Abstract

Message Passing Algorithms for Optimization

Nicholas Robert Ruozzi

2011

The max-product algorithm, which attempts to compute the most probable assignment (MAP) of a given probability distribution via a distributed, local message passing scheme, has recently found applications in convex minimization and combinatorial optimization. Unfortunately, the max-product algorithm is not guaranteed to converge and, even if it does, is not guaranteed to produce the MAP assignment. Many alternative message passing schemes have been proposed to overcome these difficulties (e.g. TRMP, MPLP, max-sum diffusion). These algorithms can be viewed as coordinate ascent schemes over different duals of a linear programming formulation of the MAP problem. If these algorithms converge to a unique assignment, then this assignment is guaranteed to be the maximum of the objective function. Although these algorithms provide stronger guarantees than max-product upon convergence, they do not always converge to a unique assignment, and in some instances, the dual optimization problem that results provides a trivial upper bound on the maximizing assignment.

In this work, we provide a systematic study of message passing algorithms for the related problem of minimizing an arbitrary real-valued objective function: from graphical models to reparameterization, reparameterization to lower bounds, and from lower bounds to convergent message passing algorithms. We generalize the known results by providing conditions under which the assignments produced by message passing algorithms can correspond to local and global optima, by providing a combinatorial characterization of when these message passing schemes can actually solve the minimization problem, and by providing a new convergent and correct message passing algorithm, called the splitting algorithm, that contains many of the known convergent message passing algorithms as a special case.

These ideas allow us to expand the usefulness of the splitting algorithm beyond the limits of other message passing algorithms. We show that there are examples of convex minimization problems on which convergent message passing algorithms fail to produce a minimizing assignment but that the splitting algorithm succeeds. We use graph covers and our conditions for local optimality to provide conditions under which the splitting algorithm can be used to solve general convex (as well
as submodular) minimization problems. These observations lead us to a generalization of diagonal dominance for arbitrary convex functions.
Message Passing Algorithms for Optimization

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Nicholas Robert Ruozzi

Dissertation Director: Sekhar Tatikonda

December 2011
Acknowledgments

I am greatly indebted to my advisor Sekhar Tatikonda not only for his advice but also for my interest in graphical models.

I would like to thank Dan Spielman and Joan Feigenbaum for their guidance and support throughout my graduate school career. For my decision to attend graduate school, I owe many thanks to Graeme Bailey and Dexter Kozen.

I would also like to thank my friends and colleagues, without whom I would never have enjoyed my time at Yale: Aaron Johnson, Lev Reyzin, Blair Kenney, Nikhil Srivastava, Justin Thaler, Steve Eckel, Justin Hart, Chris Milan, Karen Paczkowski, Shannon Stewart, and Paul Vierthaler.

Finally, I would like to thank my family for their love and support.
To mom, dad, and Anthony
# Contents

1 Introduction

1.1 Inference and Graphical Models ........................................... 2
1.2 Local Message Passing Algorithms ....................................... 3
1.3 Thesis Overview ................................................................. 4
    1.3.1 The Min-Sum Algorithm ............................................... 5
    1.3.2 Lower Bounds and Reparameterization ............................. 5
    1.3.3 Local Message Passing Algorithms ................................ 6
    1.3.4 Graph Covers ............................................................ 6
    1.3.5 Convex Optimization ................................................. 7

2 Preliminaries

2.1 Graph Theory Basics .......................................................... 8
2.2 Message Passing and Optimization ........................................ 9
    2.2.1 Dynamic Programming ................................................ 10
    2.2.2 Factorizations and Factor Graphs .................................. 12
    2.2.3 The Min-Sum Algorithm ............................................. 13

3 The Min-Sum Algorithm: Convergence and Correctness ............. 21

3.1 Shortest Paths ................................................................. 22
    3.1.1 Single-Pair Shortest Path ........................................... 22
    3.1.2 Single-Source Shortest Paths ..................................... 27
3.2 Transitive Closure ............................................................ 29
3.3 Maximum Weight Matchings ................................................. 30
3.4 Weaknesses of the Min-Sum Algorithm .................................. 32
4 Reparameterizations
4.1 Admissibility and the Min-Sum Algorithm
4.2 The Splitting Reparameterization
4.3 Lower Bounds
   4.3.1 Lower Bounds and Reparameterization
   4.3.2 Lower Bounds and the MAP LP
4.4 Factorizations and Message Reparameterizations

5 Lower Bounds and Optimality
5.1 Local Optimality
5.2 Global Optimality
5.3 Other Notions of Optimality

6 The Splitting Algorithm
6.1 The Algorithm
   6.1.1 Computation Trees
6.2 Convergence
   6.2.1 A Simple Asynchronous Algorithm
   6.2.2 Synchronous Convergence
   6.2.3 Subgradient Methods
6.3 An Alternative Derivation

7 Graph Covers and Indistinguishability
7.1 Graph Covers
7.2 Factorizations and Graph Covers
7.3 Pseudocodewords
7.4 Graph Covers and the MAP LP
   7.4.1 Maxima of the Lower Bound

8 Pairwise Graphical Models
8.1 Asynchronous and Synchronous Convergence
8.2 Pairwise Binary Graphical Models
   8.2.1 Partial Solutions
8.2.2 Applications .................................................. 84

9 Other Convergent and Correct Message Passing Algorithms 88

9.1 Local Message Passing Algorithms .................................. 89
  9.1.1 TRW-S and TRMP ............................................. 89

9.2 Non-local Message Passing ........................................... 91
  9.2.1 MPLP ............................................................... 91
  9.2.2 Max-Sum Diffusion ............................................. 92
  9.2.3 Norm-Product .................................................. 94

10 Convex Optimization .................................................. 95

10.1 Quadratic Minimization .............................................. 96
  10.1.1 The Gauss-Seidel and Jacobi Algorithms ......................... 96
  10.1.2 Minimization via the Splitting Algorithm ......................... 98

10.2 General Convex Optimization ....................................... 108

10.3 Submodular Functions ............................................... 111
# List of Figures

2.1 Examples of graphs. ......................................................... 9  
2.2 Example of the maximum weight independent set problem on a tree. ........ 11  
2.3 Example factor graph. ..................................................... 13  
2.4 Example of a computation tree. ........................................ 18  

3.1 A computation tree produced by the min-sum algorithm for the single-pair shortest path problem. ......................................................... 24  
3.2 Illustration of the construction in Theorem 3.1.1. ................................. 25  
3.3 An example of a transitive closure. .............................................. 30  
3.4 An example of the maximum weight matching problem. ............................ 31  
3.5 Example problems for which the min-sum algorithm performs poorly. ............ 32  

5.1 A factor graph for which the two-hop neighborhood of every node is not a tree. .... 52  

6.1 Computation trees of the splitting algorithm. ........................................ 57  
6.2 Splitting the factor nodes of a factor graph. ......................................... 65  
6.3 Splitting the variable nodes of a factor graph. ....................................... 66  

7.1 An example of a graph cover. .................................................... 69  
7.2 An example of a maximum weight independent set problem whose graph cover contains a different solution than the original problem. .............................. 72  

8.1 Simplification of pairwise factor graphs. ............................................ 77  
8.2 Kronecker double cover of a pairwise factor graph. .................................. 80  

10.1 A positive definite matrix that is covered by a matrix with negative eigenvalues. .. 100
10.2 A positive definite matrix for which the variances in the min-sum algorithm converge but the means do not. ................................................................. 104

10.3 A comparison of the performance of the min-sum algorithm and the splitting algo-
       rithm for the quadratic minimization problem. ................................. 107
List of Algorithms

1 Bellman-Ford Algorithm ................................................. 29
2 Synchronous Splitting Algorithm ...................................... 55
3 Cyclic Coordinate Ascent ............................................... 58
4 Asynchronous Splitting Algorithm .................................... 58
5 Damped Synchronous Splitting Algorithm ........................... 62
6 Subgradient Ascent ........................................................ 64
7 Pairwise Synchronous Splitting Algorithm ......................... 78
8 Pairwise Asynchronous Splitting Algorithm ......................... 79
9 Bipartite Asynchronous Splitting Algorithm ....................... 80
10 Jacobi Iteration ........................................................... 97
11 Gauss-Seidel Iteration .................................................... 97
Chapter 1

Introduction

Let \( f : \mathcal{X} \to \mathbb{R} \). Consider the problem of minimizing this function (i.e. finding an \( x^* \in \mathcal{X} \) such that \( f(x^*) \leq f(x) \) for all \( x \in x^* \)). Efficiently solving this minimization problem for different objective functions is a central problem in computer science, and many different algorithms have been designed, depending on the application, for this purpose. In this thesis, we study the properties of one such algorithm: the min-sum message passing algorithm. This algorithm, and variants of it, have been rediscovered many times in many different communities [3]: the Viterbi algorithm, the Baum-Welch algorithm, the Kalman filter, iterative decoding algorithms for error-correcting codes, the fast Fourier transform on finite Abelian groups, the belief revision algorithm, the belief propagation algorithm, and many more.

More generally, each of these algorithms can be cast as an optimization problem over a graphical model: a combinatorial representation of the relationships among the variables in the objective function, \( f \), that consists of a set nodes and a set of edges that interconnect them. The min-sum algorithm is then a message passing algorithm over the graphical model (nodes pass messages to other nodes across the edges of the graphical model) whose goal is to minimize the objective function. The min-sum algorithm, and others like it, are known to perform well in practice [33], but a complete theoretical characterization of the algorithm’s behavior on arbitrary objective functions has yet to be found. Our goal in this work is to narrow the gap between theory and practice: not only do we want to better understand the behavior of the min-sum algorithm, but we also want to design alternative message passing algorithms that provably outperform it.
1.1 Inference and Graphical Models

Historically, the study of graphical models has focused on statistical inference. Let \( p(x_1, \ldots, x_n) \) be a probability distribution. Because of their importance in a wide variety of applications, two inference problems are typically studied in this context: the maximum a posteriori (MAP) estimation problem and the computation of marginal probabilities.

A marginal of a probability distribution \( p \) is the function obtained by fixing the value of one of the random variables and summing out over the remaining variables. For example, the marginal function, \( p_i(x_i) \), corresponding to the \( i^{th} \) variable is given by:

\[
p_i(x_i) = \sum_{x_j \neq x_i} p(x'_1, \ldots, x'_n)
\]

Computing the marginals of a given probability distribution is, in general, an expensive operation. However, when the probability distribution has some additional structure, we may be able to compute the marginals much faster.

**Example 1.1.1 (Efficient marginalization).**

Suppose \( p(x_1, x_2, x_3) \) is a probability distribution over three random variables that take values in the set \( \{1, \ldots, k\} \). Further, suppose that \( p \) can be written as a product of functions as follows:

\[
p(x_1, x_2, x_3) = q_{12}(x_1, x_2)q_{13}(x_1, x_3)
\]

Now, consider computing the marginal of \( p \) for the variable \( x_1 \):

\[
p_1(x_1) = \sum_{x_2} \sum_{x_3} p(x_1, x_2, x_3)
\]

As \( x_2 \) and \( x_3 \) can each take one of \( k \) different values, this summation contains \( k^2 \) distinct terms for each fixed value of \( x_1 \). However, if we exploit the observation that \( p \) can be written as a product, we can rewrite the summation as

\[
p_1(x_1) = \sum_{x_2} q_{12}(x_1, x_2) q_{13}(x_1, x_3)
\]

\[
= \sum_{x_2} q_{12}(x_1, x_2) \left[ \sum_{x_3} q_{13}(x_1, x_3) \right]
\]

which only requires summing \( 2k \) distinct terms for each fixed value of \( x_1 \).
As an alternative to computing the marginals, we may also be interested in computing the most likely configuration of our joint probability distribution. This problem is typically referred to as maximum a posteriori estimation, MAP estimation for short, and is a solution to the following optimization problem:

\[
x^* \in \arg \max_x p(x_1, \ldots, x_n)
\] (1.6)

Again, notice that, like summation, computing the maximum can be computationally expensive, but if the probability distribution can be written as a product of smaller functions, then we may be able to compute the maximum more efficiently. A similar idea can be applied over any semiring [3] (e.g. sum and product, max and product, min and sum, etc.).

Graphical models describe the relationship between the pieces of the decomposition, called potentials, of a particular probability distribution, and in certain instances, we can exploit the structure of the graphical model to efficiently compute both the marginals and the MAP estimate.

