The Use of Maximum Edge Weight Clique and Markov Random Field
Problem Formulation to Discover Motifs

by

Alexandra Kravchuk

A thesis submitted to the Faculty of Graduate and Postdoctoral
Affairs in partial fulfillment of the requirements
for the degree of

Master of Science

in

Mathematics and Statistics with a Specialization in Bioinformatics

Carleton University
Ottawa, Ontario

©2021

Alexandra Kravchuk

Abstract

In this thesis, we attempt to find approximately repeating patterns in DNA called motifs by modeling the problem both as a maximum edge weight clique problem (MEWCP) and as a maximum a posteriori inference on a Markov random field (MAP-MRF). Though we simplify this motif discovery problem as an ungapped sequence alignment with a known motif length, we make up for this simplification by generating multiple solutions. We implement three different approaches: an integer linear programming formulation approach and a branch-and-bound approach for the MEWCP model, and a loopy belief propagation approach for the MAP-MRF model. We test our approaches by trying to find transcriptional regulators in the binding sites of *Escherichia coli* sequence data. Through motif discovery, we also make a connection between the MEWCP and MAP-MRF, and thus make a connection between any approach used to solve one graph model to any approach used to solve the other graph model.

Acknowledgements

I would like to thank my supervisor Kevin Cheung for his immense support, understanding, and mentorship. His teaching, guidance, and our thoughtful discussions were essential to the progress that I made as a student, and as an individual.

I would also like to thank Stéphane Aris-Brosou for his engaging classes and for his continued support in my academic career.

I would like to thank Kevin Cheung, Stéphane Aris-Brosou, Brett Stevens, and Alex Wong for serving on my thesis committee, and thank Elena Zaslavsky and Mona Singh for their correspondence regarding their article, *A combinatorial optimization approach for diverse motif finding applications*.

Finally, I would like to thank my friend Dave Rogers for his advice and encouragement, as well as his helpful suggestions in writing this thesis.

Contents

1	Introduction	1
1.1	Motif Discovery	1
1.2	Previous Motif Finders and Modeling the Motif Problem	3
2	Technical Preliminaries	7
2.1	Overview of Graphs	7
2.2	Overview of MEWCP	8
2.3	Overview of MAP-MRF	9
2.4	Overview of Optimization	11
2.4.1	LP Description	12
2.4.2	Branch-and-Bound Description	14
2.5	Overview of Loopy Belief Propagation	15
3	Models and Solution Methods	19
3.1	Connections between the MEWCP and MAP-MRF	19
3.1.1	Connection between Belief Propagation and ILP Formulation	19
3.1.2	Re-formulating between the MEWCP and the MAP-MRF	21
3.2	Modeling the Motif Discovery Problem as a MEWCP	21
3.2.1	Setup of the Motif Discovery as a MEWCP	21
3.2.2	The Approach Using ILP Formulation	23
3.2.3	The Approach Using Branch-and-Bound	27

3.3	Modeling the Motif Discovery Problem as a MAP-MRF	29
3.3.1	Setup of the Motif Discovery as a MAP-MRF	29
3.3.2	The Approach Using Loopy Belief Propagation	30
4	Results	36
4.1	Dataset Used and Machine Specifications	36
4.2	Survey of Results	36
4.3	Discussion	41
4.4	Limitations	43
5	Conclusion	44
5.1	Findings	44
5.2	Future endeavors	45

List of Tables

4.1	Motif discovery results for the four binding sites associated with the transcriptional regulator of <i>cspA</i> from our <i>E. coli</i> dataset. We compare between the ILP formulation using Gurobi, the ILP formulation using Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation.	38
4.2	Motif discovery results for the four binding sites associated with the transcriptional regulator of <i>hipB</i> from our <i>E. coli</i> dataset. We compare between the ILP formulation using Gurobi, the ILP formulation using Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation.	39
4.3	Motif discovery results for the four binding sites associated with the transcriptional regulator of <i>oxyR</i> from our <i>E. coli</i> dataset. We compare between the ILP formulation using Gurobi, the ILP formulation using Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation.	41

List of Figures

1.1	A DNA sequence alignment in FASTA format. The header, beginning with a “>” indicates the description of the sequence, such as its name and location, and the following string of nucleotide bases A, C, G, and T, is the sequence itself. The colored string segment in each of these sequence strings indicates a motif, where the red bases indicate an exactly matching base between all three motifs, and the blue indicates a slight variation in bases (from left to right, the blue strings show an example of a reversal mutation, a substitution mutation, and an indel mutation).	3
1.2	All mutation examples, including indels, substitutions, and reversals.	4
2.1	Graph examples.	8
2.2	More graph examples.	9
2.3	Representing a system of random variables using graphical models.	12
2.4	An example of message passing for the belief propagation performed in order to find information about the likely state of a particular random variable x_i . The example described in the text is represented in red messages, and all other messages are represented in blue.	18
2.5	An example of a suboptimal state found when calculating the beliefs for a MAP inference on a cyclic graph. If only the states with the highest scores possible are chosen for each variable, then the chosen states, x_2, y_2, z_0 with an overall score of $30 + 1 + 1 = 32$, are not the optimal system states, since the “true” optimal system states are x_2, y_2, z_2 with a score of $30 + 3 + 3 = 36$	18

3.1	This is an illustration of the claim that the MEWCP and MAP-MRF are connected. The red arrows indicate the possible approaches that can be taken to solve each respective problem, where the solid arrows represent the approaches studied by us, and the dashed arrows indicate existing approaches not studied by us. We attempt to make this claim in order to show that any single one of these approaches can be used in substitution for any other approach. The blue arrows indicate previous connecting research.	20
3.2	An example of going from a MAP-MRF formulation to a MEWCP formulation. Note that each variable becomes a vertex set, and each state of a variable of the MAP-MRF becomes its own vertex in the MEWCP. Note also that when a variable in the MAP-MRF shares an edge with another variable, then the equivalent two sets of vertices in the MEWCP are fully connected.	22
3.3	An example of going from a MEWCP formulation to a MAP-MRF formulation. We start with a graph, and we try to color it such that no two vertices with the same color share an edge (note that we do not need a minimum coloring to be able to break down a graph into a multipartite graph, but it is preferable so that we can minimize the number of random variables at the end). The colors indicate to us the disjoint vertex sets of our multipartite graph. We use a random variable to represent each disjoint set, and we represent each vertex from a set as a state of that random variable.	22
3.4	A small example of how the motif discovery problem is represented using a MEWCP. Each sequence is broken down into 3-long segments, each vertex set represents a sequence, and each vertex represents an l-long segment from that sequence. The solution to this problem is shown by coloring the vertices and bolding their shared edges. In this case, the “GAC” would be our motif.	24

3.5	A small example of a MEWCP being solved for a 2-partite graph using the ILP formulation approach. The red vertices and edge outline the maximum edge weight clique of the graph. The solution to the ILP is shown at the bottom right, where the decision variables x_2 and x_3 equal to 1 and the decision variable y_{23} is equal to 1, meaning that the vertices 2 and 3 and the edge between them are chosen for the maximum edge weight clique.	26
3.6	A small example of a MEWCP being solved for a 2-partite graph using the branch-and-bound algorithm approach. The red vertices and edge outline the maximum edge weight clique of the graph. The algorithm starts with the subproblem $P(\emptyset, \{0, 1, 2, 3, 4\})$ with a lower bound of $W(F) = 0$, and it finds an ordered set of vertices, Π , and the corresponding upper bounds of each new clique formed from each of those vertices. The algorithm then looks through $P(\{3\}, \{0, 1, 2\})$ and finds another ordered set of vertices Π with upper bounds. The branching ends at $P(\{2, 3\}, \emptyset)$ when there are no more vertices that can be added. The algorithm does not investigate $P(\{0, 3\}, \emptyset)$ and $P(\{1, 3\}, \emptyset)$ because they have upper bounds of $W(F) = 1$, which is lower than the lower bound of $W(F) = 10$ for the subproblem $P(\{3\}, \{0, 1, 2\})$. The algorithm returns to the top, but it does not investigate $P(\{4\}, \{0, 1, 2\})$ because the subproblem has an upper bound of $W(F) = 1$, which is lower than the new lower bound of $W(F) = 10$ (found previously). This means that $\{2, 3\}$ are the vertices belonging to our maximum edge weight clique.	30
3.7	The unmodified Algorithm 1 pseudocode taken from Shimizu et al. (2020).	31
3.8	The modified Algorithm 2 pseudocode taken from Shimizu et al. (2020) with our modifications shown in red.	32
3.9	The modified Algorithm 3 pseudocode taken from Shimizu et al. (2020) with our modifications shown in red.	32

3.10	A small example of how the motif discovery problem is represented using a MAP-MRF. Each sequence is broken down into 3-long segments, each vertex represents a sequence, and the state of each vertex represents an l-long segment from that sequence. The solution to this problem is shown by bolding our motif, “GAC”. . . .	33
3.11	A small example of a MAP-MRF being solved for a system of random variables using the loopy belief propagation approach. The orange squares outline the most likely states of the random variables of the system (note that the selection of $x_0 = s_2$ and $x_1 = s_3$ leads to the highest score possible). Messages are chosen to pass twice to x_0 to compute the max marginals of x_0 , and messages are chosen to pass twice to x_1 to compute the max marginals of x_1 . We can see that for x_0 , s_2 has the highest value for the max marginals, and for x_1 , s_3 has the highest value for the max marginals.	35
4.1	Motif discovery results for the four binding sites associated with the transcriptional regulator of <i>cspA</i> from our <i>E. coli</i> dataset. Here the optimal solution found by our exact motif finders and our heuristic motif finder is the string segment in red. Our transcriptional regulator is the string segment in blue.	37
4.2	Motif discovery results for the four binding sites associated with the transcriptional regulator of <i>hipB</i> from our <i>E. coli</i> dataset. Here the optimal solution found by our exact motif finders and our heuristic motif finder is the string segment in red. Our transcriptional regulator is the string segment in blue. The string segment in purple is the overlap between our optimal solution and the transcriptional regulator. . . .	40
4.3	Motif discovery results for the four binding sites associated with the transcriptional regulator of <i>oxyR</i> from our <i>E. coli</i> dataset. Here the optimal solution found by our exact motif finders is the string segment in red. Our transcriptional regulator is the string segment in blue. The string segment in purple is the overlap between our optimal solution and the transcriptional regulator.	40

Chapter 1

Introduction

1.1 Motif Discovery

In the post-genomic era that followed the completion of the Human Genome Project, there has been a widespread availability of genetic data in the form of either long sequences of nucleotides found within DNA or long sequences of amino acids found within proteins (Manning et al., 2013). Looking for approximately repeating patterns, or *motifs*, within this genetic data (and particularly looking for the motifs between different sequences) is of interest in the field of bioinformatics because this data gives us insights into the evolutionary relationships among organisms and insights into the functions of genes (Saito et al., 2007), both of which give us many practical applications in an endless number of fields, such as forensics, medicine, technology, and so much more (Zambelli et al., 2012). Being able to gain the biological profile of an individual and personalizing treatment, for example, may revolutionize disease detection, prevention, and treatment (Kahvejian et al., 2008).

DNA is made up of a sequence of nucleotides, which are defined by their sugar (deoxyribose), phosphate groups, and nitrogenous bases (represented by the letters **A**, **C**, **G**, or **T** and often shortened to *base*) and proteins are made up of a sequence of amino acids (represented by one of 20 different letters). By comparing the sequence of nitrogenous bases or sequence of amino acids among different or similar species, it is possible to find various motifs (Deonier et al., 2005). One such

sequence comparison is called a *sequence alignment*, in which segments of a DNA nitrogenous base sequence or a protein amino acid sequence are arranged together in the hopes of discovering a common motif (Deonier et al., 2005). Figure 1.1 depicts one such sequence alignment in which there is a shared motif.

Before applying any computational methods to search through the sequence data, however, that sequence data must first be collected in a lab and converted into a readable text file that contains the description of each sequence collected, as well as the sequence itself. When DNA is collected, or *sequenced*, it is done using one of the many existing methods that vary on the type of sequence being explored: there are large-scale sequencing methods (Poinar et al., 2006), high throughput methods (Meyer and Kircher, 2010), and many new, more precise methods still being developed (Schloss et al., 2020). These sequences are often cleaned up and converted into sequence text files known as GenBank, EMBL, or *FASTA* files (Gilbert, 2003). Figure 1.1 in particular is a sequence alignment written in FASTA format.

