## open the row results
open ROW_RESULTS, "$clp_row_results_file"
   or die "ERROR: $clp_row_results_file could not be opened.";
chomp(my @clp_row_results = <ROW_RESULTS>);
close ROW_RESULTS;

## open the freq results
open FREQ_RESULTS, "$clp_freq_results_file"
   or die "ERROR: $clp_freq_results_file could not be opened.";
chomp(my @clp_freq_results = <FREQ_RESULTS>);
close FREQ_RESULTS;

## open the file with non-zero bins
open VARS, "$non_zero_vars_file"
   or die "ERROR: $non_zero_vars_file could not be opened.";
chomp(my @non_zero_vars = <VARS>);
close VARS;

my @node_names;
my @edge_names;

## sanity check to make sure the files are the same length
if($#clp_row_results == $#non_zero_vars
      && $#clp_row_results == $#clp_freq_results)
{
   for(my $i = 0; $i <= $#non_zero_vars; $i++)
   {
      ## get node1
      my $node1 = "bin".$non_zero_vars[$i];

      push @node_names, $node1;

      ## get node(s)2
      ## split the line from the clp file
      my @row_results =  split /\s+/, $clp_row_results[$i];

      my $noise = scalar @row_results;

      ## get the corresponding frequency
```

```
61        my $freq = $clp_freq_results[$i];

63        for(my $j = 0; $j <= $#row_results; $j++)
          {
65           if($row_results[$j] != 0)
             {
67              my $node2 = "bin".$row_results[$j];

69              push @node_names, $node2;

71              ## print the edge
                ## source   target   interaction_type
73                 edge_attr   source_attr target_attr
                print $node1."\t".$node2."\t"."intra"."\t"
75                 .($noise/$freq)."\t".$chr."\t".$chr."\n";
             }
77        }
       }
79  }
    else
81  {
       print "ERROR: files are not the same length and they should be.";
83  }
```

D.2 generate_cytoscape_input_for_inter.pl

```
1   #!/usr/bin/perl
    ## will generate the input need for cytoscape analysis
3   ## argument 1: the file containing the row output from eclipse
    ## argument 2: the file containing a list of non-zero variables
5   ## Kimberly MacKay June 22, 2016

7   use strict;
    use warnings;
9
    ## check to ensure nine arguments were passed in
11  die "ERROR: must pass in nine argumnets." if @ARGV != 9;

13  ## grab the command line arguments
    my $clp_row_results_file = $ARGV[0];
15  my $clp_freq_results_file = $ARGV[1];
    my $non_zero_vars_file = $ARGV[2];
17  my $start_node_1 = $ARGV[3];
    my $stop_node_1 = $ARGV[4];
19  my $start_node_2 = $ARGV[5];
    my $stop_node_2 = $ARGV[6];
21  my $chr1 = $ARGV[7];
    my $chr2 = $ARGV[8];
23
    ##################################################
25  ## parse the results file
    ##################################################
27  ## open the row results
    open ROW_RESULTS, "$clp_row_results_file"
29     or die "ERROR: $clp_row_results_file could not be opened.";
    chomp(my @clp_row_results = <ROW_RESULTS>);
31  close ROW_RESULTS;

33  ## open the freq results
    open FREQ_RESULTS, "$clp_freq_results_file"
35     or die "ERROR: $clp_freq_results_file could not be opened.";
    chomp(my @clp_freq_results = <FREQ_RESULTS>);
37  close FREQ_RESULTS;

39  ## open the file with non-zero bins
    open VARS, "$non_zero_vars_file"
41     or die "ERROR: $non_zero_vars_file could not be opened.";
    chomp(my @non_zero_vars = <VARS>);
43  close VARS;

45  ## sanity check to make sure the files are the same length
    if($#clp_row_results == $#non_zero_vars
47        && $#clp_row_results == $#clp_freq_results)
    {
49     for(my $i = 0; $i <= $#non_zero_vars; $i++)
       {
51        ## get node1
```

```
      my $node1 = "bin".$non_zero_vars[$i];

      ## get node(s)2
      ## split the line from the clp file
      my @row_results =  split /\s+/, $clp_row_results[$i];

      my $noise = scalar @row_results;

      ## get the corresponding frequency
      my $freq = $clp_freq_results[$i];

      for(my $j = 0; $j <= $#row_results; $j++)
      {
         if($row_results[$j] != 0)
         {
            my $node2 = "bin".$row_results[$j];

            if($non_zero_vars[$i] >= $start_node_1
               && $non_zero_vars[$i] <= $stop_node_1)
            {
               if($row_results[$j] >= $start_node_1
                  && $row_results[$j] <= $stop_node_1)
               {
                  ## print the edge
                  ## source   target   interaction_type
                     edge_attr   source_attr target_attr
                  print $node1."\t".$node2."\t"."inter"."\t"
                     .($noise/$freq)."\t".$chr1."\t".$chr1."\n";
               }
               else
               {
                  print $node1."\t".$node2."\t"."inter"."\t"
                     .($noise/$freq)."\t".$chr1."\t".$chr2."\n";
               }
            }
            else
            {
               if($row_results[$j] >= $start_node_1
                     && $row_results[$j] <= $stop_node_1)
               {
                  print $node1."\t".$node2."\t"."inter"."\t"
                     .($noise/$freq)."\t".$chr2."\t".$chr1."\n";
               }
               else
               {
                  print $node1."\t".$node2."\t"."inter"."\t"
                     .($noise/$freq)."\t".$chr2."\t".$chr2."\n";
               }
            }
         }
      }
   }
}
else
{
   print "ERROR: files are not the same length and they should be.";
}
```

D.3 generate_linear_cytoscape_input.pl