### 1.2 Local Message Passing Algorithms

Belief propagation was originally formulated by Judea Pearl to solve inference problems in Bayesian networks [34]. Pearl demonstrated that, when the graphical model is tree structured, a simple, distributed message passing algorithm, dubbed "belief propagation", is guaranteed to converge to the exact marginals of the input probability distribution. If the belief propagation algorithm is run on an arbitrary graphical model (i.e. one that may contain cycles), then neither convergence nor correctness are guaranteed. However, in many settings, the "loopy" belief propagation algorithm often produces good approximations to the true marginals [33].

Similarly, Pearl proposed an algorithm for MAP estimation that he dubbed "belief revision". The optimization analogs of belief propagation are more commonly known as the max-product and min-sum algorithms. They have similar guarantees to the belief propagation algorithm: they produce the correct MAP estimate when then graphical model is a tree, and may or may not produce the correct solution when run on graphs with cycles. In this work, we focus primarily on variants of the min-sum algorithm: a local message passing scheme designed to find a global minimum of an objective function that can be written as a sum of functions each of which depend
on a subset of the problem variables. As such, the restriction to probability distributions in the above discussion is artificial. Instead, we will be concerned with the general optimization problem: given an objective function that achieves its minimum over its domain, find an assignment to the problem variables that minimizes this objective function.

Since the development of the above message passing algorithms, many more distributed message passing algorithms have been proposed for these inference problems. In this work, we will try to understand what causes the min-sum algorithm to fail to converge to a correct solution in an attempt to design message passing algorithms that provide stronger guarantees on both convergence and correctness.

1.3 Thesis Overview

In this thesis, we unify, generalize, and extend the known results for local message passing algorithms. The main contributions of this thesis are as follows:

1. A new distributed, local message passing algorithm known as the splitting algorithm which contains many of the other convergent and correct message passing algorithms as a special case. This algorithm is more appropriately thought of as a family of message passing algorithms, and we provide examples that highlight the strengths and weakness of different choices of the parameters.

2. Conditions under which the estimate produced by the splitting algorithm is guaranteed to be a global optima.

3. Conditions under which the estimate produced by the splitting algorithm is guaranteed to be a local optima.

4. A deeper understanding of the relationships between graph covers and the convergence and correctness of local message passing algorithms that applies not only to convergent and correct message passing algorithms, but also to other message passing schemes that only guarantee local optimality. This understanding allows us to provide necessary conditions for these algorithms to converge to the correct solution.

5. An improved algorithm for the quadratic minimization problem that leverages the advantages of the splitting algorithm to solve problem instances for which other known message passing
algorithms would fail.

The chapters of this thesis are designed to be read sequentially: each chapter draws on the results and insights of the previous chapters in order to describe the process of constructing general purpose optimization techniques based on message passing. Many of the results that this thesis builds on were developed for specific problems or in specific subareas. We attempt to unify these results into a comprehensive discussion based on reparameterizations and lower bounds. The results presented in this thesis have appeared in a variety of publications: [37][40][39][38].

1.3.1 The Min-Sum Algorithm

We begin by reviewing the definitions and basic properties of the min-sum algorithm and factorizations in Chapter 2. In Chapter 3, we examine the limits of the min-sum algorithm by example. We show that by analyzing the computation trees produced by the min-sum algorithm we can show that the min-sum algorithm is guaranteed to compute the correct minimizing assignment for the single-pair shortest path problem, the single-source shortest path problem, and the transitive closure problem. These results stand in contrast to the well-studied behavior of the min-sum algorithm for the maximum weight matching problem and maximum weight independent set problems for which the min-sum algorithm does not converge to the correct solution in all instances [41, 41]. These observations suggest a close link between the computational complexity of a problem and the ability of the min-sum algorithm to solve the problem.

1.3.2 Lower Bounds and Reparameterization

With an understanding of the strengths and limitations of the min-sum algorithm, we begin studying ways in which to improve the algorithm. In Chapter 4, we take the first step towards convergent algorithms by studying reparameterizations of the objective function. Upon convergence, the min-sum algorithm produces a reparameterization (an alternate factorization of the objective function as a sum of functions). We show how to produce more general reparameterizations of the objective function and a specific reparameterization called the splitting reparameterization. The splitting reparameterization is actually a family of reparameterizations that is parameterized by a vector of non-zero reals. From these reparameterizations we show how to derive concave lower bounds on the objective function in one of two ways: first by using the observation that the minimum of a sum is lower bounded by the sum of minimums, and second for real-valued objective functions over a finite
state space by using duality and a linear programming formulation of the optimization problem. For real-valued objective functions, we show that any two factorizations of the objective function can be reparameterized into each other via message reparameterizations. This generalizes a similar result [21] for pairwise factorizations.

1.3.3 Local Message Passing Algorithms

In addition to producing a reparameterization of the objective function, the min-sum algorithm also produces a vector of min-consistent beliefs. An estimate of the optimal assignment is then extracted from these beliefs. By analogy, we can define beliefs for the splitting reparameterization. In Chapter 5, we investigate the properties of min-consistent beliefs that reparameterize the objective function. We provide conditions on the parameter vector under which the estimate is guaranteed to be a global minimum of the objective function. Further, we provide novel conditions under which the estimate need only correspond to a local optima of the objective function.

In Chapter 6, we introduce the splitting algorithm. We show that there always exists a choice of the parameter vector such that an asynchronous variant of the splitting algorithm corresponds to a block coordinate ascent scheme for a specific concave lower bound derived from the splitting reparameterization. For this choice of the parameter vector, if the algorithm converges and the beliefs are locally decodable, then the estimate produced by the splitting algorithm is guaranteed to minimize the objective function. We conclude this chapter with a discussion of how convergent synchronous algorithms can be derived from the convergent asynchronous algorithm.

1.3.4 Graph Covers

In Chapter 7, we use the observation that local message passing algorithms cannot distinguish two factor graphs with the same local structure to provide necessary conditions for the splitting (or any bound maximizing algorithm) to converge to a locally decodable estimate. For real-valued objective functions over finite state spaces, this is accomplished by showing that there is a direct correspondence between assignments on graph covers and rational feasible points of the MAP LP (a result that was proven only for MAP LPs generated by coding problems [11]). We explain how to extend this analysis to arbitrary objective functions. This notion of indistinguishability based on graph covers has far reaching consequences for the splitting algorithm, the min-sum algorithm, and other iterative methods (e.g. coordinate ascent).
In addition to the above discussion for arbitrary factor graphs, we show, in Chapter 8, that for the special case of pairwise binary factorizations, we can apply our observations to obtain necessary and sufficient conditions for the fixed points of the splitting algorithm to be locally decodable which generalize the convergence and correctness results of [47] for this special case. Further, we show that, in this case, the fixed point solutions of the min-sum algorithm all correspond to a local optimum on some 2-cover of the factor graph.

### 1.3.5 Convex Optimization

Much of the work on convex function minimization, with respect to the min-sum algorithm, has focused on the quadratic minimization problem [32, 29]. For scaled diagonally dominant matrices, the min-sum algorithm is guaranteed to converge, but for problems that are positive definite, but not scaled diagonally dominant the algorithm may or may not converge [29]. In Chapter 10, we show that the splitting algorithm outperforms the min-sum algorithm for this problem: we show how certain choices of the parameter vector allow us to solve problems for which the standard min-sum algorithm fails, and we experimentally demonstrate that there exist choices of the parameter vector that allow the splitting algorithm to converge in fewer iterations than the min-sum algorithm. Throughout this chapter, we discuss the relationship between the splitting algorithm and other iterative algorithms for quadratic minimization. We show that the same tools that we have developed for the splitting algorithm can be applied to understand these other iterative algorithms as well and that, under certain conditions, the splitting algorithm can be reduced to these methods. With the results for the quadratic minimization problem as a guide, we discuss the general problem of minimizing a convex function and the related problem of minimizing a submodular function via the splitting algorithm.
Chapter 2

Preliminaries

In this chapter, we review the background material necessary for this thesis. We begin by reviewing basic terminology from graph theory in Section 2.1 and then we introduce factorizations, factor graphs, and the min-sum algorithm in Section 2.2.

2.1 Graph Theory Basics

A graph $G = (V,E)$ is composed of a set of vertices or nodes, $V$, and a set of edges, $E \subseteq V \times V$. The edges can either be directed or undirected. In the directed case, the edge $(i,j)$ indicates that there is a directed edge from $i$ to $j$ but not necessarily the other way around. For undirected graphs, $(i,j) \in E$ indicates that there is an undirected edge between the nodes $i$ and $j$. We can visualize undirected graphs by drawing a node for each $i \in V$ and drawing an edge connecting the node $i$ to the node $j$ whenever $(i,j) \in E$. For directed graphs, the edges are drawn with an arrow to indicated the directionality of the arrows. See Figure 2.1 for examples of directed and undirected graphs. The graphs in this work should be assumed to be undirected unless otherwise specified.

A subgraph $H = (V^H, E^H)$ of $G$ is a graph formed by taking a subset of the vertices of $G$ and a subset of the edges connecting those vertices. For example, $H = (\{1, 2\}, \{(1, 2)\})$ is a subgraph of the graph in Figure 2.1 (a).

For an undirected graph $G = (V,E)$, two nodes $i \in V$ and $j \in V$ are neighbors if there is an edge joining $i$ and $j$ in $E$ (i.e. $(i,j) \in E$). Throughout this work, we use $\partial i$ to denote the neighbors
of $i$:

$$\partial i = \{ j \in V | (i, j) \in E \}$$  \hfill (2.1)

The degree of a vertex is equal to the number of its neighbors in the graph. In our notation, the degree of a vertex $i$ is given by $|\partial i|$.

A path in the graph $G$, is a subset of the vertices, $v_1, \ldots, v_k$, such that there is an edge from $v_i$ to $v_{i+1}$ for each $i \in \{1, \ldots, k\}$. In this case, we say that there is a path from $v_1$ to $v_k$. For directed graphs, each edge of the path must be directed from $v_i$ to $v_{i+1}$. Similarly, a subset of the vertices, $v_1, \ldots, v_k$, forms a cycle if there is a path from $v_1$ to $v_k$ and there is an edge from $v_k$ to $v_1$.

We say that an undirected graph is connected if for all $i, j \in V$ there is a path from $i$ to $j$. A directed graph is connected if it is connected when treated as an undirected graph (i.e. ignoring the arrows). A directed graph is strongly connected if there is a directed path between every pair of vertices in the graph.

A graph is a tree when it is connected and no subset of the vertices forms a cycle. A graph is bipartite when the vertex set can be partitioned into two sets $A$ and $B$ such that there are no edges between vertices of the same set. Every tree is a bipartite graph. For a connected graph $G = (V, E)$, a spanning tree is a subgraph $H = (V, E^H)$ such that $H$ is a tree. Trees are often oriented by a special node called the root node. Let $T = (V, E)$ be a tree rooted at the node $r \in V$, and fix $i \in V$ such that $i \neq r$. There exists a unique path from $i$ to the root, $r$. The first node along this path, $j$, is called the parent of $i$. Equivalently, we say that $i$ is a child of $j$.

### 2.2 Message Passing and Optimization

In this section, we review the problem formulation, necessary terminology, and basic results concerning the min-sum algorithm. Let $f : \prod_i \mathcal{X}_i \rightarrow \mathbb{R} \cup \{\infty\}$, where each $\mathcal{X}_i$ is an arbitrary set
Throughout this paper, we will be interested in finding an element 
\((x_1, ..., x_n) \in \prod_i \mathcal{X}_i\) that minimizes \(f\), and as such, we will assume that there is such an element:

**Assumption 1.** \(\exists x^* \in \prod_i \mathcal{X}_i\) such that \(f(x^*) = \inf_x f(x)\).

We note that we will in general allow \(f\) to take the value \(\infty\) over its domain. However, as we will see in subsequent chapters, some results will only apply when \(f\) is a proper, real-valued function (i.e. \(f\) does not take the value \(\infty\) or \(-\infty\) for any element in its domain).

For an arbitrary function, computing the minimum may be computationally expensive, especially if \(n\) is large. A typical scientific application may involve hundreds of thousands of variables and potential functions, and storing the entire problem on one computer may be difficult, if not impossible. In other applications, such as sensor networks, processing power and storage are limited. Because local message passing algorithms like the min-sum algorithm are decentralized and distributed, they can operate on scales at which typical algorithms would be impractical.

Although we will discuss algorithms for the minimization problem, some applications have a more natural formulation as maximization problems. In these instances, we can use the equivalence \(\max_x f(x) = -\min_x -f(x)\) in order to convert the maximization problem into a minimization problem.

### 2.2.1 Dynamic Programming

As a motivation for studying the general problem, we begin by examining the connection between message passing algorithms and more standard notions in computer science.