In sequence alignments, there are mutations that are assumed to occur in DNA and proteins of divergent species. There are four types of mutations, called *insertion*, *deletion*, *substitutions*, and *reversals* (Deonier et al., 2005). Insertion and deletion are mutations in which a *residue* (one of the letters of the sequence of either DNA or proteins, which could also be called a base if it is a DNA residue, or an amino acid if it is a protein residue) is added or removed respectively to a sequence. They tend to be grouped together as an *indel* mutation when comparing sequences, since there is no way to be sure if one sequence inserted a residue or if the other deleted a residue (Deonier et al., 2005). Substitutions are mutations in which one residue from a sequence is replaced with a different residue, leading to a mismatch of residues between that sequence and other sequences. When performing the sequence alignment, *gaps* (marked as “-”) are placed between residues in the sequences being studied, and the placement of gaps optimizes the number of identical characters between sequences (Deonier et al., 2005). Reversals are mutations in which the order of a collection of residues is reversed. Figure 1.2 shows all examples of mutations that may occur.

Because of the noise caused by the mutations of sequences that may affect the motifs, and

because the length of the sequences and number of sequences in an alignment can be quite large, finding the motifs, or rather solving the *motif discovery problem*, becomes a computational challenge.

```
>Sequence 01
ACAAAGCGGGACCAAAGCCATGACAAAAACGCGTAACAAAAGTGTCTATAATCACGGCAG
AAAAGTCCACATTGATTATTTGCACGGCGTCACACTTTGCTATGCCCTAGCATTTTTATC
CATAAGATTAGCGGATCCTACCTGACGCTTTTATCGCAACTCTCTACTGTTTCTCCATA
CCCGTTTTTTGGATGGAGTGAAACGATGGCGATTGCAATTGGCCTCGATTTTGGCAGTG
>Sequence 02
TGCCGTCACGTGCGTCTTTTACTGGCTCTTCTCGCTAACCCAACCGGTAACCCCGCTTATT
AAAAGCATTCTGTAACAAAGCGGGACCAAAGCCATGACAAAAACGCGTAACAAAAGTGTG
TATAATCACGGCAGAAAAGTCCATTACGATTAAAGCGCACGGCGTCA---TTGCTATGCC
ATAGCATTTTTATCCATAAGATTAGCGGATCCTACCTGACGCTTTTATCGCAACTCTCT
>Sequence 03
AATGAACCTTTATGAATTTTATCTGCTGTAAAATTAGGTGGTTAATAATAATCTCAATAAT
TCAACTTAATTTGAAAATTGGAATATCCATCACATAACGACATGTGCGAGCAATTTAATC
CATATTTATGCTGTTTCCGACCTGACACCTGCGTGAGTTGGTCCACATTGATTACAGCA
CGGCGTCACACTTTGCTATGCCGGAAAAAATGGTTACTATCAATACGGAATCTGCTTTAA
```

Figure 1.1: A DNA sequence alignment in FASTA format. The header, beginning with a “>” indicates the description of the sequence, such as its name and location, and the following string of nucleotide bases A, C, G, and T, is the sequence itself. The colored string segment in each of these sequence strings indicates a motif, where the red bases indicate an exactly matching base between all three motifs, and the blue indicates a slight variation in bases (from left to right, the blue strings show an example of a reversal mutation, a substitution mutation, and an indel mutation).

1.2 Previous Motif Finders and Modeling the Motif Problem

In the past, there have been two popular approaches to solving motif discovery problems: the enumerative approach and the probabilistic approach (Hashim et al., 2019). Enumerative approaches (such as Weeder (Pavesi et al., 2004) and DREME (Bailey, 2011)) are exhaustive, and thus it is possible to find a *global optimal solution* (a solution with no better solutions existing) to the motif discovery problem. Though it appears to be ideal, it is not useful for non-exact motifs and it is very computationally demanding, making it useful in searching only for shorter motifs (like the ones in eukaryotic DNA sequences) (Zaslavsky and Singh, 2006). On the other hand, the probabilistic approaches (such as MEME (Bailey et al., 2015), AlignACE (Chen et al., 2008) and Gibbs Motif Sampler (Thijs et al., 2001; Thompson et al., 2003)), which use position-specific weight matrices, are less computationally demanding and thus perfect for searching for longer motifs (like the ones found in prokaryotic DNA sequences) (Zaslavsky and Singh, 2006). How-

If the motif discovery problem is treated as a multiple alignment problem in which the motifs are not gapped and the lengths of the motifs are known, then it is possible to model the problem in two intuitive ways: either as a maximum edge weight clique problem, or as a maximum a posteriori inference on a Markov random field. Despite this over-simplification of the motif discovery, handling the problem this way has been shown to out-perform known methods such as Weeder, MEME, and Gibbs Motif Sampler when attempting to find longer motifs that are common to bacterial datasets (Yanover et al., 2009; Zaslavsky and Singh, 2006). Furthermore, if we attempt to find a collection of multiple solutions to the motif discovery problem, then we have a higher chance of finding the “true” motif within our group of solutions. This means that we will treat the desire for multiple solutions as an important part of our motif discovery problem.

For the maximum edge weight clique model setup described in Section 3.2.1, we consider two known exact approaches: the integer linear programming formulation which is outlined in Section 3.2.2, and the branch-and-bound algorithm which is outlined in Section 3.2.3. However, approaches for solving the maximum edge weight clique are not limited to the integer linear programming formulation and the branch-and-bound algorithm, and there is still ongoing research for development of new or improved approaches. This means that establishing a firmer connection between motif discovery problems and maximum edge weight clique problems can open up a large number of algorithms and tools that could be used for motif discovery. In Section 3.3.1 we explore the maximum a posteriori inference on a Markov random field, and consider an approach, loopy belief propagation, that is an adjustment of a common heuristic and outlined in Section 3.3.2. For each of these approaches, we consider the output of multiple solutions to increase our chance of finding the “true” motif. The results for all three of these approaches can be found in Section 4.2.

In Section 3.1 we go even further and discuss the connection between the maximum edge weight clique problem and the maximum a posteriori inference on a Markov random field, which also opens up the possibility of using any approach developed for one model as an approach for the other model. Inspiration for this research was motivated by Zaslavsky and Singh (2006) and their use of a linear programming formulation approach to solve a maximum edge weight clique

problem modeled from the motif discovery problem.

Chapter 2

Technical Preliminaries

2.1 Overview of Graphs

Graphs are mathematical models that are made up of objects, which are visually depicted by circles called *vertices*, and relations between those objects, visually depicted as lines between the circles called *edges* which may have value called the *weight* associated with them (Chartrand, 1977). This simple depiction makes them suitable for modeling many different types of problems. A *multipartite graph*, or an *N -partite graph*, is a graph in which each vertex of the graph belongs to one of N disjoint sets, and in which no two vertices that belong to the same set are connected via an edge. An example of a multipartite graph is shown in Figure 2.1a and Figure 2.1c. A *clique* of a graph is a subset of vertices that are all connected to each other, examples of which are shown in Figure 2.1a, Figure 2.1b, and Figure 2.1c. The *maximum edge weight clique* of a graph is the clique that has the highest sum of edge weights. Although finding the maximum edge weight clique of a graph is considered to be an NP-hard problem (Shimizu et al., 2020), there exist many algorithms that are able to find the maximum edge weight cliques with surprising accuracy.

A graph is said to be *complete* when all of the vertices are connected by edges, shown in Figure 2.1b, and a multipartite graph is said to be complete when all of the vertices of one set are connected to all of the vertices of all of the other sets, shown in Figure 2.1c. Graphs can be *cyclic* or *acyclic*, meaning that the graph either has at least one *cycle*, a sequence of connected

un-repeating edges in which the first vertex and last vertex are the only repeated vertices, or no cycles, respectively. Acyclic graphs are otherwise known as *forest* graphs, or more specifically *tree* graphs if all of the vertices are connected through a collection of edges. An example of each graph is shown in Figure 2.2a and Figure 2.2b. Tree graphs are thought to have parent vertices that branch to child vertices, and rooted trees have a *root* vertex that has no parents, and *leaf* vertices that have no children, shown in Figure 2.2b. When the edges have a direction, then they are known as *directed edges* and the graph becomes a *directed graph*. Figure 2.2b shows an example of a directed graph.

(a) A 2-partite graph. Note that the vertices of each set do not connect to vertices of the same set. The red vertices and edge outline one clique from the graph.
 (b) A complete graph. Note that any collection of vertices forms a clique. The red vertices and edges outline one clique from the graph.
 (c) A complete 3-partite graph. Note that if one vertex is chosen from each set, that collection of vertices will form a clique. The red vertices and edges outline one clique from the graph.

Figure 2.1: Graph examples.

2.2 Overview of MEWCP

The maximum edge weight clique problem, which will subsequently be referred to as the *MEWCP*, is the problem of finding a selection of vertices and edges of a graph clique to obtain the highest weighted clique, and can be solved using a large number of existing algorithms with new algorithms still being developed. One strategy for finding an optimal solution of the MEWCP involves treating the problem as an optimization problem and solving it by using either *integer*

Figure 2.2: More graph examples.

linear programming formulation and *integer linear programming* solvers (Zaslavsky and Singh, 2006), or by using a *branch-and-bound algorithm* (Shimizu et al., 2020).

2.3 Overview of MAP-MRF

When it is desirable to predict the overall state of a *system of random variables* (a system of variables in which each variable has the possibility of being one of several possible values, or *states*), it is common to either use a *Bayesian network* or a *Markov random field* to model the system (chapter 4 of Koller and Friedman (2009))¹. There are benefits and drawbacks to both, but ultimately the selection of one representation over another depends on the type of graph that is best suited for modeling the state of a system. When using a graph to model a system of random variables, the random variables are represented by vertices, and the dependencies between those random variables are represented either by edges (implying an interdependency between the state of one variable and another) or by directed edges (implying a causation of the state of one variable state on the state of another), where the state of the variables will influence the weights of their shared edges (chapter 4 of Koller and Friedman (2009)).

¹A very helpful collection of notes based on this textbook could be found here: <https://ermongroup.github.io/cs228-notes/representation/undirected/>

If all dependencies between the variables can be written as direct causations, then the state of a system is best represented as a directed acyclic graph, making it desirable to use a Bayesian network to represent the system. However, if the dependencies between the variables are complicated, unknown, or involve properties that cannot be represented using only direct causations (such as interdependencies between variables), then the state of the system is best represented as a undirected cyclic graph, making it desirable to use a Markov random field to represent the system. An example of a Bayesian network is outlined in Figure 2.3a and an example of a Markov random field is outlined in Figure 2.3b. It is often best to choose a Bayesian network over a Markov random field whenever possible, since Bayesian networks are easier to interpret and more computationally manageable, but in situations where it is not possible to create a directed acyclic graph representation, the Markov random field is a very useful formulation (chapter 4 of Koller and Friedman (2009)).

When a system of random variables has complicated interdependencies and a Markov random field is chosen to represent the system, the probability of the system is defined using a score of the variables:

$$\tilde{P}(S) = \phi(s_1, s_2)\phi(s_1, s_3) \dots \phi(s_2, s_3) \dots \phi(s_i, s_j) \dots \phi(s_{n-1}, s_n)$$

where $S = (s_1, s_2, \dots, s_n)$ represents the overall state of the system of random variables in which each random variable $x_i = s_i$ where s_i is a state of x_i , and $\phi(s_i, s_j)$ is the relationship weight of the state s_i of x_i and state s_j of x_j . The probability of a system is represented as:

$$P(S) = \frac{1}{Z} \tilde{P}(S)$$

where Z is the normalization factor, $Z = \sum_{s_1, s_2, \dots, s_n} \tilde{P}(S)$. A further generalization of the probability can be written as:

$$P(S) = \frac{1}{Z} \prod_{c \in C} \phi_c(s_c)$$

where $\phi_c(s_c) = \phi(s_i, s_j, \dots)$ can represent a relationship weight between multiple states of variables (in which these variables are variables within single clique, C , and have corresponding states

$s_i, s_j, \dots$), and Z is the normalization factor $Z = \sum_{s_1, s_2, \dots, s_n} \prod_{c \in C} \phi_c(s_c)$.