```
#!/usr/bin/perl
## generates the linear interactions needed for input into cytoscape visualization
## argument 1: the size of the matrix
## argument 2: the interaction matrix

## Kimberly MacKay July 4, 2016

use strict;
use warnings;
use List::MoreUtils 'uniq';

## check to ensure two arguments were passed in
die "ERROR: must pass in two argumnets." if @ARGV != 2;

my $num_variables = $ARGV[0];
my $HiC_file = $ARGV[1];
```

```
my $scale = 1000;
my $chr1_stop = 558;
my $chr2_stop = 1012;
my $chr3_stop = 1258;

##################################################
## parse the interaction matrix
##################################################
## open the interaction matrix file
open HIC, "$HiC_file" or die "ERROR: $HiC_file could not be opened.";
chomp(my @matrix_file = <HIC>);
close HIC;

my @frequencies;

## for each line after the header line
for(my $row = 1; $row <= $#matrix_file; $row++)
{
   ## split the line
   my @matrix_line = split /\t/, $matrix_file[$row];

   ## loop through the entire file to extract the frequencies
   ## note: we only have to extract one half of the matrix since it is symmetric
   ## along the diagonal
   for(my $col = 1; $col <= 1258; $col++)
   {
      ## adjusts NA's to 0's
      if($matrix_line[$col] =~ "NA")
      {
         $frequencies[$row][$col] = 0;
      }
      else
      {
         ## trying a smaller integer size to see if it improves speed
         $frequencies[$row][$col] = int($matrix_line[$col]*$scale);
      }
   }
}

##################################################
## make an array for the chromosomes and initialize it
##################################################
my @chrs;

for(my $i = 1; $i <= $chr3_stop; $i++)
{
   if($i <= $chr1_stop)
   {
      $chrs[$i] = 1;
   }
   elsif($i <= $chr2_stop)
   {
      $chrs[$i] = 2;
   }
   else
   {
      $chrs[$i] = 3;
   }
}

##################################################
## print out the linear interactions and their
## "distances" according to my frequency value
##################################################

for(my $row = 1; $row <= $#frequencies; $row++)
{
   my $col = $row;

   if($row != $chr1_stop && $row != $chr2_stop && $row != $chr3_stop)
   {
      if($frequencies[$row][$col] != 0)
      {
         print "bin".$row."\tbin".($row+1)."\tlinear\t"
            .(1/$frequencies[$row][$col])."\t".$chrs[$row]."\t"
            .$chrs[($row+1)]."\n";
      }
      else
      {
         print "bin".$row."\tbin".($row+1)."\tlinear\t"
            .(0.001)."\t".$chrs[$row]."\t".$chrs[($row+1)]."\n";
```

```
        }
100     }
}
```

Appendix E

Example Command Line Input for the Minimal CLP Program

```
## Start ECLiPSe
kam945@nugget:~$ eclipse -g 16G.
ECLiPSe Constraint Logic Programming System [kernel]
Kernel and basic libraries copyright Cisco Systems, Inc.
and subject to the Cisco-style Mozilla Public Licence 1.1
(see legal/cmpl.txt or http://eclipseclp.org/licence)
Source available at www.sourceforge.org/projects/eclipse-clp
GMP library copyright Free Software Foundation, see legal/lgpl.txt
For other libraries see their individual copyright notices
Version 6.1 #220 (x86_64_linux), Thu Apr 21 22:40 2016

## Compile the CLP Program
[eclipse 1]: ['chr3_reduced_domain.ecl'].
source_processor.eco loaded in 0.01 seconds
hash.eco   loaded in 0.01 seconds
compiler_common.eco loaded in 0.02 seconds
compiler_normalise.eco loaded in 0.01 seconds
compiler_map.eco loaded in 0.01 seconds
compiler_analysis.eco loaded in 0.01 seconds
compiler_peephole.eco loaded in 0.01 seconds
compiler_codegen.eco loaded in 0.02 seconds
compiler_varclass.eco loaded in 0.01 seconds
compiler_indexing.eco loaded in 0.01 seconds
compiler_regassign.eco loaded in 0.01 seconds
asm.eco    loaded in 0.02 seconds
module_options.eco loaded in 0.01 seconds
ecl_compiler.eco loaded in 0.14 seconds
lists.eco  loaded in 0.01 seconds
constraint_pools.eco loaded in 0.00 seconds
Loaded Gecode solver 4.4.0
/mnt2/birl/sw/linux/eclipse/lib/gfd.ecl compiled 614984 bytes in 0.77 seconds
chr3_reduced_domain.ecl compiled 3343280 bytes in 1.97 seconds

Yes (2.11s cpu)

## Query the CLP program
[eclipse 2]: maximize("row_output.txt", "freq_output.txt", R).
Found a solution with cost -9612

R = [1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028,
1029, 1030, 1031, 1032, 1033, 1034, ...]