A common problem solving strategy in computer science is to break apart large problems into smaller subproblems, solve the subproblems, and then combine the solutions of the subproblems into a solution to the larger problem. To solve each subproblem, we can recursively employ a similar strategy. This style of problem solving is typically referred to as divide-and-conquer.

Closely related to divide-and-conquer is the notion of dynamic programming. In dynamic programming, we start with an optimization problem that can expressed as an optimization over subproblems via some recurrence. Naively computing the recurrence can be costly; we may end up solving many of the same subproblems over and over again. Instead, we employ a bottom up strategy: we create a table containing the solutions to all of the subproblems by solving the smallest subproblems first, recording their solutions in the table, and then using these solutions to compute the solutions to the next smallest subproblems. Local message passing algorithms such as belief
Figure 2.2: Example of the maximum weight independent set problem on a tree. The number inside of each node corresponds to the weight of that node.

propagation and min-sum are, in many ways, a natural by-product of the same problem solving strategies underlying dynamic programming. This connection is best illustrated by example:

Example 2.2.1 (Maximum weight independent set on a tree).
Let $G = (V, E)$ be a graph with associated weights $w_i$ for each node $i \in V$. A set, $S \subseteq V$, of the vertices forms an independent set if no two vertices in the subset are joined by an edge in $E$. The weight of a set $S$, written $w(S)$, is the sum of the weights of all of the vertices in $S$. The maximum weight independent set problem is to find an independent set in $G$ of maximum weight.

Computing the maximum weight independent set for an arbitrary graph is known to be NP-hard. However, in the special case that the graph $G$ is a tree, the problem can be solved in polynomial time using standard dynamic programming techniques. To see this, let $T$ be a tree rooted at a node $r$. Define the function $mwis(i, \text{inSet})$ to be the value of the maximum weight independent set in the subtree rooted at $i$ formed by the descendants (children of $i$, $i$’s children’s children, etc.) of node $i$ where inSet is either zero or one to indicate whether node $i$ should be taken as part of the independent set. The function $mwis$ is defined via the following recursion:

$$mwis(i, 1) = w_i + \sum_{c \text{ child of } i} mwis(c, 0)$$  \hspace{1cm} (2.2)

$$mwis(i, 0) = \sum_{c \text{ child of } i} \max\{mwis(c, 0), mwis(c, 1)\}$$  \hspace{1cm} (2.3)

We have that $w(S^*) = \max\{mwis(r, 0), mwis(r, 1)\}$ for any maximum weight independent set, $S^*$, on $T$. Notice that $mwis(i, 1)$ and $mwis(i, 0)$ only depend on the value of $mwis$ for the children of node $i$. To see how we might turn this into a distributed algorithm, suppose we could perform computations at each node of the graph. We can have each leaf node $l$ compute $mwis(l, 1)$ and $mwis(l, 0)$ and pass these results to their parents. The nodes directly above the leaf nodes can then perform a similar computation and pass the results to their parents and so on. Formally, let $p(i)$ be the parent of node $i$ in the tree, and define a vector of messages, $m$, by setting $m_{i \rightarrow p(i)}(1) =$
The algorithm described in Example 2.2.1 exactly computes the maximum weight independent set on a tree. In some sense, the remainder of this work is an attempt to answer the following question: if the graph is not a tree, under what conditions can we use a similar message passing strategy to find the maximum weight independent set?

### 2.2.2 Factorizations and Factor Graphs

The basic observation of the min-sum algorithm is that, even though the original minimization problem may be difficult, if $f$ can be written as a sum of functions depending on only a small subset of the variables, then we may be able to minimize the objective function by performing a series of minimizations over (presumably easier) sub-problems. To make this concrete, let $\mathcal{A} \subseteq 2^{\{1, \ldots, n\}}$.

We say that $f$ factorizes over $\mathcal{A}$ if we can write $f$ as a sum of real valued potential functions $\phi_i : \mathcal{X}_i \to \mathbb{R} \cup \{\infty\}$ and $\psi_\alpha : \mathcal{X}_\alpha \to \mathbb{R} \cup \{\infty\}$ as follows:

$$f(x) = \sum_{i=1}^{n} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \psi_\alpha(x_\alpha) \quad (2.5)$$

This factorization is by no means unique. For example, suppose we are given the objective function $f(x_1, x_2) = x_1 + x_2 + x_1x_2$. There are many different ways that we can factorize $f$:

$$f(x_1, x_2) = x_1 + x_2 + x_1x_2 \quad (2.6)$$

$$= x_1 + (x_2 + x_1x_2) \quad (2.7)$$

$$= (x_1 + x_2 + x_1x_2) \quad (2.8)$$

$$= x_1 + x_2 + \frac{x_1x_2}{2} + \frac{x_1x_2}{2} \quad (2.9)$$

Each of these rewritings represents a different factorization of $f$ (the parenthesis indicate a single
potential function). All of these factorizations can be captured by the above definitions, except for the last. Recall that $\mathcal{A}$ was taken to be a subset of $2^{\{1,...,n\}}$. In order to accommodate the factorization given by equation 2.9, we will allow $\mathcal{A}$ to be a multiset whose elements are members of the set $2^{\{1,...,n\}}$.

The set of all factorizations of the objective function $f(x)$ over the set $\mathcal{A}$ forms a convex set:

$$F_{\mathcal{A}}(f) = \{(\phi, \psi) | \sum_{i=1}^{n} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \psi_\alpha(x_\alpha) = f(x) \text{ for all } x \}$$

(2.10)

If $(\phi, \psi) \in F_{\mathcal{A}}(f)$ and $(\phi', \psi') \in F_{\mathcal{A}}(f)$, then $(\phi, \psi)$ is called a reparameterization of $(\phi', \psi')$ and vice versa.

Every factorization of $f$ has a corresponding graphical representation known as a factor graph. Factor graphs provide a visual representation of the relationship among the potential functions. The factor graph consists of a variable node $i$ for each variable $x_i$, a factor node $\alpha$ for each of the potentials $\psi_\alpha$, and an edge joining the factor node corresponding to $\alpha$ to the variable node representing $x_i$ if $i \in \alpha$. For a concrete example, see Figure 2.3.

\subsection{2.2.3 The Min-Sum Algorithm}

The min-sum algorithm is a local message passing algorithm over a factor graph. During the execution of the min-sum algorithm, messages are passed back and forth between adjacent nodes of the graph. In the algorithm, there are two types of messages: messages passed from variable nodes to factor nodes and messages passed from factor nodes to variable nodes. On the $t^{th}$ iteration of

![Factor graph corresponding to $f(x_1, x_2, x_3) = \phi_1 + \phi_2 + \phi_3 + \psi_{12} + \psi_{23} + \psi_{13}$. By convention, variable nodes are represented as circles and factor nodes are represented as squares.](image)
the algorithm, messages are passed along each edge of the factor graph as follows:

\[ m^t_{i \rightarrow \alpha}(x_i) = \kappa + \phi_i(x_i) + \sum_{\beta \in \partial i \setminus \alpha} m^{t-1}_{\beta \rightarrow i}(x_i) \]  

\[ m^t_{\alpha \rightarrow i}(x_i) = \kappa + \min_{x_\alpha \setminus i} \left[ \psi_\alpha(x_\alpha) + \sum_{k \in \alpha \setminus \{i\}} m^{t-1}_{k \rightarrow \alpha}(x_k) \right] \]  

where \( \partial i \) denotes the set of all \( \alpha \in A \) such that \( i \in \alpha \) (intuitively, this is the set of neighbors of variable node \( x_i \) in the factor graph), \( x_\alpha \) is the vector formed from the entries of \( x \) by selecting only the indices in \( \alpha \), and \( \alpha \setminus i \) is abusive notation for the set-theoretic difference \( \alpha \setminus \{i\} \). When the graph is a tree, these message updates can be derived using the same dynamic programming techniques we saw in Example 2.2.1. When the graph is not a tree, the same updates are used as if the graph was a tree. Understanding when these updates converge to the correct solution for a given graph is the central question underlying the study of the min-sum algorithm.

Each message update has an arbitrary normalization factor \( \kappa \). Because \( \kappa \) is not a function of any of the variables, it only affects the value of the minimum and not where the minimum is located. As such, we are free to choose it however we like for each message and each time step. In practice, these constants are used to avoid numerical issues that may arise during the execution of the algorithm.

**Definition 2.2.1.** A vector of messages \( m = \{ m_{\alpha \rightarrow i}, m_{i \rightarrow \alpha} \} \) is **real-valued** if for all \( \alpha \in A \), \( \forall i \in \alpha \), and \( \forall x_i \in X \), \( m_{\alpha \rightarrow i}(x_i) \) and \( m_{i \rightarrow \alpha}(x_i) \) are real-valued functions (i.e. they do not take the value \( \infty \) for any \( x_i \in X \)).

We will think of the messages as a vector of functions indexed by the edge over which the message is passed. Any vector of real-valued messages is a valid choice for the vector of initial messages \( m^0 \), and the choice of initial messages can greatly affect the behavior of the algorithm. A typical assumption is that the initial messages are chosen such that \( m^0_{\alpha \rightarrow i} \equiv 0 \) and \( m^0_{i \rightarrow \alpha} \equiv 0 \). This uniformity assumption is often useful when we need to analyze the evolution of the algorithm over time, but ideally, we would like to design message passing schemes that perform well independent of initialization.

In Example 2.2.1, we saw that the maximum weight independent set could be computed by looking at all of the messages passed to the root node. \( \text{mwis}(r, \text{inSet}) \) is special in that it recursively depends on all of the problem variables. Because of this, we can compute the value of the maxi-
mum weight independent set simply by trying both possible values of inSet. For general objective functions, this phenomenon is captured by a marginal computation. Recall that a min-marginal of \( f \) is a function of one or more variables obtained by fixing a subset of the variables and minimizing the function \( f \) over all of the remaining variables. For example, the min-marginal for the variable \( x_i \) would be the function \( f_i(x_i) = \min_{x':x_i=x_i} f(x') \).

We can use the messages in order to construct an estimate of the min-marginals of \( f \). Given any vector of messages, \( m^t \), we can construct a set of beliefs that are intended to approximate the min-marginals of \( f \):

\[
\begin{align*}
 b^t_i(x_i) &= \kappa + \phi_i(x_i) + \sum_{\alpha \in \partial i} m^t_{\alpha \rightarrow i}(x_i) \\
 b^t_\alpha(x_\alpha) &= \kappa + \psi_\alpha(x_\alpha) + \sum_{i \in \alpha} m^t_{i \rightarrow \alpha}(x_i)
\end{align*}
\]

If the beliefs corresponded to the true min-marginals of \( f \) (i.e. \( b^t_i(x_i) = \min_{x':x_i=x_i} f(x') \)), then for any \( y_i \in \arg\min_{x_i} b^t_i(x_i) \) there exists a vector \( x^* \) such that \( x^*_i = y_i \) and \( x^* \) minimizes the function \( f \). If the \( |\arg\min_{x_i} b^t_i(x_i)| = 1 \) for all \( i \), then we can take \( x^* = y \), but, if the objective function has more than one optimal solution, then we may not be able to construct such an \( x^* \) so easily. For this reason, theoretical results in this area typically assume that the objective function has a unique global minimum. Although this assumption is common, we will not adopt this convention in this work.

Because our beliefs are not necessarily the true min-marginals, we can only approximate the optimal assignment by computing an estimate of the argmin:

\[
x^*_i \in \arg\min_{x_i} b^t_i(x_i)
\]

**Definition 2.2.2.** A vector, \( b = (\{b_i\}, \{b_\alpha\}) \), of beliefs is **locally decodable** to \( x^* \) if \( b_i(x^*_i) < b_i(x_i) \) for all \( i, x_i \neq x^*_i \). Equivalently, each \( b_i \) has a unique minimum at \( x^*_i \).

If the algorithm converges to a vector of beliefs that are locally decodable to \( x^* \), then we hope that the vector \( x^* \) is a global minimum of the objective function. This is indeed the case when the factor graph contains no cycles. Informally, this follows from the correctness of dynamic programming on a tree. We will defer a formal proof of this result until later in this work (see Corollary 5.2.2). For now, consider using the min-sum algorithm to compute the maximum weight
independent set on a tree:

Example 2.2.2 (Maximum weight independent set on a tree revisited).