For any system of random variables, it may be of interest to calculate one of two things: either the *marginal inference* of one the variables (the probability of some x_i having a state of s_i), or the *maximum a posteriori*, otherwise known as the *MAP* (the most likely state assignment S , or the configuration S that gives a maximum $P(S)$). The MAP can be written out as

$$\max_{S=(s_1, s_2, \dots, s_n)} P(S)$$

in which we are given some sort of prior data that will help with the prediction of S . Finding the MAP can also be written as

$$\max_{S=(s_1, s_2, \dots, s_n)} \log(P(S))$$

since taking the log will not change the maximality of a configuration of S , and therefore the MAP can be written as

$$\max_{S=(s_1, s_2, \dots, s_n)} \sum_{c \in C} \theta_c(s_c) - \log(Z)$$

where $\theta_c(s_c) = \log(\phi_c(s_c))$ or

$$\max_{S=(s_1, s_2, \dots, s_n)} \sum_{c \in C} \theta_c(s_c)$$

since $\log(Z)$ is a normalizing constant that also does not affect the maximality of a configuration of S . We will refer to the *MAP inference on the Markov random field* as our *MAP-MRF*. Often either *belief propagation* or *Markov chain Monte Carlo* are used in order to perform a calculation relating to either the marginal inference on the MRF or the MAP-MRF, and both algorithms can find a “good” approximate solution with a reasonable runtime (chapter 4 of Koller and Friedman (2009)).

2.4 Overview of Optimization

The formulation of an optimization problem involves establishing a function of variables called the *objective function* that is either minimized or maximized to find the desired optimal solution (Rader, 2010), and establishing a space of possible variable solutions, called either the *search*

(a) A Bayesian network that shows how a forest fire might happen. The likelihood of low atmospheric pressure can influence the likelihood of the weather either being sunny or stormy. This in turn influences the likelihood of people going camping. A forest fire could be caused either by people camping and making campfires, or by a storm, so the likelihood of a storm and the likelihood of people camping both influence the likelihood of a forest fire.

(b) A Markov random field that shows whether individuals from a friend group will decide to go to the movies. In this case, Sarah might be good friends with Steve and Carrie, but she does not know Harry very well, so Harry's decision to attend the movies will not directly influence Sarah's decision, and vice versa. However, if Harry goes to the movies, then Steve and Carrie will be more likely to go, which in turn will make Sarah more likely to go.

Figure 2.3: Representing a system of random variables using graphical models.

space or the space of *feasible* solutions (Gusfield, 2019; Morrison et al., 2016). The optimal solution may refer to either the value of the objective function when it is optimized, or the variable values for which the objective function reaches its optimized value, so we refer to the former as our *optimal value* and the later as our *optimal variable values*. In an optimization problem, there is no difference between minimizing or maximizing the optimal solution, since a minimization problem can be converted to an equivalent maximization problem by maximizing the negation of the objective to be minimized (Gusfield, 2019).

2.4.1 LP Description

In the case of a linear programming formulation, or an *LP formulation*, we add what are called *linear constraints* to our optimization problem, which are equations that limit the choice of variables for our objective function (Gusfield, 2019). These linear constraints can both specify the

possible values of each of the variables, as well as relate the variables (Rader, 2010). The LP may report three possible outcomes: if there is no feasible solution, then the LP will report that there is no solution; if there exists a set of feasible solutions, then the LP will report one optimal solution and one set of optimal variable values (even though there may exist another set of optimal variable values that optimize the objective function); and if there exists a set of feasible solutions but no bound on those solutions, then the LP reports the unboundedness of the problem (Gusfield, 2019).

One advantage of using LP is that there are solvers that can solve large instances very quickly and it is often much faster to implement than it is to find and implement a polynomial-time algorithm for a problem (Gusfield, 2019). Another benefit to using LPs is that LPs work for both minimization and maximization problems, which eliminates the need for a separate algorithm to either maximize or minimize, respectively (Gusfield, 2019). At practically no cost to the optimality of the solution, LP formulation is helpful in obtaining a solution quickly and easily.

A small example of an LP would be the following:

$$\begin{aligned} \max \quad & 3x_1 + 4x_2 \\ \text{s.t.} \quad & x_1 + 3x_2 \geq 2 \\ & 3x_1 + 2x_2 = 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

In this particular example, we attempt to maximize our objective value, $3x_1 + 4x_2$, in which the variables, x_1 and x_2 , are subject to (s.t.) the linear constraints $x_1 + 3x_2 \geq 2$ and $-3x_1 + 2x_2 = 5$. The two variables are also subject to the linear constraint $x_1, x_2 \geq 0$, which indicates to us that x_1 and x_2 must both be non-negative real numbers. In this particular case, our optimal value is 10, and our optimal variable values are $x_1 = 0$ and $x_2 = \frac{5}{2}$.

A computer software program for solving LP problems is called an *LP solver*. An LP solver can be thought of as a black box, into which a concrete formulation of a problem can be inputted, and from which an optimal solution to that problem is outputted (Gusfield, 2019). There are many different LP solvers that exist for optimization problems: the common commercial

solvers are IBM ILOG CPLEX, Gurobi, and FICO Xpress Optimization (Bertsimas and Tsitsiklis, 1997; Cornuéjols, 2008) and the common open-source solvers are CBC, GLPK, JuliaOpt, and SCIP (which requires a commercial license for consulting work) (Bertsimas and Tsitsiklis, 1997; Cornuéjols, 2008). However, since the concrete formulation of the particular problem may be complicated, a modeling language such as AMPL, ZIMPL, or JuMP is useful for converting one set of parameters formed from a particular problem (such as the vertex and edge parameters of a graph that models the particular problem) to a concrete formulation ready to be inputted into an LP solver (Bertsekas, 1997; Bertsimas and Tsitsiklis, 1997; Cornuéjols, 2008; Williams, 2013).

While LPs can become exceptionally useful in solving optimization problems, sometimes it is desirable to find an optimal solution with integer variable values (Gusfield, 2019). For certain problems, it is necessary for the variable solutions to be integers, meaning it is necessary to build an *integer linear program*, or *ILP*, rather than an LP. However, solving ILPs is an NP hard optimization problem, and a polynomial-time ILP solver does not exist, even though there exist polynomial-time LP solvers (Gusfield, 2019).

2.4.2 Branch-and-Bound Description

In the case of the branch-and-bound algorithm, there is a search of possible optimal values, called the *candidate solutions*, that are found and compared with the current best solution, also called the *incumbent solution* (Morrison et al., 2016). When a candidate solution is better than the incumbent solution, it replaces the incumbent solution and the algorithm proceeds with the search using this new incumbent solution. This candidate solution gives either an *upper bound* (for minimization problems) or a *lower bound* (for maximization problems) for the optimal value given a smaller search space than the original search space.

The branch-and-bound algorithm is a tree-search algorithm that searches for the maximum solution of a given problem by dividing the original problem into smaller subproblems and finding the optimal solutions of those subproblems (Land and Doig, 1960). For each subproblem analyzed, an upper bound is calculated, and determines how large a lower bound can be. If a subproblem has a lower bound higher than the upper, then it can be pruned. This greatly helps in reducing

the computational runtime of an exhaustive search, while still making it possible to find the true optimal solution.

Typically the branch-and-bound algorithm consists of three components that are often uniquely made to fit the specific problem being tackled: the branching strategy, the search strategy, and the pruning rule (Morrison et al., 2016). The branching strategy is involved with the creation of the subproblems from the original problem, the search strategy is involved with establishing the order through which the subproblems are handled, and the pruning rule is involved with the calculations of the upper bounds of the subproblems. For each of these components, a technique may be chosen from a class of possible component techniques to suit the particular problem being solved. When choosing what techniques to use for each component, one must keep in mind that the implementation of each of these components greatly affects the performance of the branch-and-bound algorithm (Morrison et al., 2016). Not only this, but the type of technique chosen for the pruning rules may also affect the possible techniques that can be chosen for the branching strategy and search strategy, and the type of technique chosen for the branching strategy can affect the possible techniques chosen for the search strategy (Morrison et al., 2016).

There are two phases in the branch-and-bound algorithm: the search for an optimal result and the verification of the optimal result through solving all of the remaining subproblems (Morrison et al., 2016). The runtime of the branch-and-bound algorithm can often be improved if a good solution is found using a heuristic technique and inputted into the start of the algorithm, or if an optimal solution is found early in the search, which helps limit the size of the search tree created by the algorithm. If the verification runtime takes too long to confirm the optimal solution, the branch-and-bound algorithm can then be used as a heuristic and the unverified solution may be considered a “good” solution to the original problem (Morrison et al., 2016).

2.5 Overview of Loopy Belief Propagation

Belief propagation is a useful message-passing algorithm that can be used on tree graphs chosen to represent systems of random variables to find either a marginal probability using a *sum-product* message-passing, or a MAP inference using a *max-product* message-passing (chapter 13 of Koller

and Friedman (2009))². To find information about the likely state of a particular random variable x_i , the tree is rooted at x_i , and messages are passed from each child to its parent until x_i receives a message from all of its children. Any message $\mu_{\beta \rightarrow \alpha}(x_\alpha)$, passed from child β to parent α , contains the product of the prior messages received to that child, multiplied by a new factor supplied by the child itself (either $\sum_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta)$ or $\max_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta)$). An illustration of message-passing is shown in Figure 2.4

The sum-product message would be

$$\mu_{\beta \rightarrow \alpha}(x_\alpha) = \sum_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta) \prod_{\gamma \in \text{Nb}(x_\beta) \setminus \{\alpha\}} \mu_{\gamma \rightarrow \beta}(x_\beta)$$

and the computed marginal probability would be

$$P(x_i) = \frac{1}{Z} \prod_{j \in \text{Nb}(x_i)} \mu_{j \rightarrow i}(x_i)$$

The max-product message would be

$$\mu_{\beta \rightarrow \alpha}(x_\alpha) = \max_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta) \prod_{\gamma \in \text{Nb}(x_\beta) \setminus \{\alpha\}} \mu_{\gamma \rightarrow \beta}(x_\beta)$$

and the associated score for the MAP would be

$$\tilde{P}(x_i) = \prod_{j \in \text{Nb}(x_i)} \mu_{j \rightarrow i}(x_i)$$

Note that because Z is a normalizing constant that is often difficult to calculate, we do not need it for our MAP search since we are only concerned with finding our best state configuration S . Note also that when directionality is not important (like in the case of a Markov random field), we do not need to specify a relationship weight of $\phi_{(\beta, \alpha)}$, and instead we use ϕ .

The problem gets trickier when the graph that models the system of random variables is not a

²A very helpful collection of notes based on this textbook could be found here: <https://ermongroup.github.io/cs228-notes/inference/jt/>

tree. If this is the case, then it is best to use *loopy belief propagation*. Loopy belief propagation is a heuristic in which message-passing calculations are performed despite the graph not being a tree, with iterated steps over t between 1 and some chosen T (for a tree, convergence is achieved in two passes), or stopping at some other specified criterion. In the sum-product case, we first initialize our messages as

$$\mu_{\beta \rightarrow \alpha}^{(0)}(x_\alpha) = 1, \forall (\beta, \alpha) \in E$$

where E is our set of edges, and where every desired iteration update to the messages would be

$$\mu_{\beta \rightarrow \alpha}^{(t+1)}(x_\alpha) = \sum_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta) \prod_{\gamma \in Nb(x_\beta) \setminus \{\alpha\}} \mu_{\gamma \rightarrow \beta}^{(t)}(x_\beta)$$

and the calculation of the final computed marginal probability would be

$$P(x_i) = \frac{1}{Z} \prod_{j \in Nb(x_i)} \mu_{j \rightarrow i}^{(T+1)}(x_i)$$

In the max-product case, we would initialize our message like for sum-product, but our desired iteration update to the messages would be

$$\mu_{\beta \rightarrow \alpha}^{(t+1)}(x_\alpha) = \max_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta) \prod_{\gamma \in Nb(x_\beta) \setminus \{\alpha\}} \mu_{\gamma \rightarrow \beta}^{(t)}(x_\beta)$$

and our final associated score for the MAP would be

$$\tilde{P}(x_i) = \prod_{j \in Nb(x_i)} \mu_{j \rightarrow i}^{(T+1)}(x_i)$$

It is also worth mentioning that belief propagation is sometimes not able to solve the MAP inference exactly for a cyclic graph, and instead might return a suboptimal state. An example where such a suboptimal state is calculated is shown in Figure 2.5.