Delayed goals:
   gfd : gfd_do_propagate(gfd_prob(nvars(467)))

Yes (0.53s cpu)
```

Figure E.1: An example of the command line input and output for running the CLP program which will determine the optimal solution for the chromosome 3 intra-interaction sub-problem.

Appendix F

Synthetic Datasets Used for Initial Testing and Development

F.1 5×5 Matrix

```
bin1 bin2 bin3 bin4 bin5
bin1 1 2 3 4 5
bin2 2 3 4 5 1
bin3 3 4 5 1 2
bin4 4 5 1 2 3
bin5 5 1 2 3 4
```

F.2 10×10 Matrix

```
bin1 bin2 bin3 bin4 bin5 bin6 bin7 bin8 bin9 bin10
bin1 1 2 3 4 5 6 7 8 9 10
bin2 2 3 4 5 6 7 8 9 10 1
bin3 3 4 5 6 7 8 9 10 1 2
bin4 4 5 6 7 8 9 10 1 2 3
bin5 5 6 7 8 9 10 1 2 3 4
bin6 6 7 8 9 10 1 2 3 4 5
bin7 7 8 9 10 1 2 3 4 5 6
bin8 8 9 10 1 2 3 4 5 6 7
bin9 9 10 1 2 3 4 5 6 7 8
bin10 10 1 2 3 4 5 6 7 8 9
```

F.3 15×15 Matrix

```
bin1 bin2 bin3 bin4 bin5 bin6 bin7 bin8 bin9 bin10 bin11 bin12 bin13 bin14 bin15
bin1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
bin2 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1
bin3 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2
bin4 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3
bin5 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4
bin6 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5
bin7 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6
bin8 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7
bin9 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8
bin10 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9
bin11 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10
bin12 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11
bin13 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12
bin14 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13
bin15 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

F.4 20×20 Matrix

```
bin1 bin2 bin3 bin4 bin5 bin6 bin7 bin8 bin9 bin10 bin11 bin12 bin13 bin14 bin15 bin16 bin17
bin18 bin19 bin20
bin1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
bin2 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1
bin3 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2
bin4 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3
bin5 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4
bin6 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5
bin7 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6
bin8 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7
bin9 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8
bin10 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9
bin11 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10
bin12 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11
bin13 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12
bin14 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13
bin15 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14
bin16 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
bin17 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
bin18 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
bin19 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
bin20 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

F.5 22×22 Matrix

```
bin1 bin2 bin3 bin4 bin5 bin6 bin7 bin8 bin9 bin10 bin11 bin12 bin13 bin14 bin15 bin16 bin17
bin18 bin19 bin20 bin21 bin22
bin1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
bin2 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1
bin3 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2
bin4 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3
bin5 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4
bin6 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5
bin7 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6
bin8 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7
bin9 9 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8
bin10 10 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9
bin11 11 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10
bin12 12 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11
bin13 13 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12
bin14 14 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13
bin15 15 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14
bin16 16 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
bin17 17 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
bin18 18 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
bin19 19 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
bin20 20 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
bin21 21 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
bin22 22 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
```

F.6 25×25 Matrix

```
bin1 bin2 bin3 bin4 bin5 bin6 bin7 bin8 bin9 bin10 bin11 bin12 bin13 bin14 bin15 bin16 bin17
bin18 bin19 bin20 bin21 bin22 bin23 bin24 bin25
bin1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
bin2 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1
bin3 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2
bin4 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3
bin5 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4
bin6 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5
bin7 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6
bin8 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7
bin9 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8
bin10 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9
bin11 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10
bin12 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11
bin13 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12
bin14 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13
bin15 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14
bin16 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
bin17 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
bin18 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
bin19 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
bin20 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
bin21 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
bin22 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
bin23 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
bin24 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
bin25 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

F.7 30×30 Matrix

```
bin1 bin2 bin3 bin4 bin5 bin6 bin7 bin8 bin9 bin10 bin11 bin12 bin13 bin14 bin15 bin16 bin17
bin18 bin19 bin20 bin21 bin22 bin23 bin24 bin25 bin26 bin27 bin28 bin29 bin30
bin1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
bin2 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1
bin3 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2
bin4 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3
bin5 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4
bin6 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5
bin7 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6
bin8 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7
bin9 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8
bin10 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9
bin11 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10
bin12 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11
bin13 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12
bin14 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13
bin15 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14
bin16 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
bin17 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
bin18 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
bin19 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
bin20 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

```
bin21 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
bin22 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
bin23 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
bin24 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
bin25 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
bin26 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
bin27 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
bin28 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
bin29 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
bin30 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
```