Consider the maximum weight independent set problem on the graph $G = (V, E)$ with weights $w_i$ from Example 2.2.1. Let $x \in \{0, 1\}^n$ be an indicator vector for a set $S \subseteq V$ where $x_i = 1$ if $i \in S$ and zero otherwise. We can construct an objective function for the maximum weight independent set problem as follows:

$$f(x) = -\sum_{i \in V} w_i x_i - \sum_{(i,j) \in E} \log\{x_i + x_j \leq 1\}$$

(2.16)

where $\{x_i + x_j \leq 1\}$ is one if $x_i + x_j \leq 1$ and zero otherwise. Fix a vector $\mathbf{x}$ and let $S$ be the corresponding set. We can check that $f(x) = -w(S)$ if $S$ is an independent set in $G$ and $f(x) = \infty$ otherwise. Consequently, any $x$ that minimizes $f$ must correspond to a maximum weight independent set. For the natural factorization given by equation 2.16, the min-sum message passing updates can be written as:

$$m_{i \rightarrow (i,j)}(x_i) = -w_i x_i + \sum_{k \in \partial i \setminus j} m_{i \rightarrow k}^{-1}(x_i)$$

(2.17)

$$m_{(i,j) \rightarrow i}(x_i) = \min_{x_j}[-\log(x_i + x_j \leq 1) + m_{j \rightarrow (i,j)}^{-1}(x_j)]$$

(2.18)

By substituting equation 2.18 into equation 2.17, we have

$$m_{i \rightarrow (i,j)}(x_i) = -w_i x_i + \sum_{k \in \partial i \setminus j} \min_{x_k}[-\log(x_i + x_k \leq 1) + m_{k \rightarrow (i,k)}^{-2}(x_k)]$$

(2.19)

or, equivalently,

$$-m_{i \rightarrow (i,j)}(x_i) = w_i x_i + \sum_{k \in \partial i \setminus j} \max_{x_k}[-\log(x_i + x_k \leq 1) + -m_{k \rightarrow (i,k)}^{-2}(x_k)]$$

(2.20)

which is equivalent to the recurrence, equations 2.2 and 2.3, derived using dynamic programming.

If $G$ is a tree with root $r \in V$, then a simple proof by induction can be used to show that the messages will converge to fixed values after at most $|V|$ time steps. Further, for all $t > |V|$, $-m_{i \rightarrow (i,p(i))}^t(x_i) = \text{mwis}(i, x_i)$ where $p(i) \in V$ is the parent of $i$ in $G$. From this observation, we can conclude that $\max_{x_r} -b_r(x_r)$ corresponds to the value of the maximum weight independent set.
Notice that, unlike the dynamic programming solution, there is no fixed root node, and the min-sum algorithm does not pass messages only in one direction in the tree. The min-sum algorithm actually computes all of the min-marginals simultaneously, instead of just the one at the root.

**Computation Trees**

An important tool in the analysis of the min-sum algorithm is the notion of a computation tree. Intuitively, the computation tree is an unrolled version of the original graph that captures the evolution of the messages passed by the min-sum algorithm needed to compute the belief at time $t$ at a particular node of the factor graph. Computation trees describe the evolution of the beliefs over time, which, in some cases, can help us prove correctness and/or convergence of the message passing updates. For example, the convergence of the min-sum algorithm on graphs containing a single cycle can be demonstrated by analyzing the computation trees produced by the min-sum algorithm at each time step [54].

The depth $t$ computation tree rooted at node $i$ contains all of the length $t$ non-backtracking walks in the factor graph starting at node $i$. A walk is non-backtracking if it does not go back and forth successively between two vertices. For any node $v$ in the factor graph, the computation tree at time $t$ rooted at $v$, denoted by $T_v(t)$, is defined recursively as follows: $T_v(0)$ is just the node $v$, the root of the tree. The tree $T_v(t)$ at time $t > 0$ is generated from $T_v(t - 1)$ by adding to each leaf of $T_v(t - 1)$ a copy of each of its neighbors in $G$ (and the corresponding edge), except for the neighbor that is already present in $T_v(t - 1)$. Each node of $T_v(t)$ is a copy of a node in $G$, and the potentials on the nodes in $T_v(t)$, which operate on a subset of the variables in $T_v(t)$, are copies of the potentials of the corresponding nodes in $G$. The construction of a computation tree for the graph in Figure 2.3 is pictured in Figure 2.4. Note that each variable node in $T_v(t)$ represents a distinct copy of some variable $x_j$ in the original graph.

Given any initialization of the messages, $T_v(t)$ captures the information available to node $v$ at time $t$. At time $t = 0$, node $v$ has received only the initial messages from its neighbors, so $T_v(0)$ consists only of $v$. At time $t = 1$, $v$ receives the round one messages from all of its neighbors, so $v$’s neighbors are added to the tree. These round one messages depend only on the initial messages, so the tree terminates at this point. By construction, we have the following lemma:

**Lemma 2.2.1.** The belief at node $v$ produced by the min-sum algorithm at time $t$ corresponds to the
**Fixed Point Properties**

Computation trees provide us with a dynamic view of the min-sum algorithm. After a finite number of time steps, we hope that the beliefs on the computation trees stop changing and that the message vector converges to a fixed point of the message update equations (in practice, when the beliefs change by less than some small amount, we say that the algorithm has converged). For any real-valued objective function \( f \) (i.e. \(|f(x)| < \infty \) for all \( x \)), there always exists a fixed point of the message update equations (see Theorem 2 of [51]).

Ideally, the beliefs constructed from any fixed point of the message update equations would be the true min-marginals of the function \( f \). If the beliefs are the exact min-marginals, then the estimate corresponding to our beliefs would indeed be a global minimum. Unfortunately, the algorithm is only known to produce the exact min-marginals on special factor graphs (e.g. when the factor graph is a tree, see Section 4.1). Instead, we show that the fixed-point beliefs are similar to min-marginals. Like the messages, we will think of the beliefs as a vector of functions indexed by the nodes of the factor graph. Consider the following definitions:

**Definition 2.2.3.** A vector of beliefs, \( b \), is **admissible** for a function \( f \) if

\[
f(x) = \kappa + \sum_i b_i(x_i) + \sum_{\alpha} \left( b_\alpha(x_\alpha) - \sum_{k \in \alpha} b_k(x_k) \right)
\]

Beliefs satisfying this property are said to reparameterize the objective function.
Definition 2.2.4. A vector of beliefs, \( b \), is **min-consistent** if for all \( \alpha \) and all \( i \in \alpha \):

\[
\min_{x_{\alpha \setminus i}} b_{\alpha}(x_{\alpha}) = \kappa + b_i(x_i)
\]

Any vector of beliefs that satisfies these two properties produces a reparameterization of the original objective function in terms of the beliefs. Additionally, any vector of beliefs obtained from a fixed point of the message updates satisfies these two properties:

**Theorem 2.2.2.** For any vector of fixed-point messages, the corresponding beliefs are admissible and min-consistent.

**Proof.** Let \( m \) be a fixed point of the message update equations:

\[
m_{i \to \alpha}(x_i) = \kappa + \phi_i(x_i) + \sum_{\beta \in \partial \setminus \alpha} m_{\beta \to i}(x_i) \tag{2.21}
m_{\alpha \to i}(x_i) = \kappa + \min_{x_{\alpha \setminus i}} \left[ \psi_{\alpha}(x_{\alpha}) + \sum_{k \in \alpha \setminus i} m_{k \to \alpha}(x_k) \right] \tag{2.22}
\]

First, we show that \( m \) produces min-consistent beliefs. Take \( \alpha \in \mathcal{A} \) and choose some \( i \in \alpha \). Without loss of generality, we can assume that the constants are uniformly equal to zero.

\[
\min_{x_{\alpha \setminus i}} b_{\alpha}(x_{\alpha}) = \min_{x_{\alpha \setminus i}} \psi_{\alpha}(x_{\alpha}) + \sum_{k \in \alpha} m_{k \to \alpha}(x_k) \tag{2.23}
= m_{i \to \alpha}(x_i) + \min_{x_{\alpha \setminus i}} \psi_{\alpha}(x_{\alpha}) + \sum_{k \in \alpha \setminus i} m_{k \to \alpha}(x_k) \tag{2.24}
= m_{i \to \alpha}(x_i) + m_{\alpha \to i}(x_i) \tag{2.25}
= \phi_i(x_i) + \left[ \sum_{\beta \in \partial \setminus \alpha} m_{\beta \to i}(x_i) \right] + m_{\alpha \to i}(x_i) \tag{2.26}
= b_i(x_i) \tag{2.27}
\]

Next, we can check that the beliefs are admissible. Again, we can assume that the constants
are uniformly equal to zero.

\begin{align*}
f(x) &= \sum_i \phi_i(x_i) + \sum_\alpha \psi_\alpha(x_\alpha) \\
&= \sum_i \left[ \phi_i(x_i) + \sum_{\alpha \in \partial i} m_{\alpha \rightarrow i}(x) \right] + \sum_\alpha \left[ \psi_\alpha(x_\alpha) - \sum_{i \in \alpha} m_{\alpha \rightarrow i}(x_i) \right] \\
&= \sum_i b_i(x) + \sum_\alpha \left[ \psi_\alpha(x_\alpha) - \sum_{i \in \alpha} m_{\alpha \rightarrow i}(x_i) \right] \\
&= \sum_i b_i(x) + \sum_\alpha \left[ \psi_\alpha(x_\alpha) + \sum_{i \in \alpha} m_{i \rightarrow \alpha}(x_i) - \sum_{i \in \alpha} m_{i \rightarrow \alpha}(x_i) - \sum_{i \in \alpha} m_{\alpha \rightarrow i}(x_i) \right] \\
&= \sum_i b_i(x) + \sum_\alpha \left[ \psi_\alpha(x_\alpha) - \sum_{i \in \alpha} \left[ m_{i \rightarrow \alpha}(x_i) + m_{\alpha \rightarrow i}(x_i) \right] \right] \\
&= \sum_i b_i(x) + \sum_\alpha \left[ \psi_\alpha(x_\alpha) - \sum_{i \in \alpha} m_{i \rightarrow \alpha}(x_i) - \sum_{i \in \alpha} \left[ m_{i \rightarrow \alpha}(x_i) + m_{\alpha \rightarrow i}(x_i) \right] \right] \\
&= \sum_i b_i(x) + \sum_\alpha \left[ \psi_\alpha(x_\alpha) - \sum_{i \in \alpha} b_i(x_i) \right]
\end{align*}

Theorem is not new, and other proofs can be found, for example, in [51]. We present the proof here for completeness, and we will make use of similar ideas in subsequent chapters.
Chapter 3

The Min-Sum Algorithm:
Convergence and Correctness

In this chapter, we explore the convergence and correctness properties of the min-sum algorithm by examining its behavior in a variety of application areas. Sections 3.1 and 3.2 demonstrate that the min-sum algorithm can be successfully employed to solve simple problems on graphs that are known to be solvable in polynomial time. Sections 3.3 and 3.4 discuss example problems on which the min-sum algorithm performs suboptimally (e.g. the algorithm does not converge, converges to the wrong answer, etc.). The discussion in this chapter is designed to highlight the strengths and weaknesses of the min-sum algorithm as well as to demonstrate the computation tree approach for proving the convergence and correctness of the min-sum algorithm.

As we saw in Section 2.2.3, convergence of the min-sum algorithm is typically taken to mean that the message vector at time $t$ is converging to a fixed point of the message update equations. This is a strong requirement, and it is not necessary to ensure the usefulness of the algorithm. In this chapter, we observe that a weaker notion of convergence can be used to obtain practical algorithms. We will say that the min-sum algorithm has converged at time $t'$ if for all $t \geq t'$ the estimates, produced by the singleton beliefs, remain fixed. Convergence of the standard min-sum algorithm, in this sense, can be demonstrated by a careful analysis of the computation trees. Other notions of convergence are discussed in Chapter 6.
3.1 Shortest Paths

Let \( G = (V, E) \) be a directed graph. For each edge \( e \in E \) there is an associated non-negative rational weight, \( w_e \). The weight of any path \( v_1, \ldots, v_k \) is equal to the sum of the weights along each edge of the path, \( w_{v_1,v_2} + \ldots + w_{v_{k-1},v_k} \). A typical shortest path problem involves computing the path of minimum weight between two nodes in a graph. Such problems are solvable in polynomial time via Dijkstra’s algorithm, the Bellman-Ford algorithm, and others.