Figure 2.4: An example of message passing for the belief propagation performed in order to find information about the likely state of a particular random variable x_i . The example described in the text is represented in red messages, and all other messages are represented in blue.

	y_0	y_1	y_2
x_0	10	1	1
x_1	1	1	1
x_2	1	1	30

	z_0	z_1	z_2
x_0	10	1	1
x_1	1	1	1
x_2	1	1	3

	z_0	z_1	z_2
y_0	10	1	1
y_1	1	1	1
y_2	1	1	3

States of x : $\{x_0, x_1, x_2\}$

States of y :
 $\{y_0, y_1, y_2\}$

States of z :
 $\{z_0, z_1, z_2\}$

Max Marginals: $[x_0, x_1, x_2] = [10, 1, 30]$
 $[y_0, y_1, y_2] = [10, 1, 30]$
 $[z_0, z_1, z_2] = [10, 1, 3]$ } Highest scores possible

Chosen states: x_2, y_2, z_0 with a score of 32

True best state: x_2, y_2, z_2 with a score of 36

Figure 2.5: An example of a suboptimal state found when calculating the beliefs for a MAP inference on a cyclic graph. If only the states with the highest scores possible are chosen for each variable, then the chosen states, x_2, y_2, z_0 with an overall score of $30 + 1 + 1 = 32$, are not the optimal system states, since the “true” optimal system states are x_2, y_2, z_2 with a score of $30 + 3 + 3 = 36$.

Chapter 3

Models and Solution Methods

3.1 Connections between the MEWCP and MAP-MRF

While there exists literature on the connection between the approaches used to solve the MEWCP and the MAP-MRF, the connection between the MEWCP and the MAP-MRF in particular seems not to have been explicitly stated. For instance, belief propagation has been used as a heuristic for certain classes of ILPs in the past (Gelfand et al., 2013), but often maximum edge weight cliques and Markov random fields are not mentioned as they are a special subcategory of a more general connection. We make the assertion here that the MEWCP and a MAP-MRF are equivalent, and in making this assertion, we are able to relate a collection of approaches commonly used for MAP-MRF to a collection of approaches used by the MEWCP. Figure 3.1 outlines this assertion.

3.1.1 Connection between Belief Propagation and ILP Formulation

The MAP-MRF is an optimization problem. Note that this is different from finding the marginal inference Markov random field, because we are concerned with finding the state of a system of random variables, rather than a probability. Writing out the MAP-MRF yields

$$\max_{S=(s_1, s_2, \dots, s_n)} \sum_{c \in C} \theta_c(s_c)$$

Figure 3.1: This is an illustration of the claim that the MEWCP and MAP-MRF are connected. The red arrows indicate the possible approaches that can be taken to solve each respective problem, where the solid arrows represent the approaches studied by us, and the dashed arrows indicate existing approaches not studied by us. We attempt to make this claim in order to show that any single one of these approaches can be used in substitution for any other approach. The blue arrows indicate previous connecting research.

where $\theta_c(s_c) = \log(\phi_c(s_c))$. If we use the notation $S(x_i)$ to indicate the set of all states of variable x_i , and introduce the indicator variables $q_{s_i}^i$ for each $s_i \in S(x_i)$ and $q_{s_i, s_j}^{i,j}$ for each $s_i \in S(x_i)$ and $s_j \in S(x_j)$, we can then define the problem as:

$$\max_{S=\{s_1, s_2, \dots, s_N\}} \sum_{\forall i \in \{1, \dots, N\}} \sum_{\forall s_i \in S(x_i)} \theta_i(s_i) q_{s_i}^i + \sum_{\forall i, j \in \{1, \dots, N\}} \sum_{\forall s_i \in S(x_i), s_j \in S(x_j)} \theta_{ij}(s_i, s_j) q_{s_i, s_j}^{i,j}$$

subject to

$$q_{s_i}^i \in \{0, 1\}, \forall i \in \{1, \dots, N\}, s_i \in S(x_i)$$

$$q_{s_i, s_j}^{i,j} \in \{0, 1\}, \forall i, j \in \{1, \dots, N\}, s_i \in S(x_i), s_j \in S(x_j)$$

$$\sum_{\forall s_i \in S(x_i)} q_{s_i}^i = 1, \forall i \in \{1, \dots, N\}$$

$$\sum_{\forall s_j \in S(x_j)} q_{s_i, s_j}^{i,j} = q_{s_i}^i, \forall j \in \{1, \dots, N\}, s_i \notin S(x_j)$$

which is equivalent to the formulation for a MEWCP for a complete multipartite graph (chapter 13 of Koller and Friedman (2009))¹, seen in Section 3.2.1. There seems to exist a further deeper connection between belief propagation and LP through the weighted matching graph problem (which tries to find a set of edges so that each vertex is matched with another vertex and the weights of the chosen edges are maximized) and all instances of being able to use belief propagation as a heuristic to LPs (Gelfand et al., 2013). However, in this case either the MAP goes unmentioned, or the belief propagation finds the marginal inference rather than the MAP of a system of random variables.

3.1.2 Re-formulating between the MEWCP and the MAP-MRF

Any MAP-MRF can be re-formulated as a MEWCP of a multipartite graph. This is because each state of the random variable in a Markov random field can be represented as a vertex in one of the vertex sets of a multipartite graph. Figure 3.2 shows how this is done. Any MEWCP of a graph can also be re-formulated as a MAP-MRF, even if the graph is not seemingly a multipartite graph. This could be done by using a vertex coloring (a coloring of vertices so that no two vertices of the same color touch) to separate the vertices into sets of different colors. These sets will become the random variable. Figure 3.3 shows how this is done.

3.2 Modeling the Motif Discovery Problem as a MEWCP

3.2.1 Setup of the Motif Discovery as a MEWCP

If the motif discovery is simplified to a search of the best ungapped local multiple sequence alignments, then the main goal of the motif discovery problem is to, given a set of N input sequences $\{S_1, S_2, \dots, S_N\}$ and a motif-length parameter l , find an l -long segment (referred to as a

¹A very helpful collection of notes based on this textbook could be found here: <https://ermongroup.github.io/cs228-notes/inference/map/>

Figure 3.2: An example of going from a MAP-MRF formulation to a MEWCP formulation. Note that each variable becomes a vertex set, and each state of a variable of the MAP-MRF becomes its own vertex in the MEWCP. Note also that when a variable in the MAP-MRF shares an edge with another variable, then the equivalent two sets of vertices in the MEWCP are fully connected.

Figure 3.3: An example of going from a MEWCP formulation to a MAP-MRF formulation. We start with a graph, and we try to color it such that no two vertices with the same color share an edge (note that we do not need a minimum coloring to be able to break down a graph into a multipartite graph, but it is preferable so that we can minimize the number of random variables at the end). The colors indicate to us the disjoint vertex sets of our multipartite graph. We use a random variable to represent each disjoint set, and we represent each vertex from a set as a state of that random variable.

k -mer in some literature) from each input sequence so that the similarity between the l -long segments is maximized. The graph that is modeled from this formulation is a N -partite graph, wherein each set of vertices represents a single input sequence, and each vertex within each set represents an l -long segment from that input sequence. The edges of the graph only connect vertices of one

set of l -long segments to vertices of another set of l -long segments, which means that no two vertices of the same set of l -long segments will ever be connected, making the graph a *complete multipartite graph*. The weight of each edge will be the similarity between the two connecting l -long segments, and the solution to the MEWCP of this graph will find the motif, since it finds the most similar l -long segments shared between the N input segments. A small example of a graph representing the motif discovery problem is shown in Figure 3.4.

In order to find the similarity between the l -long segments, the *sum-of-pairs score*, or *SP score*, is used (Yanover et al., 2009; Zaslavsky and Singh, 2006). If the l -long segments were to all be lined up, then the similarity sums of each column may be calculated using a variety of methods: in simple cases, a 1 is added whenever there is a matching base or amino acid, and in more complicated cases (which we will use for our problem because it better reflects the natural realities of sequences), a $\log(1/f(b))$ may be added when there is a base b pairing, such that $f(b)$ is the zero-corrected frequency of base b in the background distribution of the l -long segment (Zaslavsky and Singh, 2006). This means that $f(b) = n(b)/N$ where $n(b)$ is the number of appearances of base b in the sequence alignment and N is the total number of bases. The sum of the column sums gives the final SP score. The SP scoring scheme proves to be superior to the position-specific scoring matrix, or PSSM, because the SP can address nucleotide/amino acid dependencies, while the PSSM assumes independence of motif positions (Zaslavsky and Singh, 2006).

3.2.2 The Approach Using ILP Formulation

Once the MEWCP has been specified for graph $G = (V, E)$ where V refers to the set of vertices of the graph, V_j refers to the vertex set j of the multipartite graph, E refers to the set of edges of the graph, and w_{uv} refers to the weight of the edge between vertex u and vertex v , it is then necessary to make the concrete formulation for the linear program. This formulation is largely taken from a previous application of LP solvers to perform motif discovery (Zaslavsky and Singh, 2006).

Figure 3.4: A small example of how the motif discovery problem is represented using a MEWCP. Each sequence is broken down into 3-long segments, each vertex set represents a sequence, and each vertex represents an l-long segment from that sequence. The solution to this problem is shown by coloring the vertices and bolding their shared edges. In this case, the “GAC” would be our motif.

The abstraction of the concrete formulation is the following:

$$\text{Variables} = \begin{cases} x_u & \forall u \in V \\ y_{uv} & \forall \{u, v\} \in E \end{cases}$$

where x_u is the binary decision variable for whether vertex u ends up in the clique, and y_{uv} is the binary decision variable for the existence of an edge between vertices u and v in the clique. This means that $x_u = 1$ when u is selected for the clique, and $y_{uv} = 1$ if u and v are both selected for

the clique.

Also:

$$\text{Linear Constraints} = \begin{cases} x_u, y_{uv} \in \{0, 1\} \\ \sum_{u \in V_j} x_u = 1 & \text{for } 1 \leq j \leq N \\ \sum_{u \in V_j} y_{uv} = x_v & \text{for } 1 \leq j \leq N, v \in V \setminus V_j \end{cases}$$

where $\sum_{u \in V_j} x_u = 1$ ensures that exactly one vertex is chosen from every set, and $\sum_{u \in V_j} y_{uv} = x_v$ ensures that for one vertex of one set, only one other vertex from a different set is picked to be connected via an edge.

Finally:

$$\text{Objective Function} = \max \sum_{\{u,v\} \in E} w_{uv} y_{uv}$$

which will solve for us MEWCP. A small example of this problem can be found in Figure 3.5.

Once we have completed the first part of this approach, turning the motif discovery problem into a MEWCP, we can then set up the ILP formulation approach in python² for the MEWCP by using the setup of ILP described above. There exist a large number of LP/ILP solvers that exist as python packages, but for our approach in particular, we will use the Gurobi (Gurobi Optimization, 2021) and Google OR-Tools (Perron and Furnon, 2019) python packages. We note that Google OR-Tools provides two kinds of solvers: a *constraint solver* and an ILP solver. Constraint solvers are usually applied to constraint satisfaction problems (though can be attempted to solve optimization problems, like in our implementation) and are based on search and domain reduction, whereas ILP solvers are numerical and are based on solving linear integer problems.

Even though we may treat both solvers similarly as black boxes that output our optimal solution, the output of multiple optimal solutions is handled differently between Gurobi and Google OR-Tools. When using Gurobi, it is possible to request a desired number of ordered optimal solutions (from best solution, to second best solution, and so on). However, Google OR-Tools does

²Our implementation can be found here: <https://github.com/AKMoSThesis/The-Use-of-Maximum-Edge-Weight-Clique-and-Markov-Random-Field-Problem-Formulation-to-Discover-Motifs>

not have this capability, so instead we must first find the optimal solution, then the entire space of feasible solutions that have an objective value greater than or equal to (optimal value) $\times 0.99$. Our chosen value of 0.99 can be substituted with another number between 0 and 1 depending on how big we want the solution space to be, but we found that in practice 0.99 is a good value to use to ensure that there are not too many solutions and not too few solutions. We also note that the setup of the ILP (the creation of the objective function, the variables, and the constraints) must differ slightly between Gurobi and Google OR-Tools.