3.1.1 Single-Pair Shortest Path

Given a pair of vertices \( s, t \in V \), the single-pair shortest path problem is to find a minimum weight path from \( s \) to \( t \). If there is path between \( s \) and \( t \) in \( G \), then the shortest \( s-t \) path problem can be formulated as an integer program:

\[
\begin{align*}
\text{minimize:} & \quad \sum_{e \in E} w_e x_e \\
\text{subject to:} & \quad \sum_{j \text{ st. } (j,i) \in E} x_{(j,i)} = \sum_{j \text{ st. } (i,j) \in E} x_{(i,j)} \quad \text{for } i \in V - \{s, t\} \\
& \quad \sum_{(s,i) \in E} x_{(s,i)} = 1 + \sum_{(i,s) \in E} x_{(i,s)} \\
& \quad \sum_{(j,t) \in E} x_{(j,t)} = 1 + \sum_{(t,j) \in E} x_{(t,j)} \\
& \quad x_e \in \{0, 1\} \text{ for each } e \in E
\end{align*}
\]

Here, \( x_e \) is a zero-one variable that indicates whether the edge \( e \) should be taken on the path from \( s \) to \( t \). A valid \( s-t \) path must have at least one more edge out of \( s \) than into \( s \), at least one more edge into \( t \) than out of \( t \), and for each intermediate node on the \( s-t \) path the number of edges on the \( s-t \) path that enter this node must be exactly the same as the number of edges that leave this node.

This integer program can be converted into a linear program by relaxing the integrality constraint from \( x_e \in \{0, 1\} \) to \( x_e \geq 0 \). The matrix describing this linear program corresponds to the \( V \times E \) incidence matrix for \( G \). Such a matrix is totally unimodular [44]. For our purposes, this
means that if there is a unique shortest path from \( s \) to \( t \) in \( G \), then the linear program has a 
unique optimal solution which corresponds to a solution of the integer program. Consequently, sol-
lutions to the shortest \( s-t \) path problem can be found efficiently using standard linear programming 
techniques.

We can convert the integer program into an objective function to be minimized via the min-sum 
algorithm by constructing an indicator function \( \psi_i(x_{I(i)}) \) for each of the constraints of the linear 
program where \( I(i) \) is the set of edges incident to node \( i \):

\[
f(x) = \sum_{e \in E} w_e x_e + \sum_{i \in V} -\log \psi_i(x_{I(i)}). \tag{3.1}
\]

For any vector \( \mathbf{x} \in \{0, 1\}^{|E|} \), \( f(\mathbf{x}) \) is equal to \( \sum_{e \in E} w_e \mathbf{x}_e \) if the nonzero elements of \( \mathbf{x}_e \) define a 
directed walk from \( s \) to \( t \) in \( G \) and equals infinity otherwise. Therefore, minima of \( f \) correspond to 
minimal weight \( s-t \) paths. If there is no path from \( s \) to \( t \), then \( f \) is infinity for all choices of \( x \).

We can attempt to use the min-sum algorithm in order to minimize this objective function. For 
simplicity, we will assume that the initial messages are identically equal to zero. If there is a unique 
shortest path from \( s \) to \( t \) in \( G \), then by examining the computation trees produced by the min-sum 
algorithm, we can show that the estimates produced at time \( t \) by the min-sum algorithm converge 
to the correct solution. Recall that computation trees model the message passing structure of the 
min-sum algorithm: minimal solutions on \( T_s(t) \) correspond to minima of \( b_{x}^t(x_e) \) where \( b^t \) is the 
vector of beliefs obtained by running the min-sum algorithm for \( t \) time steps (see Section 2.2.3).

In this case, the minimization problem on the computation tree is different from the original \( s-t \) 
path problem (see Figure 3.1). There are now multiple copies of \( s \) and \( t \) which makes the problem 
on the computation tree a multi-source/multi-sink shortest path problem. Specifically, a feasible 
solution on the computation tree is a set of edges such that every copy of \( s \) in the tree has a path 
to a copy of \( t \) or a boundary node (allowed by the message initialization), every copy of \( t \) in the 
tree has a path from a copy of \( s \) or some boundary node, and every vertex has at least as many out 
edges as in edges. A feasible solution is minimal if no other feasible solution has a smaller weight.

If \( G \) has a unique minimum \( s-t \) path, then the min-sum algorithm always converges to the correct 
solution in a number of steps that depends on the weight of the shortest path and the weight of the 
second shortest path. Define \( w(S) = \sum_{e \in S} w_e \) for \( S \) a set of edges. Let \( w_{\min} = \min_{e \in E} w_e \), and let 
\( \epsilon \) be the difference in weight between the second best \( s-t \) path in \( G \) and the optimal \( s-t \) path in \( G \).
We have the following theorem:

**Theorem 3.1.1.** If $G$ has a unique minimum path $s-t$, $P^*$, then an edge $e \in E$ is in $P^*$ if and only if every minimal solution on $Te(2n)$ contains the root for $n > \frac{w(P^*)^2}{w_{\min}} + \frac{w(P^*)}{w_{\min}}$. In other words, for all $n > \frac{w(P^*)^2}{w_{\min}} + \frac{w(P^*)}{w_{\min}}$, $1 \in \arg \min \{b_n(x_e) \mid e \in M\}$ if and only if $e \in P^*$.

**Proof.** ($\Rightarrow$) Suppose by way of contradiction that $e = (u,v)$ is in the min $s$-$t$ path $P^*$ on $G$ but that there is some minimal solution $M$ on the computation tree rooted at $e$ at time $2n$ that does not contain the root.

This proof builds an alternating set of paths in $Te(2n)$ that can be swapped to improve the optimality of the solution. We will construct two sets of edges, $M_{\text{sub-opt}}$ and $M_{\text{opt}}$, as follows:

1. Starting at $v$, follow the edges in $P^*$ forward until reaching a copy of $t$ or until doing so would require traversing an edge in $M$. Similarly, starting at $u$ follow the edges in $P^*$ backwards until reaching a copy of $s$ or until doing so would require traversing an edge in $M$. Add this copy of a sub-path of $P^*$ to $M_{\text{opt}}$.

2. By construction, for each sub-path $P$ in $M_{\text{opt}}$ not originating at a leaf, there must be at least one path in $M$ that either (possibly both for the edge added in step 1)

   (a) leaves the head of $P$ and terminates in a copy of $t$ or in a leaf of $Te(2n)$. If no such path is in $M_{\text{sub-opt}}$, choose such a path $P'$ in $M$ and follow it until $t$, the boundary, or until doing so would require traversing a copy of an edge in $P^*$. Add this sub-path to $M_{\text{sub-opt}}$.

   (b) enters the tail of $P$ and originates in a copy of $s$ or in a leaf of $Te(2n)$. If no such path is in $M_{\text{sub-opt}}$, choose such a path $P'$ in $M$ and follow it backwards until $s$, the boundary,
or until doing so would require traversing a copy of an edge in $P^\ast$. Add this sub-path to $M_{\text{sub-opt}}$.

3. By construction, for each sub-path $P$ in $M_{\text{sub-opt}}$ not touching the boundary, there must be a copy of $P^\ast$ that either

(a) leaves the head of $P$ and terminates in a copy of $t$ or in a leaf of $T_c(2n)$. If this path is not in $M_{\text{opt}}$, follow it until $t$, the boundary, or or until doing so would require traversing a copy of an edge in $M$. Add this sub-path to $M_{\text{opt}}$.

(b) enters the tail of $P$ and originates in a copy of $s$ or in a leaf of $T_c(2n)$. If this path is not in $M_{\text{opt}}$, follow it backwards until $s$, the boundary, or until doing so would require traversing a copy of an edge in $M$. Add this sub-path to $M_{\text{opt}}$.

4. Repeat steps 2 through 3 until no additional paths satisfying the above criteria can be found.

This process builds a set of paths starting at the root that alternates between subpaths of copies of $P^\ast$ and subpaths of the solution $M$. Figure 3.2 illustrates the construction for a graph $G$ whose unique minimum s-t path is $(s, a), (a, b), (b, c), (c, d), (d, e), (e, f), (f, t)$.

Let $M^\ast = (M \setminus M_{\text{sub-opt}}) \cup M_{\text{opt}}$. Notice that $M^\ast$ is a feasible solution on $T_c(2n)$ since the in degree and out degree of every node in $M$ is preserved, and by the construction, sub-paths in $M_{\text{opt}}$ satisfy the constraints at their heads and tails when added to $M$.

Let $k$ be the number of disjoint sub-paths in $M_{\text{opt}}$ and $k'$ be the number of disjoint sub-paths in $M_{\text{sub-opt}}$. In the worst case, $k = k' + 1$ due to the alternating construction. As all other possible outcomes can be reduced to this situation, we only illustrate the proof in this instance. There are two cases:

Case 1: $k > \frac{w(P^\ast)}{\epsilon} + 1$
In this case there are enough alternations in order to ensure that the added edges do not exceed the removed edges in total weight. We have \((k - 1)\epsilon > w(P^*)\), so

\[
\begin{align*}
w(M^*) & = w(M) - w(M_{sub-opt}) + w(M_{opt}) \\
& < w(M) - (k - 1)\epsilon + w(P^*) \\
& < w(M)
\end{align*}
\]

Case 2: \(k \leq \frac{w(P^*)}{\epsilon} + 1\)

There are no longer a sufficient number of alternations in order to guarantee that the swap actually improved the solution. Instead, we will rely on the observation that \(M_{opt} \cup M_{sub-opt}\) contains precisely \(2n + 1\) edges. Because the number of alternations is relatively small, only so many of these edges can correspond to edges in \(M_{opt}\). Therefore, for large enough \(n\), the new solution \(M^*\) will be an improvement over \(M\).

\[
\begin{align*}
w(M^*) & = w(M) - w(M_{sub-opt}) + w(M_{opt}) \\
& \leq w(M) - (2n - k|P^*|)w_{min} + w(M_{opt}) \\
& \leq w(M) - (2n - \frac{kw(P^*)}{w_{min}})w_{min} + kw(P^*) \\
& \leq w(M) - 2nw_{min} + 2kw(P^*) \\
& < w(M)
\end{align*}
\]

Here, \(|P^*|\) is the number of edges in the path \(P^*\). In either case, \(w(M^*) < w(M)\) contradicting the minimality of \(M\).

\((\Leftarrow)\) For the opposite direction, suppose by way of contradiction that every minimal solution on \(T_{c}(2n + 1)\) contains the root edge \(e\) but that \(e\) is not in \(P^*\). We can construct \(M_{sub-opt}\) and \(M_{opt}\) in an alternating fashion as above in order to reach a contradiction. As the details of the proof are nearly identical, we omit them here.

Theorem 3.1.1 provides only a loose upper bound on the number of iterations required to compute the shortest path, and tighter analyses may be possible. Despite this, for a graph \(G = (V, E)\) with a unique minimum \(s\)-\(t\) path, we can upper bound the running time of the min-sum algorithm as \(O\left(\frac{w^2}{w_{min}}|E||V|\right)^2\).
3.1.2 Single-Source Shortest Paths

Instead of finding the shortest path between two vertices of a directed graph, consider finding the shortest path from one vertex in the graph to all other vertices in the graph. This problem is known as the single-source shortest path problem: given a directed graph \( G = (V, E) \), a non-negative weight \( w_e \) for each \( e \in E \), and a vertex \( s \in V \), compute the shortest path from \( s \) to all other nodes in \( V \). The following is a linear program for the single-source shortest path problem:

\[
\begin{align*}
\text{maximize} & \quad \sum_{i \in V \setminus s} x_i \\
\text{subject to} & \quad x_j \leq w_{ij} + x_i \quad \forall (i, j) \in E \\
& \quad x_s = 0, x_i \geq 0 \quad \forall i \in V \setminus s
\end{align*}
\]

Solving this linear program is equivalent to minimizing the following objective function:

\[
f(x) = -\log\{x_s = 0\} - \sum_{i \in V \setminus s} x_i - \sum_{(i, j) \in E} \log\{x_j \leq w_{ij} + x_i\} \quad (3.2)
\]

Intuitively, if \( x^* \in \arg \min_x f(x) \), then \( x^*_i \) must correspond to the length of the shortest path from \( s \) to \( i \). As \( x_i \) can be any nonnegative real, the minimization is performed over a continuous state space. The message from \( \{i, j\} \) to \( i \) is then a function from \( \mathbb{R} \) to \( \mathbb{R} \). As storing the message vector could require, in the worst case, recording the value of \( m_{(i,j)\to i}(x_i) \) for each \( (i, j) \in E \) and each \( x_i \in \mathbb{R} \), the min-sum algorithm may not be a practical message passing scheme for this problem.

Ignoring these complexities for the moment, we can show that the standard min-sum algorithm will converge to the correct solution for this problem:

**Theorem 3.1.2.** Let \( x^* \) be the optimal solution to the linear program for the graph \( G = (V, E) \). For all \( t > 2|V| \) and each \( i \in V \), \( \arg \min_x b^t_i(x) = \{x^*_i\} \) where \( b^t \) is the vector of beliefs produced by the min-sum algorithm at time \( t \).