For our python implementation, the user is able to select their desired solver by either specifying Gurobi or Google OR-Tools. If the user wishes to specify Gurobi, then the user must also specify the sequence file name, the motif length, and the number of desired solutions. If the user wishes to specify Google OR-Tools, then the user must also specify the sequence file name, the motif length, and whether they desire multiple solutions (expressed as either as “True” if the user wants multiple solutions or as “False” if the user does not want multiple solutions).

$$\begin{aligned}
\max \quad & y_{03} + y_{04} + y_{13} + y_{14} + 10y_{23} + y_{24} \\
\text{s.t.} \quad & x_u \in \{0, 1\} \text{ for } u \in V \\
& y_{uv} \in \{0, 1\} \text{ for } \{u, v\} \in E \\
& x_0 + x_1 + x_2 = 1 \\
& x_3 + x_4 = 1 \\
& y_{03} + y_{04} = x_0 \\
& y_{13} + y_{14} = x_1 \\
& y_{23} + y_{24} = x_2 \\
& y_{03} + y_{13} + y_{23} = x_3 \\
& y_{04} + y_{14} + y_{24} = x_4
\end{aligned}$$

$$\begin{aligned}
x_2, x_3 &= 1 \\
x_0, x_1, x_4 &= 0 \\
y_{23} &= 1 \\
y_{03}, y_{04}, y_{13}, y_{14}, y_{24} &= 0
\end{aligned}$$

Figure 3.5: A small example of a MEWCP being solved for a 2-partite graph using the ILP formulation approach. The red vertices and edge outline the maximum edge weight clique of the graph. The solution to the ILP is shown at the bottom right, where the decision variables x_2 and x_3 equal to 1 and the decision variable y_{23} is equal to 1, meaning that the vertices 2 and 3 and the edge between them are chosen for the maximum edge weight clique.

3.2.3 The Approach Using Branch-and-Bound

To solve the MEWCP using the branch-and-bound algorithm, the three components, the branching strategy, the search strategy, and the pruning rules must be specified to suit the problem. We will use $G = (V, E)$ to refer to a graph with a set of vertices V and a set of edges E , and $G(V)$ to refer to a graph with a set of vertices V and an implied set of edges $E(V)$. We will also use w_v and w_{uv} to refer to the weight of vertex v and the weight of the edge between vertex u and vertex v respectively, $N(v)$ to refer to the set of neighboring vertices to vertex v , and $W(V)$ to refer to the overall weight of $G(V)$, which we can find using that $W(V) = \sum_{v \in V} w_v + \sum_{\{u,v\} \in E(V)} w_{uv}$. Finally, we will also use C to denote a set of vertices in a feasible solution to the MEWCP on $G(V)$, and S to denote the set of vertices that neighbors every vertex in C , excluding the vertices already in C . This formulation is largely taken from a previous application of branch-and-bound to solve a general MEWCP (Shimizu et al., 2020).

Branching Strategy

Our goal is to solve a collection of subproblems, $P(C, S)$, on graph $G(V)$, where C is a clique of $G(V)$ and S is the set of possible candidate vertices to be added to the clique. $P(\emptyset, V)$, which denotes our original MEWCP, will attempt to find an F that maximizes $W(F)$, where F is the “best clique” in the graph $G(V)$ found by our algorithm.

Each subproblem of $P(C, S)$ will calculate a vertex sequence, $\Pi = [p_1, p_2, \dots, p_{|S|}]$ that specifies the order of vertices that should be searched through in the hopes of adding it to the clique, and a list of upper bounds, $upper = [\dots]$, of each new clique when p_i is added to the previous clique. Using the list Π to specify the order of searches of a candidate p_i to be added, the algorithm then searches through subproblem $P(C \cup p_i, (S \setminus \{p_j | j < i\}) \cap N(p_i))$, where $C \cup p_i$ indicates that the subproblem has a new vertex, p_i , added to the clique, and $(S \setminus \{p_j | j < i\}) \cap N(p_i)$ indicates the set of all vertices that are neighbors to all of the vertices from the smaller clique C and that are also neighbors of the newly-added vertex.

Searching Strategy

We note that the problem $P(C, S)$ gets smaller as the clique C becomes bigger because a bigger C will make a smaller S , which then makes the search space smaller. Because subproblems are generated until there are no more vertices that can be added to the clique, meaning $S = \emptyset$, we can say that this branch-and-bound algorithm explores subproblems to the very depth of the tree with a “completed” clique C before backtracking to find another clique, meaning that the search for a solution to the MEWCP becomes a depth-first search.

Pruning Rules

For this algorithm, $P(C, S)$ calculates the upper bound of $W(F)$, and any subproblems that calculate a $W(F)$ less than a previously calculated $W(F)$ are pruned. In order to find $\Pi = [p_1, p_2, \dots, p_{|S|}]$ and $upper = [\dots]$, which indicate to us the list of p_i 's and their upper bounds on $W(F)$, we establish a specified weight, $\sigma(v)$, to each vertex v , such that the specified weight is calculated using both the original weight of the vertex and the weight of the edges that connect to that vertex, $\sigma(v) = w_v + \sum_{i < \tau(v)} \max\{w_{uv} | v \in I_i \cap N(v)\}$ where $\tau(v) = j$ indicates that vertex v belongs to set j of the multipartite graph and I_i indicates the i th set of the k disjoint sets that make up the multipartite graph. Using this specified weight, the Π and $upper$ is found for $P(C, S)$ by combining $W(H) \leq \sum_{i=1}^k \max\{\sigma(v) | v \in I_i\}$ and $W(F) = W(C) + W(H')$ to form $W(F) \leq W(C) + \sum_{i=1}^k \max\{\sigma(v) | v \in I'_i\}$, where H' and I'_i are taken from the graph G' which differs from the original graph G by replacing w_{s_i} with $\sum_{u \in C} w_{us_i}$ for every s_i in S , but is otherwise equivalent.

A small example of this problem can be found in Figure 3.6. After turning the motif discovery problem into a MEWCP problem and working out the details of the branching strategy, the searching strategy, and the pruning rules, it is then possible to put together the branch-and-bound algorithm³.

³Our implementation can be found here: <https://github.com/AKMoSThesis/The-Use-of-Maximum-Edge-Weight-Clique-and-Markov-Random-Field-Problem-Formulation-to-Discover-Motifs>

Note that the vertex coloring for this particular problem is unnecessary, since our multipartite graph already has a known set of disjoint vertices that are calculated when the motif discovery problem is turned into a MEWCP. Nevertheless, the vertex coloring was left in this python implementation because previous research showed that regardless of whether the disjoint sets of vertices were known, the branch-and-bound algorithm still greatly outperformed many existing methods, including methods that use a CPLEX solver (Shimizu et al., 2020). However, the use of a global clique used in previous research was not used in our python implementation, since it is not desirable to use global variables in python. The algorithm was further changed from previous research to be able to produce multiple solutions. To do this, the algorithm is iterated multiple times, where iteration prevents the selection of an already-found clique, which then forces the selection of the next-best solution. The detailed pseudocode for this implementation can be found in Shimizu et al. (2020), with our Algorithm 1 being unchanged from the original and with a modified Algorithm 2 and Algorithm 3, all outlined in Figure 3.7, Figure 3.8 and Figure 3.9.

3.3 Modeling the Motif Discovery Problem as a MAP-MRF

3.3.1 Setup of the Motif Discovery as a MAP-MRF

When trying to model the motif discovery problem as a prediction of the state of a system of random variables, we consider a random l -long segment from one sequence to be a random variable, and we attempt to find the motif, which we consider to be the “true” state of that random variable. We use a Markov random field instead of a Bayesian network, because the appearance of a motif in one sequence is not known to directly cause the appearance of the motif in another sequence, but rather each motif must relate somehow to each other motif. Since the dependencies are unknown and the selection of one l -long segment from one sequence must depend on the selection of l -long segments from each of the other sequences, the graph modeling this problem is complete (cyclic) with complicated interdependencies, making the Markov random field the more appropriate choice.

In this case, $S = \{s_1, s_2, \dots, s_N\}$ will represent the overall state of the system, where each s_i represents an l -long segment selected from sequence S_i . Since the selection of any one motif will

Figure 3.6: A small example of a MEWCP being solved for a 2-partite graph using the branch-and-bound algorithm approach. The red vertices and edge outline the maximum edge weight clique of the graph. The algorithm starts with the subproblem $P(\emptyset, \{0, 1, 2, 3, 4\})$ with a lower bound of $W(F) = 0$, and it finds an ordered set of vertices, Π , and the corresponding upper bounds of each new clique formed from each of those vertices. The algorithm then looks through $P(\{3\}, \{0, 1, 2\})$ and finds another ordered set of vertices Π with upper bounds. The branching ends at $P(\{2, 3\}, \emptyset)$ when there are no more vertices that can be added. The algorithm does not investigate $P(\{0, 3\}, \emptyset)$ and $P(\{1, 3\}, \emptyset)$ because they have upper bounds of $W(F) = 1$, which is lower than the lower bound of $W(F) = 10$ for the subproblem $P(\{3\}, \{0, 1, 2\})$. The algorithm returns to the top, but it does not investigate $P(\{4\}, \{0, 1, 2\})$ because the subproblem has an upper bound of $W(F) = 1$, which is lower than the new lower bound of $W(F) = 10$ (found previously). This means that $\{2, 3\}$ are the vertices belonging to our maximum edge weight clique.

affect the selection of another, there will exist a relationship weight of $\phi(s_i, s_j)$ for each pair of variables, making an overall score of the system $\tilde{P}(S) = \prod_{s_i \in S_i, s_j \in S_j, i < j} \phi(s_i, s_j)$ and an overall probability of the system $P(S) = \frac{1}{Z} \prod_{s_i \in S_i, s_j \in S_j, i < j} \phi(s_i, s_j)$. The calculation of each relationship weight will be the sum-of pairs score outlined in Section 3.2.1. In order to solve this problem and find the motifs, we want to do a MAP-MRF, since we want to find the state most likely to give us the motifs. A small example of a graph representing the motif discovery problem is shown in Figure 3.10.

3.3.2 The Approach Using Loopy Belief Propagation

To solve the MAP-MRF, we can use the max-product message passing loopy belief propagation⁴. Since belief propagation is more suitable for tree graphs and loopy belief propagation is

⁴The loopy belief propagation component of the code was taken from <https://github.com/krashkov/Belief-Propagation>

Algorithm 1 Calculate a vertex sequence and upper bounds

INPUT: a subproblem $P(C, S)$ **OUTPUT:** a vertex sequence Π and an array $upper[\dots]$

```
1: procedure CALC_SEQ_AND_UB( $C, S$ )
2:   for  $v \in S$  do
3:      $\sigma[v] \leftarrow \sum_{u \in C} w_{uv}$ 
4:   end for
5:    $T \leftarrow S$ 
6:    $k \leftarrow 0$ 
7:   while  $T \neq \emptyset$  do
8:      $k \leftarrow k + 1$ 
9:      $I_k \leftarrow \emptyset$ 
10:     $X \leftarrow T$ 
11:    while  $X \neq \emptyset$  do
12:       $v \leftarrow$  a vertex of minimum  $\sigma[\dots]$  in  $X$ 
13:       $I_k \leftarrow I_k \cup \{v\}$ 
14:      Append  $v$  to the head of  $\Pi$ .
15:       $upper[v] \leftarrow \sigma[v] + \sum_{i < k} \max\{\sigma[u] | u \in I_i\}$ 
16:       $X \leftarrow X \setminus N(v)$ 
17:       $T \leftarrow T \setminus \{v\}$ 
18:    end while
19:    for  $v \in T$  do
20:       $\sigma \leftarrow \sigma[v] + \max\{w_{uv} | u \in I_k \cap N(v)\}$ 
21:    end for
22:  end while
23:  return  $\Pi, upper[\dots]$ 
24: end procedure
```

Figure 3.7: The unmodified Algorithm 1 pseudocode taken from Shimizu et al. (2020).

more suitable for cyclic graphs, we will use loopy belief propagation on our complete graph. That means that we will first initialize our messages as

$$\mu_{\beta \rightarrow \alpha}^{(0)}(x_\alpha) = 1, \forall (\beta, \alpha) \in E$$

where E is our set of edges, and where every desired iteration update to the messages from 1 to $T = 10$ (a good number we found in practice) would be

$$\mu_{\beta \rightarrow \alpha}^{(t+1)}(x_\alpha) = \max_{x_\beta} \phi_{(\beta, \alpha)}(x_\alpha, x_\beta) \prod_{\gamma \in Nb(x_\beta) \setminus \{x_\alpha\}} \mu_{\gamma \rightarrow \beta}^{(t)}(x_\beta)$$

Algorithm 2 MECQ (adjusted)

INPUT: $G = (V, E), w., C_{initial}$ **OUTPUT:** a maximum edge weight clique C_{max}

```
1:  $C_{max} \leftarrow C_{initial}$ 
2:  $C_{max} \leftarrow \text{EXPAND}(\emptyset, C_{max}, V, \emptyset)$ 
3:  $cliques = \{C_{max}\}$ 
4: for num 1:num_solutions-1 do
5:    $C_{max} \leftarrow C_{initial}$ 
6:    $C_{max} \leftarrow \text{EXPAND}(\emptyset, C_{max}, V, cliques)$ 
7:    $cliques \leftarrow cliques \cup \{C_{max}\}$ 
8: end for
9: return  $C_{max}$ 
```

Figure 3.8: The modified Algorithm 2 pseudocode taken from Shimizu et al. (2020) with our modifications shown in red.

Algorithm 3 Solving a subproblem (adjusted)

INPUT: a subproblem $P(C, S), cliques$ **OUTPUT:** update C_{max} to a better clique if it exists.

```
1: procedure EXPAND( $C, C_{max}, S, cliques$ )
2: if  $S = \emptyset$  then
3:   if  $W(C) > W(C_{max})$  and  $C \notin cliques$  then
4:      $C_{max} \leftarrow C$ 
5:   end if
6:   return
7: end if
8:  $\Pi, upper[\dots] \leftarrow \text{CALC\_SEQ\_AND\_UB}(C, S)$ 
9: for each  $p_i$  in order of  $\Pi$  do
10:  if  $W(C) + upper[p_i] > W(C_{max})$  then
11:     $C_{max} \leftarrow \text{EXPAND}(C \cup \{p_i\}, C_{max}, (S \setminus \{p_j | j < i\}) \cap N(p_i), cliques)$ 
12:  end if
13: end for
14: return  $C_{max}$ 
15: end procedure
```

Figure 3.9: The modified Algorithm 3 pseudocode taken from Shimizu et al. (2020) with our modifications shown in red.

and our final associated score for the MAP would be

$$\tilde{P}(x_i) = \prod_{j \in Nb(x_i)} \mu_{j \rightarrow i}^{(T+1)}(x_i)$$

Figure 3.10: A small example of how the motif discovery problem is represented using a MAP-MRF. Each sequence is broken down into 3-long segments, each vertex represents a sequence, and the state of each vertex represents an l -long segment from that sequence. The solution to this problem is shown by bolding our motif, “GAC”.

In order to find the best max marginal, we will find the s_i that gives us the highest possible score of $\tilde{P}(x_i = s_i)$, where x_i used to represent a randomly-chosen l -long segment of sequence S_i . We will record our first computed optimal solution as $S^1 = (s_1^1, s_2^1, \dots, s_N^1)$, with a superscript of 1 to indicate that these are the states found in the first solution. A small example of this problem can be found in Figure 3.11.

In order to generate multiple solutions, we will keep track of our previous solutions and use

them to find new ones. This strategy is largely taken from a previous application of belief propagation to perform motif discovery (Yanover et al., 2009; Yanover and Weiss, 2003). To obtain the m th next solution, we will pick an $s_i^j \in \{s_i^1, s_i^2, \dots, s_i^{m-1}\}$ to switch with an s_i^{j*} so that this switch leads to the highest possible score $\tilde{P}(x_i = s_i^{j*})$ out of any switch. We will constrain our future searches to avoid repeating a switch of s_i^j with s_i^{j*} (and avoid repeating any prior switch already performed), and we will constrain our current search to ensure that x_i equals s_i^{j*} (and also avoid repeating any prior switch already performed). In finding our best max marginals given these constraints, we are able to find our m th next solution, $S^m = (s_1^m, s_2^m, \dots, s_i^m(= s_i^{j*}), \dots, s_N^m)$. We note that all random variables are considered when trying to find the next solution, and that all random variables may be considered again for a subsequent solution. The detailed pseudocode for this implementation can be found in Yanover et al. (2009). Our implementation is written in python⁵. We note that our solution order is based on our $\tilde{P}(S)$ score values, and that once we find a solution, we must recalculate our overall score to be the sum of all the similarities between the motifs.

⁵Our implementation can be found here: <https://github.com/AKMoSThesis/The-Use-of-Maximum-Edge-Weight-Clique-and-Markov-Random-Field-Problem-Formulation-to-Discover-Motifs>

Figure 3.11: A small example of a MAP-MRF being solved for a system of random variables using the loopy belief propagation approach. The orange squares outline the most likely states of the random variables of the system (note that the selection of $x_0 = s_2$ and $x_1 = s_3$ leads to the highest score possible). Messages are chosen to pass twice to x_0 to compute the max marginals of x_0 , and messages are chosen to pass twice to x_1 to compute the max marginals of x_1 . We can see that for x_0 , s_2 has the highest value for the max marginals, and for x_1 , s_3 has the highest value for the max marginals.

Chapter 4

Results

4.1 Dataset Used and Machine Specifications

In order to test the three approaches outlined in Section 3, we will attempt to detect the motifs found in the DNA sequence data of *Escherichia coli* K-12 genome binding sites (Osada et al., 2004). This sequence data is in FASTA format, where the string of capitalized letters in each *binding site* (sequence) indicates the *transcriptional regulator* (motif) of the binding site, and all other letters not belonging to the transcriptional regulator of the binding site are lowercase. The data (in the form of a 105KB FASTA file) consists of 35 transcription factors and 410 binding sites assembled from the data of 68 regulatory proteins and their aligned DNA binding sites (McGuire et al., 2000; Robison et al., 1998). We used a remote serve equipped with 125GB of RAM and two 8-core CPUs, model Intel Xeon E5-2667v2 with clock speed 3.3GHz.

4.2 Survey of Results

To test our three approaches, we chose the binding sites (four in total) associated with the transcriptional regulator of *cspA* from our dataset, using an $l = 20$ in an attempt to detect the transcriptional regulator of length 20. For these sequences, we found that the ILP formulation using both Gurobi and Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation all outputted the same best solution (having a score of 134.788), with the same motifs found from each sequence. When we attempted to find the next several best solutions, we found

that the ILP formulation using Gurobi and Google OR-Tools and the branch-and-bound algorithm all produced the same solutions (having scores of 133.227 and 133.087). However, the loopy belief propagation found different, inferior solutions from the other two approaches (having scores of 133.045 and 132.060). These inferior solutions were also found by the ILP formulation using Gurobi, but they were found to be the fourth and sixth best solutions (according to Gurobi) rather than the second and third best solutions. The detailed results for the binding sites associated with the transcriptional regulator of *cspA* are shown in Table 4.1 and Figure 4.1.

The fastest exact motif finder tested on the binding sites associated with the transcriptional regulator of *cspA* was the ILP formulation with Gurobi (taking about a minute and a half), and the second fastest exact motif finder was the ILP formulation with Google OR-Tools (which was about eight times slower) followed by the branch-and-bound algorithm (which was even slower than that). However, the loopy belief propagation, our heuristic motif finder, found a result about four times faster than the ILP formulation with Gurobi.

```
>cspA:001
CGGTTTGAACCTCACCTTCCAGATCCCACCAGCGGGAGGCGACGGCTTCAAATTTAGCGAT
CTCTTCGTGGTCTACGTTATGGTTTACCGGCGATTTTTTCCGCATTATTGGCACTTCTAC
TCCGTAATTGGCAAGACAAACGAGTATATCAGGCATTGGATGTGAATAAAGCGTATAGGT
TTACCTCAAACGCGCGGCTGTGTTATAATTTGCGACCTT
>cspA:002
GATCCCACCAGCGGGAGGCGACGGCTTCAAATTTAGCGATCTCTTCGTGGTCTACGTTAT
GGTTTACCGGCGATTTTTTCGGCATTATTGGCACTTCTACTCCGTAATTGGCAAGACAAA
CGAGTATATCAGGCATTGGATGTGAATAAAGCGTATAGGTTTACCTCAAACGCGCGGCT
GTGTTATAATTTGCGACCTTTGAATCCGGGATACAGTAGA
>cspA:003
AAATTTAGCGATCTCTTCGTGGTCTACGTTATGGTTTACCGGCGATTTTTTCGGCATTCTAT
TGGCACTTCTACTCCGTAATTGGCAAGACAAACGAGTATATCAGGCATTGGATGTGAATA
AAGCGTATAGGTTTACCTCAAACGCGCGGCTGTGTTATAATTTGCGACCTTTGAATCCG
GGATACAGTAGAGGGATAGCGGTTAGATGAGCGACCTTGC
>cspA:004
ATTTCTTCCAGCGTTTCAAGTGATACATTCTTTGCTGCGCACGAAGAGTACGGATGTTG
TTCAGAATTTTAAAGTGCTTCGCTCATTGTAGTAATCTCAAACTTATATTGGGGTGGTTTG
TTGAGGTAATAATAGAGCCTTAAATTCAGTTGTGAATAGCCAGGAATGTAAGGAATTCA
AAATTGTTCTTTATTTTGCGCGCAATAAATATCTTTTC
```

Figure 4.1: Motif discovery results for the four binding sites associated with the transcriptional regulator of *cspA* from our *E. coli* dataset. Here the optimal solution found by our exact motif finders and our heuristic motif finder is the string segment in red. Our transcriptional regulator is the string segment in blue.

For the binding sites associated with the transcriptional regulator of *hipB* (with four binding sites and a transcriptional regulator length of $l = 31$), we obtained different results than with

ILP Formulation using Gurobi			
Motif Length	20		
Time (secs)	84.78		
Solution #	Solution 0	Solution 1	Solution 2
cspA:001	CGTTATGGTTTACCGGCGAT	CCTCAAACGCGCGGCTGTG	ACCTCAAACGCGCGGCTGT
cspA:002	CGTTATGGTTTACCGGCGAT	CCTCAAACGCGCGGCTGTG	ACCTCAAACGCGCGGCTGT
cspA:003	CGTTATGGTTTACCGGCGAT	CCTCAAACGCGCGGCTGTG	ACCTCAAACGCGCGGCTGT
cspA:004	CTTTATTTGTGCCGCCAAT	GCACGAAGAGTACGGATGTT	CGCACGAAGAGTACGGATGT
Score	134.788	133.227	133.087

ILP Formulation using Google OR-Tools			
Motif Length	20		
Time (secs)	639.05		
Solution #	Solution 0		
cspA:001	CGTTATGGTTTACCGGCGAT		
cspA:002	CGTTATGGTTTACCGGCGAT		
cspA:003	CGTTATGGTTTACCGGCGAT		
cspA:004	CTTTATTTGTGCCGCCAAT		
Score	134.788		

Branch-and-bound algorithm			
Motif Length	20		
Time (secs)	18351.40		
Solution #	Solution 0	Solution 1	Solution 2
cspA:001	CGTTATGGTTTACCGGCGAT	CCTCAAACGCGCGGCTGTG	ACCTCAAACGCGCGGCTGT
cspA:002	CGTTATGGTTTACCGGCGAT	CCTCAAACGCGCGGCTGTG	ACCTCAAACGCGCGGCTGT
cspA:003	CGTTATGGTTTACCGGCGAT	CCTCAAACGCGCGGCTGTG	ACCTCAAACGCGCGGCTGT
cspA:004	CTTTATTTGTGCCGCCAAT	GCACGAAGAGTACGGATGTT	CGCACGAAGAGTACGGATGT
Score	134.788	133.227	133.087

Loopy Belief Propagation			
Motif Length	20		
Time (secs)	18.99		
Solution #	Solution 0	Solution 1	Solution 2
cspA:001	CGTTATGGTTTACCGGCGAT	TTGGATGTGAATAAAGCGTA	ACGTTATGGTTTACCGGCGA
cspA:002	CGTTATGGTTTACCGGCGAT	TTGGATGTGAATAAAGCGTA	ACGTTATGGTTTACCGGCGA
cspA:003	CGTTATGGTTTACCGGCGAT	TTGGATGTGAATAAAGCGTA	ACGTTATGGTTTACCGGCGA
cspA:004	CTTTATTTGTGCCGCCAAT	TTGCCTGCGCACGAAGAGTA	TCTTTATTTGTGCCGCCAA
Score	134.788	133.045	132.060

Transcriptional regulator	
Motif Length	20
cspA:001	GCATTCATTGGCACTTCTAC
cspA:002	TCCGTAATTGGCAAGACAAA
cspA:003	TCAGGCATTGGATGTGAATA
cspA:004	ACTTATATTGGGTGGTTTGT
Score	72.212

Table 4.1: Motif discovery results for the four binding sites associated with the transcriptional regulator of *cspA* from our *E. coli* dataset. We compare between the ILP formulation using Gurobi, the ILP formulation using Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation.

cspA. For the binding sites associated with the transcriptional regulator of *hipB*, we were able to find the same best solutions for all three of our approaches, meaning our heuristic motif finder, loopy belief propagation, was able to find the exact solutions. The computational runtime of the ILP formulation using Google OR-Tools also ended up being much shorter than the computational runtime of the ILP formulation using Gurobi. The detailed results for the binding sites associated with the transcriptional regulator of *hipB* are shown in Table 4.2 and Figure 4.2.