**Proof.** A copy of the shortest path from \( s \) to \( i \) must be contained in \( T_i(2|V|) \). Let \( v_1, \ldots, v_k \) denote the vertices of the path such that \( v_1 \in T_i(2|V|) \) is a copy of \( s \) and \( v_k \) is the root of the computation.
tree. The constraints enforce that

\[ x_v \leq w_{vk-1}v_k + x_{k-1} \]  
\[ \leq w_{vk-1}v_k + w_{vk-2}v_{k-1} + x_{k-2} \]  
\[ \leq \sum_{j=1}^{k} w_{v_{j}v_{j+1}} \]  
\[ = x^*_i \]  

As this is the shortest path from \( s \) to \( i \), this is the maximum value that \( x_v \) can achieve.

The vector of messages takes a special form when the initial messages are chosen to be uniformly equal to zero. For any time \( t \), a simple proof by induction shows that each message can be parameterized, up to a constant, as a monotone decreasing function plus a constraint that places an upper bound on the variable.

For example, \( m_{(i,j)\rightarrow i}(x_i) = k^t_{i,j\rightarrow i}(x_i) - \log\{ x_i \leq d^t_{i,j\rightarrow i} \} \) for a monotone decreasing function \( k^t_{i,j\rightarrow i} \) and a nonnegative constant \( d^t_{i,j\rightarrow i} \in \mathbb{R} \cup \{ \infty \} \). We will demonstrate the inductive step. Suppose there exists an edge from \( i \neq s \) to \( j \) in \( G \).

\[ m^t_{(i,j)\rightarrow i}(x_i) = \min_{x_j} -\log\{ x_j \leq w_{ij} + x_i \} + m^{t-1}_{j\rightarrow(i,j)}(x_j) \]  
\[ = \min_{x_j} -\log\{ x_j \leq w_{ij} + x_i \} + k^{t-1}_{j\rightarrow(i,j)}(x_j) - \log\{ x_j \leq d^{t-1}_{j\rightarrow(i,j)} \} \]  
\[ = k^{t-1}_{j\rightarrow(i,j)}(w_{ij} + x_i)\{ w_{ij} + x_i \leq d^{t-1}_{j\rightarrow(i,j)} \} \]  
\[ + k^{t-1}_{j\rightarrow(i,j)}(d^{t-1}_{j\rightarrow(i,j)})\{ w_{ij} + x_i > d^{t-1}_{j\rightarrow(i,j)} \} \]  

\[ m^t_{(i,j)\rightarrow j}(x_j) = \min_{x_i} -\log\{ x_j \leq w_{ij} + x_i \} + m^{t-1}_{i\rightarrow(j,i)}(x_i) \]  
\[ = \min_{x_i} -\log\{ x_j \leq w_{ij} + x_i \} + k^{t-1}_{i\rightarrow(j,i)}(x_i) - \log\{ x_i \leq d^{t-1}_{i\rightarrow(j,i)} \} \]  
\[ = -\log\{ x_j \leq w_{ij} + d^{t}_{i\rightarrow(j,i)} \} + k^{t-1}_{i\rightarrow(j,i)}(d^{t-1}_{i\rightarrow(j,i)}) \]  

Observe that, with our choice of initialization, \( m^t_{(i,j)\rightarrow i}(x_i) \) is a monotone decreasing function for all \( t \geq 0 \). As such, these messages do not effect the location of the minimum of \( b^t_i(x_i) \), only its value. Consequently, the algorithm only needs to pass factor to variable messages of the form \( m^t_{(i,j)\rightarrow j}(x_j) \) which in turn only require passing variable to factor messages of the form \( m^t_{i\rightarrow(j,i)}(x_i) \).
Algorithm 1 Bellman-Ford Algorithm

1: Set $x_s = 0$.
2: Set $x_i = \infty$ for all $i \in V \setminus s$.
3: for $k = 1$ to $|V| - 1$ do
4:   for each $(i, j) \in E$ do
5:     if $x_i + w_{ij} < x_j$ then
6:       $x_j = x_i + w_{ij}$
7:     end if
8:   end for
9: end for

That is, we can restrict the algorithm to only pass messages across directed edges.

A similar observation allows us to conclude that we can, in fact, fix each of the monotone decreasing functions at time $t$ to be the all zero function without changing the estimate produced by the min-sum algorithm at time $t$. As a result, every message can be parameterized by a single nonnegative number corresponding to the upper bound on the appropriate variable.

With these simplifications and Theorem 3.1.2, we have that after at most $|V|$ rounds of message passing, each of which involves passing $|E|$ messages, the estimate produced by the min-sum algorithm for the variable $x_i$ corresponds to the shortest path from $s$ to $i$ in $G$. In other words, the min-sum algorithm produces an $O(|E||V|)$ algorithm for the single-source shortest path problem.

Many other algorithms have been proposed to solve the single-source shortest path problem. A simplified version of one such algorithm, the Bellman-Ford algorithm, is described in Algorithm 1. This algorithm can be converted into a distributed algorithm on the graph $G$ by having each node $i$ at time $t$ pass a message to each one of its neighbors, $j$, that is equal to the shortest distance from $s$ to $i$ that was observed at any previous time step plus the weight of the edge from $i$ to $j$. Each node can then update its own internal estimate of the shortest distance from $s$. We can check that the distributed Bellman-Ford algorithm is then equivalent to the min-sum algorithm for the single-source shortest path problem.

3.2 Transitive Closure

Let $G = (V, E)$ be a unweighted, directed graph. The transitive closure of the graph $G$ is a graph $\overline{G} = (V, \overline{E})$ such that $(i, j) \in \overline{E}$ if and only if there is a path from $i$ to $j$ in $G$ (see Figure 3.3). The transitive closure of a graph always exists and is always unique.

For each $i \neq j \in V$, let $x_{ij} \in \{0, 1\}$ be a variable that indicates whether there is a path from $i$ to
\[ f(x) = \sum_{i,j} x_{ij} + \sum_i \sum_{j \neq i} \sum_{k \notin \{i,j\}} \log \left( x_{ij} \geq x_{ik}x_{kj} \right) + \sum_{(i,j) \in E} -\log \left( x_{ij} = 1 \right). \]

Let \( x^* = \arg\min_x f(x) \). We have that \( x^*_{ij} = 1 \) if and only if \((i,j) \in \overline{E}\).

Again, the min-sum algorithm converges for this problem:

**Theorem 3.2.1.** Let \( x^* \) be the unique minimizer of the objective function. For all \( t > 2 \log |V| \) and each \( i \in V \), \( \arg\min_x b^*_t(x) = \{x^*_i\} \) where \( b^t \) is the vector of beliefs produced by the min-sum algorithm at time \( t \).

**Proof.** Consider the computation tree \( T_{ij}(2 \log |V|) \). If there is not a path from \( i \) to \( j \) in \( |V| \), then \( b^t_{ij}(1) > b^t_{ij}(0) \) for all \( t \geq 0 \). If there is a path from \( i \) to \( j \) in \( t \), then either \((i,j) \in E\) or there must be a path from \( i \) to \( k \) and from \( k \) to \( j \) where \( k \) is the mid-point of the shortest path from \( i \) to \( j \).

In the latter case, we can again split each sub-path in half. Every time we split a path, we require messages from one step earlier in time. As the shortest path from \( i \) to \( j \) in \( V \) is at most length \( |V| \), the value at the root will be correct as long the computation tree has depth \( 2 \log |V| \).

As there are \( O(|V|^2) \) nodes and edges in the factor graph, this naive approach yields an \( O(|V|^3 \log |V|) \) algorithm for the transitive closure.

### 3.3 Maximum Weight Matchings

Let \( G = (V,E) \) be an undirected graph with edge weights \( w_e \) for each edge \( e \in E \). A matching \( M \subseteq E \) in a graph is a subset of the edges such that no two edges in \( M \) have a common endpoint. The weight of a matching, denoted \( w(M) \), is the sum of the weights of the edges contained in \( M \). The maximum weight matching problem can be solved in polynomial time by using Edmond’s blossom algorithm.
Figure 3.4: An example of the maximum weight matching problem. The dashed edges belong to the maximum weight matching.

The maximum weight matching problem (MWM) is described by the following integer program:

\[
\begin{align*}
\text{maximize} & \quad w^T x \\
\text{subject to} & \quad \sum_{k \in \partial_i} x_{ki} \leq 1 \quad \forall i \in V \\
& \quad x_{ij} \in \{0, 1\} \quad \forall (i, j) \in E
\end{align*}
\]

This integer program can be relaxed into a linear program by replacing the integrality constraints with the constraint that \( \forall (i, j) \in E, x_{ij} \geq 0 \). We can formulate an objective function for the above integer program in the standard way:

\[
f(x) = \sum_{e \in E} -w_e x_e + \sum_i \log \left\{ \sum_{k \in \partial_i} x_{ki} \leq 1 \right\}
\]

(3.13)

In the context of min-sum style message passing algorithms, the maximum weight matching problem has been studied by several authors [41, 7, 6]. For the above objective function, we can summarize their results as:

**Theorem 3.3.1.** Let \( G = (V, E) \) be a graph with nonnegative real weights on the edges. If the linear programming relaxation has no fractional optima and an integral optimum \( x^* \), then the estimate produced by the min-sum algorithm is correct for all \( t \geq \frac{2w_{\text{max}}}{c} \) where \( c = \min_{x \in \mathcal{P}} \frac{w^T(x^*-x)}{\|x^*-x\|_1} \) and \( \mathcal{P} \) is the polytope corresponding to the feasible region of the above LP relaxation.

**Proof.** This theorem was proved using a computation tree analysis similar to those above. See Theorem 1 of [41].

Notice that, unlike the shortest path and transitive closure algorithms described above, correctness and convergence of the algorithm requires not only uniqueness of the solution to the integer program, but also that the linear programming relaxation has a unique, integral optimum. For bipartite graphs, every optimal solution of the integer program is also an optimal solution to the
linear relaxation. Consequently, the min-sum algorithm can be used to effectively solve the weighted bipartite matching problem.

### 3.4 Weaknesses of the Min-Sum Algorithm

Convergence and correctness of the min-sum algorithm for arbitrary factor graphs has only been demonstrated for a few special cases such as trees and single cycles [51, 55, 2]. In this section, we describe the possible failure modes of the min-sum algorithm.

**The algorithm may not converge**

Although fixed points of the message passing updates are guaranteed to exist, there is no guarantee that the algorithm actually converges to one. For example, the min-sum algorithm does not converge to a fixed point solution for the maximum weight independent set problem in Figure 3.5a when the message vector is initialized to the zero vector. Instead, the algorithm alternates between including and excluding each node from the maximum weight independent set. This effect occurs because the leaves of each computation tree comprise approximately fifty percent of the nodes of every computation tree, and for this choice of weights, the maximum weight independent set on each computation tree always includes the boundary variable nodes.

Even though the min-sum algorithm is not guaranteed to converge, several heuristics seem to be useful for forcing the convergence of the algorithm in practice. One example of such a heuristic is the damped message update. Given a damping factor $\delta \in [0, 1]$, the damped message updates take the following form:

$$m_{\alpha \rightarrow i}^t(x_i) = \kappa + (1 - \delta)m_{\alpha \rightarrow i}^{t-1}(x_i) + \delta \min_{x_{\alpha \setminus i}} \left[ \psi_\alpha(x_\alpha) + \sum_{k \in \alpha \setminus i} m_{k \rightarrow \alpha}^{t-1}(x_k) \right]$$  \tag{3.14}
The damped message updates are a convex combination of the min-sum message update and the previous message vector. Notice that the fixed points of Equation 3.14 are also fixed points of Equation 2.12 and vice versa.

The converged estimates are not unique

Even if the algorithm does converge to a vector of fixed points messages, there is no guarantee that we can construct an assignment from the converged beliefs in a meaningful way. This is particularly problematic whenever the beliefs are not locally decodable. Consider the maximum weight independent set problem in Figure 3.5b. When the initial vector of messages is chosen to be the all zero message, the converged beliefs are not locally decodable for this problem (i.e. both zero and one minimize each of the singleton beliefs). By fixing each variable in turn, we can still construct an assignment that satisfies all of the problem constraints. For more complicated graphs, constructing an assignment from the converged beliefs may be just as hard as solving the minimization problem.

The algorithm converges to a suboptimal assignment

In the case that the beliefs are locally decodable, we can construct an estimate from the converged beliefs. However, the constructed estimate does not necessarily correspond to the global optimum of the objective function [56]. Instead, as we will see in Chapter 5, these estimates need only correspond to local optima.

Despite these weaknesses, the min-sum algorithm has found empirical success in a variety of application areas including statistical physics, combinatorial optimization [42, 41, 6, 8], computer vision, clustering [13], turbo codes [1, 9], and the minimization of quadratic functions [31, 29]; however, rigorously characterizing the behavior of the algorithm outside of a few well-structured instances has proved challenging.