For the binding sites associated with the transcriptional regulator of *oxyR* (with four binding sites and a transcriptional regulator length of $l = 40$), we obtained different results than with *cspA*

ILP Formulation using Gurobi			
Motif Length	31		
Time (secs)	1654.34		
Solution #	Solution 0	Solution 1	Solution 2
hipB:001	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:002	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:003	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:004	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
Score	264.620	264.202	263.676

ILP Formulation using Google OR-Tools			
Motif Length	31		
Time (secs)	62.01		
Solution #	Solution 0		
hipB:001	CGCGATCGCGGATATCGCAGCGTTATCCCG		
hipB:002	CGCGATCGCGGATATCGCAGCGTTATCCCG		
hipB:003	CGCGATCGCGGATATCGCAGCGTTATCCCG		
hipB:004	CGCGATCGCGGATATCGCAGCGTTATCCCG		
Score	264.620		

Branch-and-bound algorithm			
Motif Length	31		
Time (secs)	35723.23		
Solution #	Solution 0	Solution 1	Solution 2
hipB:001	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:002	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:003	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:004	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
Score	264.620	264.202	263.676

Loopy Belief Propagation			
Motif Length	31		
Time (secs)	19.01		
Solution #	Solution 0	Solution 1	Solution 2
hipB:001	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:002	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:003	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
hipB:004	CGCGATCGCGGATATCGCAGCGTTATCCCG	CCGCGATCGCGGATATCGCAGCGTTATCCC	CGGATATCGCAGCGTTATCCCGTAGAGCGG
Score	264.620	264.202	263.676

Transcriptional regulator	
Motif Length	31
hipB:001	TTATAATATCCCCTAAGCGGATAAACTTG
hipB:002	TTGACTTATCCTCACTAAAGGATAAACTT
hipB:003	AGCGTTTATCCCGTAGAGCGGATAAGATGT
hipB:004	CCTTTTATCCCGGATCGCGGATATCGCAG
Score	126.916

Table 4.2: Motif discovery results for the four binding sites associated with the transcriptional regulator of *hipB* from our *E. coli* dataset. We compare between the ILP formulation using Gurobi, the ILP formulation using Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation.

and *hipB*. For the binding sites associated with the transcriptional regulator of *oxyR*, our exact solutions were able to find the same solutions. However, for our loopy belief propagation, our “best” solution was not only different from the best solution found by the exact motif finders, but also this “best” solution had a score lower than the score for the transcriptional regulators themselves. Nevertheless, our second solution for loopy belief propagation was the best solution according to the exact motif finders. The detailed results for the binding sites associated with the transcriptional regulator of *oxyR* are shown in Table 4.3 and Figure 4.3.

```

>hipB:001
ATCCTCCTTTTTATCCGCGATCGCGGATATCGCAGCGTTTATCCCGTAGAGCGGATAAGA
TGTGTTTCCAGATTGACTTATCTCACTAAAGGATAAACTTTATAATATCCCCTTAAGCG
GATAAACTTGTGTGGACGTATGACATGATGAGCTTTCAGAAGATCTATAGCCCAACGCA
ATTGGCGAATGCAATGAACTGGTTCGCCAGCAAAATGGCTGGACGCAGA
>hipB:002
TCAAACGGCGCATCAACGCCTTATTTAAATCCTCCTTTTTATCCGCGATCGCGGATATCG
CAGCGTTTATCCCGTAGAGCGGATAAGATGTGTTTCCAGATTTGACTTATCCTCACTAAAG
GATAAACTTATAATATCCCCTTAAGCGGATAAACTTGCTGTGGACGTATGACATGATGA
GCTTTCAGAAGATCTATAGCCCAACGCAATTGGCGAATGCAATGAACTG
>hipB:003
CCGCATGTCGAGATGGCATGCGGCTTAATATTGCCGACTTCAAACGGCGCATCAACGCCT
TATTTAAATCCTCCTTTTTATCCGCGATCGCGGATATCGCAGCGTTTATCCCGTAGAGCG
GATAAGATGTGTTTCCAGATTGACTTATCCTCACTAAAGGATAAACTTATAATATCCC
TTAAGCGGATAAACTTGCTGTGGACGTATGACATGATGAGCTTTCAGAAG
>hipB:004
GTGAAATATAGCTGGTAAACCGTATAAGCCGCATGTCGAGATGGCATGCGGCTTAATATT
GCCGACTTCAAACGGCGCATCAACGCCTTATTTAAATCCTCCTTTTATCCGCGATCGCG
GATATCGCAGCGTTTATCCCGTAGAGCGGATAAGATGTGTTTCCAGATTGACTTATCCTC
ACTAAAGGATAAACTTATAATATCCCCTTAAGCGGATAAACTTGCTGTG

```

Figure 4.2: Motif discovery results for the four binding sites associated with the transcriptional regulator of *hipB* from our *E. coli* dataset. Here the optimal solution found by our exact motif finders and our heuristic motif finder is the string segment in red. Our transcriptional regulator is the string segment in blue. The string segment in purple is the overlap between our optimal solution and the transcriptional regulator.

```

>oxyR:001
AAAAGCATATTTATTCGAAATTCCTGGCGAGCAGATAAATAAGAATTGTTCTTATCAA
TATATCTAACTCATTGAATCTTTATTAGTTTTGTTTTTCACGCTTGTTACCACTATTAGT
GTGATAGGAACAGCCAGAATAGCGGAACACATAGCCGGTGCTATACTTAATCTCGTTAAT
TACTGGGACATAACATCAAGAGGATATGAAATTATGAGTACCGCTAAATTAGTTAAATC
>oxyR:002
TAACCCGCGTGGAACCTGGAACAACAGATGAATAACGTGCTGCCGTGGCTGCACAGCAA
ACTGGTTCCGGCAGGCACCGGGCACGCCACCGTAAGCTGGATCGTGCCGGAGTAATTGCA
GCCATTGCTGGACCTATTACGTCTCGCGCTACAATCGCGGTAATCAACGATAAGGACAC
TTTGTCTGACTAAACACTATGATTACATCGCCATCGGCGGCGCAGCGCGGTATCGC
>oxyR:003
AGCTAAAACCTTTTTTATAAAGCATTTGTCCGAATTCGGACATAATCAAAAAAGCTTAA
TTAAGATCAATTTGATCTACATCTCTTTACCAACAATATGTAAGATCTCAACTATCGCA
TCCGTGGATTAATTCAATTATAACTTCTCTTAACGCTGTGTATCGTAACGGTAACACTG
TAGAGGGGAGCACATTGATGAGCACGTGAGACGATATCCATAACACCACAGCCACTGGC
>oxyR:004
ACGTGGCAGGAATCTGCCGCACGCCGAAAATGGCGGTGTTAGCCAATGCCACCAGGTAC
TCAAGATCACGAATATTCATTATCCATCCTCCATCGCCACGATAGTTATGGCGATAGGT
AGAATAGCAATGAACGATTATCCCTATCAAGCATTCTGACTGATAATTGCTCACAGAAAC
GGAGCGGCACCTCTTTAACCCTTGAAGTCACTGCCCGTTTCGAGAGTTTCTCAACTCG

```

Figure 4.3: Motif discovery results for the four binding sites associated with the transcriptional regulator of *oxyR* from our *E. coli* dataset. Here the optimal solution found by our exact motif finders is the string segment in red. Our transcriptional regulator is the string segment in blue. The string segment in purple is the overlap between our optimal solution and the transcriptional regulator.

ILP Formulation using Gurobi			
Motif Length	40		
Time (secs)	1055.07		
Solution #	Solution 0	Solution 1	Solution 2
oxyR:001	AGCAGATAAATAAGAATTGT-TCTTATCAATATATCTAACT	GAATTGTTCTTATCAATATA-TCTAACTCAATTGAATCTTTA	GCAGATAAATAAGAATTGTT-CTTATCAATATATCTAACTC
oxyR:002	AACAGATGAATAACGTGCTG-CCGTGGCTGCACAGCAAACCT	GGAGTAATTGCAGCCATTGC-TGGCACCTATTACGTCTCGC	ACAGATGAATAACGTGCTGC-CGTGGCTGCACAGCAAACCTG
oxyR:003	AAAAGCTTAATTAAGATCAA-TTTGATCTACATCTCTTTAA	GGATTAATTCAATTATAACT-TCTCTCAACGCTGTGTATC	GTGGATTAATTCAATTATAA-CTTCTCTCAACGCTGTGTA
oxyR:004	GGTAGAATAGCAATGAACGA-TTATCCCTATCAAGCATTCT	CGATTATCCCTATCAAGCAT-TCTGACTGATAATTGCTCAC	GTAGAATAGCAATGAACGAT-TATCCCTATCAAGCATTCTG
Score	145.964	145.583	145.523

ILP Formulation using Google OR-Tools		
Motif Length	40	
Time (secs)	77459.52	
Solution #	Solution 0	
oxyR:001	AGCAGATAAATAAGAATTGT-TCTTATCAATATATCTAACT	
oxyR:002	AACAGATGAATAACGTGCTG-CCGTGGCTGCACAGCAAACCT	
oxyR:003	AAAAGCTTAATTAAGATCAA-TTTGATCTACATCTCTTTAA	
oxyR:004	GGTAGAATAGCAATGAACGA-TTATCCCTATCAAGCATTCT	
Score	145.964	

Loopy Belief Propagation			
Motif Length	40		
Time (secs)	29.89		
Solution #	Solution 0	Solution 1	Solution 2
oxyR:001	GAATTGTTCTTATCAATATA-TCTAACTCATTGAATCTTTA	AGCAGATAAATAAGAATTGT-TCTTATCAATATATCTAACT	TCTCGTTAATTACTGGGACA-TAACATCAAGAGGATATGAA
oxyR:002	AACAGATGAATAACGTGCTG-CCGTGGCTGCACAGCAAACCT	AACAGATGAATAACGTGCTG-CCGTGGCTGCACAGCAAACCT	GGTAATCAACGATAAGGACA-CTTGTCTATGACTAAACACT
oxyR:003	CGTGGATTAATTAAGATCAA-TTCTCTCTTAACGCTGTGT	AAAAGCTTAATTAAGATCAA-TTGTATCTACATCTCTTTAA	TCTCTTTAACCAACAATATG-TAAGATCTCAACTATCGCAT
oxyR:004	GGTAGAATAGCAATGAACG-ATTATCCCTATCAAGCATTCT	GGTAGAATAGCAATGAACG-ATTATCCCTATCAAGCATTCT	AGCCAATGCCACCAGGTACT-CAAGATCAGGAATATTCATT
Score	110.068	145.964	137.679

Transcriptional regulator	
Motif Length	40
oxyR:001	CGCTTGTTACCACTATTAGT-GTGATAGGAACAGCCAGAA
oxyR:002	ATCGTGCCGGAGTAATTGCA-GCCATTGCTGGCACCTATT
oxyR:003	GTAAGATCTCAACTATCGCA-TCCGTGGATTAATTCAATT
oxyR:004	GATAGTTTCATGGCGATAGGT-AGAATAGCAATGAACGATT
Score	115.007

Table 4.3: Motif discovery results for the four binding sites associated with the transcriptional regulator of *oxyR* from our *E. coli* dataset. We compare between the ILP formulation using Gurobi, the ILP formulation using Google OR-Tools, the branch-and-bound algorithm, and the loopy belief propagation.