Our goal in this work is to understand why each of these failure modes occur and to design local message passing algorithms with stronger guarantees on convergence and correctness.
Chapter 4

Reparameterizations

Recall from Theorem 2.2.2 that fixed points of the min-sum algorithm are admissible: every vector of fixed-point beliefs, \( b^* \), for the min-sum algorithm produces a reparameterization of the objective function,

\[
    f(x) = \sum_i \phi_i(x_i) + \sum_{\alpha} \psi_{\alpha}(x_{\alpha}) = \sum_i b^*_i(x_i) + \sum_{\alpha} [b^*_\alpha(x_{\alpha}) - \sum_{k \in \alpha} b^*_k(x_k)].
\]

In other words, we can view the min-sum algorithm as trying to produce a reparameterization of the objective function over \( \mathcal{A} \) in terms of min-consistent beliefs. If the factor graph is a tree, then as was observed by Pearl and others, the min-marginals of the objective function produce such a factorization:

**Lemma 4.0.1.** For each \( i \), let \( f_i(x_i) \) be the min-marginal for variable the \( x_i \), and for each \( \alpha \in \mathcal{A} \), let \( f_{\alpha} \) be the min-marginal for the vector of variables \( x_{\alpha} \). If the factor graph corresponding to \( f \) is a tree, then \( f \) can be reparameterized in terms of its min-marginals as follows:

\[
    f(x) = \kappa + \sum_i f_i(x_i) + \sum_{\alpha} \left( f_{\alpha}(x_{\alpha}) - \sum_{k \in \alpha} f_k(x_k) \right)
\]

**Proof.** For example, see Theorem 1 of [51]

When the factor graph is not a tree, the min-marginals of the objective function do not necessarily produce a factorization of the objective function in this way, but we can still hope that we can construct a minimizing assignment from admissible and min-consistent beliefs.
In this chapter, we explore reparameterizations in an attempt to understand what makes one factorization of the objective function better than another. Reparameterizations, and lower bounds derived from them, will be an essential ingredient in the design of convergent and correct message passing algorithms. In Section 4.1, we use message reparameterizations to show that a slight modification to the definition of the beliefs for the min-sum algorithm can be used to ensure that beliefs corresponding to any vector of real-valued messages is admissible. In Section 4.2, we show that a similar technique can be used to produce an alternative reparameterization of the objective function that is characterized by a vector of non-zero reals. This reparameterization will form the basis for a new message passing algorithm called the splitting algorithm (see Chapter 6).

The remainder of the chapter examines the relationship between reparameterizations and lower bounds. In Section 4.3, we demonstrate that lower bounds on the objective function can be derived directly from a given reparameterization; different reparameterizations will produce different lower bounds. Alternatively, for real-valued objective functions, lower bounds can be derived using duality and a linear program known as the MAP LP [47]. These two approaches produce similar lower bounds on the objective function. We focus on two types of lower bounds: lower bounds that are a concave function of a vector of messages and lower bounds that are a concave function of the potentials in the factorization. In Section 4.4, we demonstrate that these two types of lower bounds are equivalent for real-valued objective functions, generalizing the known results for pairwise factorizations in [21] and providing an alternative proof that does not require the existence of min-sum fixed points.

### 4.1 Admissibility and the Min-Sum Algorithm

Fixed points of the message update equations produce a reparameterization of the objective function, but an arbitrary vector of messages need not produce a new factorization. This difficulty is a direct consequence of having two types of messages (those passed from variables to factors and those passed from factors to variables). However, we could ensure admissibility by introducing a vector of messages, \( m' \), and rewriting the objective function as follows:

\[
f(x) = \sum_{i} \phi_i(x_i) + \sum_{\alpha \in \partial i} m'_{\alpha \rightarrow i}(x_i) + \sum_{\alpha} \psi_{\alpha}(x_{\alpha}) - \sum_{k \in a} m'_{\alpha \rightarrow k}(x_k) \tag{4.3}
\]
If the vector of messages is real-valued, this rewriting does not change the objective function. For our new vector $m'$, consider the following definitions for the beliefs:

$$b'_i(x_i) = \phi_i(x_i) + \sum_{\alpha \in \partial_i} m'_{\alpha \rightarrow i}(x_i) \quad (4.4)$$

$$b'_\alpha(x_\alpha) = \psi_\alpha(x_\alpha) + \sum_{k \in \alpha} [b'_k(x_k) - m'_{\alpha \rightarrow k}(x_k)] \quad (4.5)$$

With these definitions, we can express the objective function as

$$f(x) = \sum_i b'_i(x_i) + \sum_\alpha [b'_\alpha(x_\alpha) - \sum_{k \in \alpha} b'_k(x_k)]. \quad (4.6)$$

Any choice of real-valued factor to variable messages produces an alternative factorization of the objective function. We can ensure admissibility of the min-sum algorithm by defining the beliefs as in Equations 4.4 and 4.5.

### 4.2 The Splitting Reparameterization

The min-sum algorithm only produces reparameterizations of the objective function that have the same form as Equation 4.2. Many other reparameterizations in terms of messages are possible. For example, given a vector of non-zero reals, $c$, we can construct the following reparameterization of the objective function:

$$f(x) = \sum_i \phi_i(x_i) + \sum_{\alpha \in \partial_i} c_\alpha m_{\alpha \rightarrow i}(x_i) + \sum_\alpha \psi_\alpha(x_\alpha) - \sum_{k \in \alpha} c_\alpha c_k m_{\alpha \rightarrow k}(x_k) \quad (4.7)$$

$$= \sum_i \frac{c_i \phi_i(x_i)}{c_i} + \sum_{\alpha \in \partial_i} c_\alpha m_{\alpha \rightarrow i}(x_i) + \sum_\alpha \frac{c_\alpha \psi_\alpha(x_\alpha)}{c_\alpha} - \sum_{k \in \alpha} c_\alpha c_k m_{\alpha \rightarrow k}(x_k) \quad (4.8)$$

By analogy to the min-sum algorithm, we define the beliefs corresponding to this reparameterization as follows:

$$b_i(x_i) = \frac{\phi_i(x_i)}{c_i} + \sum_{\alpha \in \partial_i} c_\alpha m_{\alpha \rightarrow i}(x_i) \quad (4.9)$$

$$b_\alpha(x_\alpha) = \frac{\psi_\alpha(x_\alpha)}{c_\alpha} + \sum_{k \in \alpha} c_k [b'_k(x_k) - m_{\alpha \rightarrow k}(x_k)] \quad (4.10)$$

36
This allows us to rewrite the objective function as

\[
f(x) = \sum_i c_i b_i(x_i) + \sum_\alpha c_\alpha [b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k)]. \tag{4.11}
\]

Notice that if we choose \(c_i = 1\) for all \(i\) and \(c_\alpha = 1\) for all \(\alpha\) then we obtain the same reparameterization as the standard min-sum algorithm. We will call the reparameterization given by Equation 4.8 the splitting reparameterization (see Chapter 6).

### 4.3 Lower Bounds

Historically, concave lower bounds on the objective function were produced by analyzing duals of a particular linear programming formulation of the minimization problem. Alternatively, concave lower bounds on the objective function can be derived directly from factorizations of the objective function. As concave lower bounds can be optimized via many standard techniques, understanding the properties of these lower bounds will be important for both the convergence and the correctness of many local message passing algorithms.

#### 4.3.1 Lower Bounds and Reparameterization

We can use message reparameterizations to construct lower bounds on the objective function. For example, consider the following lower bound obtained from the splitting reparameterization:

\[
\min_x f(x) = \min_x \left[ \sum_i c_i b_i(x_i) + \sum_\alpha c_\alpha [b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k)] \right] \tag{4.12}
\]

\[
\geq \sum_i \min_x c_i b_i(x_i) + \sum_\alpha \min_{x_\alpha} c_\alpha [b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k)] \tag{4.13}
\]

Notice that this lower bound is a concave function of the message vector, \(m\), for any choice of the vector \(c\) such that each component is nonzero. Other concave lower bounds are possible using the same reparameterization:

\[
\min_x f(x) = \min_x \left[ \sum_i c_i (1 - \sum_{\alpha \in \partial i} c_\alpha) b_i(x_i) + \sum_\alpha c_\alpha b_\alpha(x_\alpha) \right] \tag{4.14}
\]

\[
= \min_x \left[ \sum_i c_i (1 - \sum_{\alpha \in \partial i} c_\alpha) b_i(x_i) + \sum_\alpha c_\alpha b_\alpha(x_\alpha) \right] \tag{4.15}
\]

\[
\geq \sum_i \min_{x_i} c_i (1 - \sum_{\alpha \in \partial i} c_\alpha) b_i(x_i) + \sum_\alpha \min_{x_\alpha} c_\alpha b_\alpha(x_\alpha) \tag{4.16}
\]
4.3.2 Lower Bounds and the MAP LP

Many authors have observed that, for finite state spaces (i.e. \( |X_i| < \infty \) for all \( i \)) and objective functions \( f : \prod_i X_i \to \mathbb{R} \), we can convert the optimization problem, \( \min_x f(x) \), into an equivalent integer program by choosing a factorization \( (\phi, \psi) \in \mathcal{F}_A(f) \) (see the definition in Section 2.2.2) and introducing an indicator vector \( \mu \) [57, 53]:

\[
\begin{align*}
\text{minimize} & \quad \sum_i \sum_{x_i} \mu_i(x_i) \phi_i(x_i) + \sum_\alpha \sum_{x_\alpha} \mu_\alpha(x_\alpha) \psi_\alpha(x_\alpha) \\
\text{subject to} & \quad \sum_{x_\alpha, i} \mu_\alpha(x_\alpha) = \mu_i(x_i) & \forall \alpha, i \in X, x_i \\
& \quad \sum_{x_i} \mu_i(x_i) = 1 & \forall i \\
& \quad \mu_i(x_i) \in \{0, 1\}, \mu_\alpha(x_\alpha) \in \{0, 1\} & \forall \alpha, i, x_i, x_\alpha
\end{align*}
\]

Recall that our original definition allowed \( f \) to take values in the set \( \mathbb{R} \cup \infty \). If the objective function takes the value \( \infty \) at some point \( x \in \prod_i X_i \), then, strictly speaking, the above construction is not technically and integer program. We can correct for this by replacing infinities with sufficiently large integers. This, of course, will change the objective function, but for sufficiently large integers, it will not change the location of the minimum. Consequently, many authors assume that \( f : \prod_i X_i \to \mathbb{R} \). This assumption is crucial to the results in this section:

**Assumption 2.** \( f : \prod_i X_i \to \mathbb{R} \) is a real-valued function over a finite domain. Note that, in this case, \( f \) factorizes over \( A \) if we can write \( f \) as a sum of real-valued potential functions \( \phi_i : X_i \to \mathbb{R} \) and \( \psi_\alpha : X_\alpha \to \mathbb{R} \) as follows:

\[
f(x) = \sum_{i=1}^n \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \psi_\alpha(x_\alpha)
\]

We will not need this assumption for all of the results in this work, and we will explicitly state when this assumption is being used. Indeed, the lower bounds constructed as in the last section do not require this assumption.

The above integer program can be relaxed into a linear program by allowing \( \mu_i \) and \( \mu_\alpha \) to be non-negative reals. The resulting linear program is typically referred to as the MAP LP. We note that the constraints can be written in matrix form as \( Ax = b \) such that the components of \( A \) and \( b \) are all integers. Consequently, any vertex of the polytope corresponding to the system of equations \( Ax = b \) must have all rational entries.

We can use the MAP LP and its dual formulations in order to construct lower bounds on the
objective function; different duals will produce different lower bounds. This approach produces lower bounds similar to the ones obtained in the last section and suggests a close relationship between duality and reparameterization. Many different lower bounds on the objective function have been derived using duality (e.g. see [14, 53]).

For the purposes of this work, any dual objective function that satisfies the conditions necessary for strong duality will suffice. With this in mind, we follow the approach in [47], and define the following optimization problem:

\[
\sup_{(\phi, \psi) \in S} \left[ \sum_i \min_{x_i} \phi_i(x_i) + \sum_{\alpha \in A} \min_{x_{\alpha}} \psi_{\alpha}(x_{\alpha}) \right]
\]

For certain choices of the set \( S \), the optimization in Equation 4.18 is dual to the MAP LP. As an example, let \( S = \mathcal{F}_A(f) \), the set of all factorizations of the objective function \( f(x) \) over the set \( A \).