4.3 Discussion

When we found that the ILP formulation using Gurobi and Google OR-Tools and the branch-and-bound algorithm produced the same results for each collection of binding sites, this lined up with our expectations, since the ILP formulation and the branch-and-bound algorithm are exact searches and are supposed to output the best global solutions. It was, however, somewhat surprising to find that the branch-and-bound algorithm was significantly less efficient than both implementa-

tions of the ILP formulation, since the branch-and-bound algorithm that solves the MEWCP was taken from a very recent study that emphasized the computational speed of this branch-and-bound algorithm over ILP methods. However, this could very likely be because the study implemented the branch-and-bound algorithm in C++, and we implemented it in python. It may also be the case that our incorporation of producing multiple solutions greatly increased our runtime, or that including the vertex coloring instead of removing it in favor of using our known multipartite graph information may have lead to unnecessary calculations (though an essential calculation for an arbitrary graph).

The very short runtime of the loopy belief propagation, as well as its slightly inferior results, also lined up with our expectations, since the loopy belief propagation is a heuristic and not an exact search. In fact, it was quite impressive that even with such a short runtime (18 and 19 seconds for *cspA* and *hipB*, respectively), we were still able to generate the same best solutions as the exact motif finders, and even in the case of *hipB*, able to find the same collection of solutions as our exact methods. However, our need for multiple solutions was emphasized in the loopy belief propagation search of *oxyR*, since our “best” solution found for *oxyR* was significantly worse than the best solution found for our exact motif finders. It could be the case here that the flaw in our results confirmed the flaw of using MAP inference using loopy belief propagation, discussed in Section 3.3.2.

Despite having a longer motif length l , we found that our computational time increased rather than decreased, even though a larger motif length l would give us a smaller model to solve. Nevertheless, it is not impossible for smaller models to be more difficult to solve than larger models. Because our computational time increased greatly with longer motifs, using the branch-and-bound algorithm to generate results for the binding sites associated with *oxyR* proved to take too long to finish (well over a couple of days). We also found that trying to discover motifs for a larger number of binding sites greatly increased our computational time. When we attempted to perform a motif discovery on an alignment of eleven sequences (such as in the case of the binding sites associated with the transcriptional regulator *narL*), we found that the runtime was too long to generate results

for this study, taking well over several days. We found that even generating solutions for 5 or 6 binding sites (such as in the case of the binding sites associated with *cytR* and *araC*, respectively) also took a very long time, well over a couple of days.

4.4 Limitations

Implementing our approaches in python rather than C or C++ certainly may have lead to limitations in generating some of our results, since C or C++ would be preferable in doing large calculations like the ones performed in motif discovery (Prechelt, 2000). Perhaps implementing our approaches in python would make no difference for our ILP formulation since our solvers, Gurobi and Google OR-Tools, are implemented in C and C++, respectively, and constructing the model in python does not lead to a large penalty. However, it could make a huge computational difference for our algorithms implemented completely in python, like in the case of our branch-and-bound algorithm and our loopy belief propagation.

Although we were able to detect some overlapping regions between our solutions and our transcriptional regulators (particularly for transcriptional regulators with higher ls), ultimately we were not able to detect the correct transcriptional regulators from our binding sites (since the scores for these transcriptional regulators were often lower than any solution found by our three approaches). This could be because of one of three reasons. One reason could be that more attention was needed for the method of calculation of the scores of our l -long segments, so that the transcriptional regulators could have the best scores or close-to-best scores of the system. This may involve doing some kind of pre-calculation of scoring that would align better with natural expectations, which opens up new avenues of research and exploration. The other reason may be that the models outlined here, while intuitive representations of motif discovery, may not conform to the reality of DNA mutations. Another, perhaps more hopeful reason may be that the approaches used here were able to find a new motifs not yet discovered, and that perhaps this model may be used to unlock more motifs not yet discovered.

Chapter 5

Conclusion

5.1 Findings

By simplifying the motif discovery problem as a multiple sequence alignment problem in which the motifs are not gapped and the lengths of the motifs are known, we are able to intuitively design two equivalent graphical models that are often unconnected in literature: the MEWCP and the MAP-MRF. The implications of this connection is that any approach used to find the solution for one graph problem can be used to find the solution for the other graph problem and *vice versa*. This connection can be expanded outwards to relate an entire collection of approaches to another collection of approaches.

We were also able to not only implement three separate motif discovery approaches (ILP formulation, branch-and-bound algorithm, and loopy belief propagation) in python, but also ensure that each of the implementations produced multiple solutions to make up for the weakness of modeling the motif discovery problem without gaps and with a known motif length, and to allow a greater likelihood that the “true” solution will be detected by any one of these approaches. Though we found that our success in finding the motifs was limited, we are still hopeful that tweaking some of the details of our implementation may lead to better results in the future.

5.2 Future endeavors

Certainly a more detailed exploration into a proper scoring method for scoring our motifs would be of great interest. Our exact approaches (ILP formulation and the branch-and bound algorithm) were able to find the optimal solutions to the MEWCP, but they were not able to detect the motifs because the motifs themselves had a lower score than the optimal solution. Another exploration that would be of interest would be to remove the vertex coloring from the branch-and-bound algorithm in the hopes of cutting down the computational time. Of course, running these three approaches on a more powerful machine (perhaps a machine with several CPUs running at once, or using Compute Canada services) to generate results for larger sequence alignments would also be extremely beneficial, since we could learn more about how each of these approaches work, and better study the affect of changing each parameter (number of sequences, motif length, and number of solutions generated) on computational time.

Perhaps a deeper exploration into these topics would involve designing a strategy for converting a cyclic graph into an equivalent tree graph in order to obtain better results using belief propagation. We could also try to use the well-known Markov chain Monte Carlo approach that is used to solve Markov random fields as a heuristic for an ILP formulation. Since there exists previous research exploring the connection between ILP and belief propagation through the weighted matching problem (Gelfand et al., 2013), we could further explore the connection between MEWCP and MAP-MRF to see if there exists an even broader graph problem connection that encompasses the Markov random field.

Bibliography

Bailey, T. L.

2011. DREME: motif discovery in transcription factor ChIP-seq data. Bioinformatics, 27(12):1653–1659.

Bailey, T. L., J. Johnson, C. E. Grant, and W. S. Noble

2015. The MEME suite. Nucleic acids research, 43(W1):W39–W49.

Bertsekas, D. P.

1997. Nonlinear programming. Journal of the Operational Research Society, 48(3):334–334.

Bertsimas, D. and J. N. Tsitsiklis

1997. Introduction to linear optimization, volume 6. Athena Scientific Belmont, MA.

Chartrand, G.

1977. Introductory graph theory. Courier Corporation.

Chen, X., L. Guo, Z. Fan, and T. Jiang

2008. W-AlignACE: an improved Gibbs sampling algorithm based on more accurate position weight matrices learned from sequence and gene expression/ChIP-chip data. Bioinformatics, 24(9):1121–1128.

Cornuéjols, G.

2008. Valid inequalities for mixed integer linear programs. Mathematical Programming, 112(1):3–44.

Deonier, R. C., S. Tavaré, and M. S. Waterman

2005. Computational genome analysis: an introduction. Springer Science & Business Media.

Gelfand, A. E., J. Shin, and M. Chertkov

2013. Belief propagation for linear programming. In 2013 IEEE International Symposium on Information Theory, Pp. 2249–2253. IEEE.

Gilbert, D.

2003. Sequence File Format Conversion with Command-Line Readseq. Current Protocols in Bioinformatics, (1):A–1E.

González-Álvarez, D. L., M. A. Vega-Rodríguez, J. A. Gómez-Pulido, and J. M. Sánchez-Pérez

2011. Finding motifs in DNA sequences applying a multiobjective artificial bee colony (MOABC) algorithm. In European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Pp. 89–100. Springer.

Gurobi Optimization, L.

2021. Gurobi Optimizer Reference Manual.

Gusfield, D.

2019. Integer linear programming in computational biology: Overview of ILP, and new results for traveling salesman problems in biology. In Bioinformatics and Phylogenetics, Pp. 373–404. Springer.

Hashim, F., M. S. Mabrouk, and W. Al-Atabany

2017. GWOMF: Grey Wolf Optimization for motif finding. In 2017 13th international computer engineering conference (ICENCO), Pp. 141–146. IEEE.

Hashim, F. A., M. S. Mabrouk, and W. Al-Atabany

2019. Review of different sequence motif finding algorithms. Avicenna journal of medical biotechnology, 11(2):130.

Kahvejian, A., J. Quackenbush, and J. F. Thompson

2008. What would you do if you could sequence everything? Nature biotechnology, 26(10):1125–1133.

Koller, D. and N. Friedman

2009. Probabilistic graphical models: principles and techniques. MIT press.

Land, A. H. and A. G. Doig

1960. An Automatic Method of Solving Discrete Programming Problems. Econometrica, 28(3):497–520.

Manning, T., R. D. Sleator, and P. Walsh

2013. Naturally selecting solutions: the use of genetic algorithms in bioinformatics. Bioengineered, 4(5):266–278.

McGuire, A. M., J. D. Hughes, and G. M. Church

2000. Conservation of DNA regulatory motifs and discovery of new motifs in microbial genomes. Genome research, 10(6):744–757.

Meyer, M. and M. Kircher

2010. Illumina sequencing library preparation for highly multiplexed target capture and sequencing. Cold Spring Harbor Protocols, 2010(6):pdb–prot5448.

Morrison, D. R., S. H. Jacobson, J. J. Sauppe, and E. C. Sewell

2016. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization, 19:79–102.

Osada, R., E. Zaslavsky, and M. Singh

2004. Comparative analysis of methods for representing and searching for transcription factor binding sites. Bioinformatics, 20(18):3516–3525.

Pavesi, G., P. Mereghetti, G. Mauri, and G. Pesole

2004. Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. Nucleic acids research, 32(suppl_2):W199–W203.

Perron, L. and V. Furnon

2019. OR-Tools.

Poinar, H. N., C. Schwarz, J. Qi, B. Shapiro, R. D. MacPhee, B. Buigues, A. Tikhonov, D. H. Huson, L. P. Tomsho, A. Auch, et al.

2006. Metagenomics to paleogenomics: large-scale sequencing of mammoth DNA. Science, 311(5759):392–394.

Prechelt, L.

2000. An empirical comparison of seven programming languages. Computer, 33(10):23–29.

Rader, D. J.

2010. Deterministic operations research: models and methods in linear optimization. John Wiley & Sons.

Robison, K., A. M. McGuire, and G. M. Church

1998. A comprehensive library of DNA-binding site matrices for 55 proteins applied to the complete Escherichia coli K-12 genome. Journal of molecular biology, 284(2):241–254.

Saito, H., S. Kashida, T. Inoue, and K. Shiba

2007. The role of peptide motifs in the evolution of a protein network. Nucleic acids research, 35(19):6357–6366.

Schloss, J. A., R. A. Gibbs, V. B. Makhijani, and A. Marziali

2020. Cultivating DNA sequencing technology after the human genome project. Annual review of genomics and human genetics, 21.

Shimizu, S., K. Yamaguchi, and S. Masuda

2020. A maximum edge-weight clique extraction algorithm based on branch-and-bound. Discrete Optimization, 37:100583.

Thijs, G., M. Lescot, K. Marchal, S. Rombauts, B. De Moor, P. Rouze, and Y. Moreau

2001. A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. Bioinformatics, 17(12):1113–1122.

Thompson, W., E. C. Rouchka, and C. E. Lawrence

2003. Gibbs Recursive Sampler: finding transcription factor binding sites. Nucleic acids research, 31(13):3580–3585.

Williams, H. P.

2013. Model building in mathematical programming. John Wiley & Sons.

Yanover, C., M. Singh, and E. Zaslavsky

2009. M are better than one: an ensemble-based motif finder and its application to regulatory element prediction. Bioinformatics, 25(7):868–874.

Yanover, C. and Y. Weiss

2003. Finding the M most probable configurations using loopy belief propagation. Advances in neural information processing systems, 16:289–296.

Zambelli, F., G. Pesole, and G. Pavesi

2012. Motif discovery and transcription factor binding sites before and after the next-generation sequencing era. Briefings in bioinformatics, 14(2):225–237.

Zaslavsky, E. and M. Singh

2006. A combinatorial optimization approach for diverse motif finding applications. Algorithms for Molecular Biology, 1(1):13.