**Lemma 4.3.1.** If \( S = \mathcal{F}_A(f) \), then the optimization problem in Equation 4.18 is dual to the MAP LP generated by any \((\phi', \psi') \in \mathcal{F}_A(f)\).

**Proof.** The proof follows from Lagrangian duality. Let \( LB(\phi, \psi) \) be the function to be maximized in Equation 4.18. We can construct a vector of Lagrange multipliers, \( \lambda \), for the constraint set \( \mathcal{F}_A(f) \) such that \( \lambda_x \) corresponds to the constraint that \( \sum_i \phi_i(x_i) + \sum_{\alpha} \psi_{\alpha}(x_{\alpha}) = f(x) \). The Lagrangian associated to the optimization in Equation 4.18 is then given by

\[
\mathcal{L}(\phi, \psi, \lambda) = \left[ \sum_i \min_{x_i} \phi_i(x_i) + \sum_{\alpha} \min_{x_{\alpha}} \psi_{\alpha}(x_{\alpha}) \right] + \sum_x \lambda_x \left[ f(x) - \sum_i \phi_i(x_i) - \sum_{\alpha} \psi_{\alpha}(x_{\alpha}) \right]
\]

The dual function, \( g(\lambda) \) is constructed by taking the supremum over all \( b \) of \( \mathcal{L}(\phi, \psi, \lambda) \): \( g(\lambda) = \sup_{(\phi, \psi)} \mathcal{L}(\phi, \psi, \lambda) \). Now, consider a single term in the sum: \( \min_{x_i} \phi_i(x_i) - \sum_x \lambda_x \phi_i(x_i) \). Define \( \mu_i(x_i) = \sum_{x_{\neq i}} \lambda_x \). With this definition, we have

\[
\min_{x_i} \phi_i(x_i) - \sum_x \lambda_x \phi_i(x_i) = \min_{x_i} \phi_i(x_i) - \sum_{x_i} \mu_i(x_i) \phi_i(x_i)
\]

\[
\leq (1 - \sum_{x_i: \mu_i(x_i) \geq 0} \mu_i(x_i)) \min_{x_i} \phi_i(x_i) - \sum_{x_i: \mu_i(x_i) < 0} \mu_i(x_i) \max_{x_i} \phi_i(x_i)
\]

This bound is always tight for some choice of the beliefs (e.g. let \( \phi_i(x_i) = \kappa \) for all \( x_i \)). This
gives us that sup \( \left[ \min_{x_i} \phi_i(x_i) - \sum_{x} \lambda_x \phi_i(x_i) \right] < \infty \) if and only if \( \sum_{x_i} \mu_i(x_i) = 1 \) and \( \mu_i(x_i) \geq 0 \) for all \( x_i \). A similar analysis can be done for the remaining two pieces of the sum, but these results can be inferred from the constraint \( \sum_{x_i} \mu_i(x_i) = 1 \) for all \( i \) and the following observation for each \( i \in \alpha \):

\[
\sum_{x_{\alpha \setminus i}} \mu_{\alpha}(x_{\alpha}) = \sum_{x_{\alpha \setminus i} \setminus x_{-\alpha}} \lambda_x = \sum_{x_{-i}} \lambda_x = \mu_i(x_i)
\]

Consequently, for any choice of \( \lambda \) such that for all \( i, \mu_i(x_i) \geq 0 \) and \( \sum_{x_i} \mu_i(x_i) = 1 \) and any choice of \((\phi', \psi') \in \mathcal{F}_A(f)\), we have the following:

\[
g(\lambda) \leq \sum_i \sup_{(\phi, \psi)} \left[ \min_{x_i} \phi_i(x_i) - \sum_{x} \lambda_x \phi_i(x_i) \right] + \sum_{i} \sup_{(\phi, \psi)} \left[ \min_{x_{\alpha}} \psi_{\alpha}(x_{\alpha}) - \sum_{x} \lambda_x \psi_{\alpha}(x_{\alpha}) \right] + \sum_{x} \lambda_x f(x)
\]

\[
= \sum_{i} \mu_i(x_i) \phi'_i(x_i) + \sum_{\alpha} \sum_{x_{\alpha}} \mu_{\alpha}(x_{\alpha}) \psi'_\alpha(x_{\alpha})
\]

Finally, this upper bound is tight. To see this, let \((\phi, \psi)\) be a uniform vector of constant beliefs. From this, we can conclude that the inf \( g(\lambda) = \inf_{\mu \in M} \sum_i \sum_{x_i} \mu_i(x_i) \phi_i(x_i) + \sum_{\alpha} \sum_{x_{\alpha}} \mu_{\alpha}(x_{\alpha}) \psi_{\alpha}(x_{\alpha}) \) where \( M \) is the set of all vectors \( \mu \) satisfying the constraints of the MAP LP.

As an another example, we can show that lower bounds of the type derived from the reparameterizations in Section 4.3.1 are also dual to the MAP LP. Given any factorization of \( f(x) = \sum_i \phi'_i(x_i) + \sum_{\alpha} \psi'_\alpha(x_{\alpha}) \) over \( \mathcal{A} \), let

\[
\mathcal{M}_A(\phi', \psi') = \{(\phi, \psi) \mid \forall i, x_i \in \mathcal{X}_i, \psi_i(x_i) = \phi'_i(x_i) + \sum_{\alpha \in \partial_i} m_{\alpha \rightarrow i}(x_i), \\
\forall \alpha \in \mathcal{A}, x_{\alpha} \in \mathcal{X}_{\alpha}, \psi_{\alpha}(x_{\alpha}) = \psi'_\alpha(x_{\alpha}) - \sum_{i \in \alpha} m_{\alpha \rightarrow i}(x_i) \text{ for some } m \}
\]

\( \mathcal{M}_A(\phi', \psi') \) corresponds to message reparameterizations of the factorization \((\phi', \psi')\).

**Lemma 4.3.2.** If \( S = \mathcal{M}_A(\phi', \psi') \), then the optimization problem in Equation 4.18 is dual to the MAP LP for the factorization \((\phi', \psi')\).
Proof. The proof follows from Lagrangian duality and arguments similar to those of Lemma 4.3.1.

There are many other choices of $\mathcal{S}$ such that the optimization problem defined in Equation 4.18 is dual to the MAP LP. If $\mathcal{S}$ is not empty and the optimization in Equation 4.18 is dual to the MAP LP for some factorization, then strong duality holds: the maximum of Equation 4.18 is equal to the minimum value of the MAP LP. Many of the convergent message passing algorithms that we investigate in this work can be viewed as coordinate ascent schemes over lower bounds of this form [47].

### 4.4 Factorizations and Message Reparameterizations

Reparameterization in terms of messages allows us to take a given factorization of the objective function and produce new factorizations. Under Assumption 2, the lower bounds produced by message reparameterizations in Lemma 4.3.2 are the same as those produced by factorizations in Lemma 4.3.1, and surprisingly, any two factorizations of the objective function can be related, up to a constant, via message reparameterizations:

**Theorem 4.4.1.** Let $f$ satisfy Assumption 2. For all $(\phi, \psi, \phi', \psi') \in \mathcal{F}_\mathcal{A}(f)$, there exists a vector of messages, $m^*$, such that

$$\phi_i(x_i) + \sum_{\alpha \in \partial i} m^*_{\alpha \rightarrow i}(x_i) = k_i + \phi'_i(x_i) \quad \text{for all } i \quad \text{and} \quad \phi_\alpha(x_\alpha) - \sum_{i \in \alpha} m^*_{\alpha \rightarrow i}(x_i) = k_\alpha + \phi'_\alpha(x_\alpha) \quad \text{for all } \alpha.$$

**Proof.** Let $g(x)$ be the zero function (i.e. $g(x) = 0$ for all $x$). The zero function factorizes over $\mathcal{A}$: $(\phi - \phi', \psi - \psi') \in \mathcal{F}_\mathcal{A}(g)$.

By Lemma 4.3.1 and strong duality, the minimum value of the MAP LP for this factorization must be equal to zero as $(0, 0) \in \mathcal{F}_\mathcal{A}(g)$ and $g(x) = 0$. Now, by Lemma 4.3.2 and the previous observation, there exists a vector of messages, $m^*$, such that

$$\sum_i \min_{x_i} \left[ \phi_i(x_i) - \phi'_i(x_i) + \sum_{\alpha \in \partial i} m^*_{\alpha \rightarrow i}(x_i) \right] + \sum_\alpha \min_{x_\alpha} \left[ \psi_\alpha(x_\alpha) - \psi'_\alpha(x_\alpha) - \sum_i m^*_{\alpha \rightarrow i}(x_i) \right] = 0.$$

Because $g(x)$ is a constant function, every assignment minimizes $g$. This implies that $\phi_i(x_i) + \sum_{\alpha \in \partial i} m^*_\alpha \rightarrow i(x_i) = \phi'_i(x_i) + k_i$ for all $i$ and some constant $k_i$ (similarly for each $\alpha$).

Theorem 4.4.1 could be interpreted as: finding "good" factorizations is equivalent to finding "good" message reparameterizations of some fixed factorization. Under Assumption 2, the sets in
lemmas 4.3.1 and 4.3.2 are equivalent in the sense that $\mathcal{F}_A(f) = \mathcal{M}_A(\phi, \psi)$ up to the addition of a constant vector for any $(\phi, \psi) \in \mathcal{F}_A(f)$. This equivalence suggests two different strategies for maximizing the lower bound. A similar theorem was proven in Lemma 6.3 of [21] for factorizations in which $|\alpha| \leq 2$ for all $\alpha \in A$. The proof therein relies on special properties of such factorizations and the existence of max-product fixed points (neither of which are used in the proof of Theorem 4.4.1).

When Assumption 2 does not hold, Theorem 4.4.1 is no longer valid. Consider the following objective function:

\begin{align*}
  f(x) &= -3x_1 - 2x_2 - 2x_1 - \log\{x_1 \neq x_2\} - \log\{x_1 \neq x_3\} - \log\{x_2 \neq x_3\} \\
  &\quad - \log\{x_1 + x_2 + x_3 \geq 0\} \\
  &= -3x_1 - 2x_2 - 2x_1 - \log\{x_1 \neq x_2\} - \log\{x_1 \neq x_3\} - \log\{x_2 \neq x_3\} \\
  &\quad - \log\{x_1 + x_2 + x_3 \leq 1\}
\end{align*}

for $x_1, x_2, x_3 \in \{0, 1\}$. These two factorizations cannot be reparameterized into each other via messages as in Theorem 4.4.1. To see why this could be the case, observe that if the infinities in these two different factorizations are converted into large integers, then these two functions are no longer factorizations of the same objective function.
Chapter 5

Lower Bounds and Optimality

In the Chapter 4, we saw that, given a vector of admissible beliefs, we can produce concave lower bounds on the objective function. The discussion in this chapter will focus on the properties of admissible and min-consistent beliefs with relation to these lower bounds. As such, these results will be applicable to a variety of algorithms, such as the min-sum algorithm, that produce beliefs with these two properties.

Recall that, given a vector of admissible and min-consistent beliefs, we construct a fixed-point estimate \( x^* \) such that \( x^*_i \in \arg \min b_i \). If the objective function had a unique global minimum and the fixed point beliefs were the true min-marginals, then \( x^* \) would indeed be the global minimum. Now, suppose that the \( b_i \) are not the true min-marginals. What can we say about the optimality of any vector \( x^* \) such that \( x^*_i \in \arg \min b_i \)? What can we say if there is a unique vector \( x^* \) with this property? In this chapter, we explore these questions by examining the lower bounds and reparameterizations discussed in Chapter 4. Our primary tool for answering these questions will be min-consistency and the following lemma:

**Lemma 5.0.2.** Let \( b \) be a vector of min-consistent beliefs for the splitting reparameterization. If there exists a unique estimate \( x^* \) that minimizes \( b_i(x_i) \) for each \( i \), then \( x^* \) also minimizes \( b_\alpha(x_\alpha) \) and, for any \( i \in \alpha \), \( x^* \) minimizes \( b_\alpha(x_\alpha) - b_i(x_i) \).

**Proof.** Because the beliefs are min-consistent for any \( i \in \alpha \), we have:

\[
\min_{x_\alpha \setminus i} b_\alpha(x_\alpha) = \kappa + b_i(x_i)
\]
From this, we can conclude that there is a some $x_\alpha$ that minimizes $b_\alpha$ with $x_i = x_i^*$. Further, because the minimum is unique for each $b_i$, $x_i^*$ must minimize $b_\alpha$. Now fix a vector $x$ and consider

$$b_\alpha(x_\alpha^*) - b_i(x_i^*) = \min_{x_\alpha/i} b_\alpha(x_i^*, x_\alpha/i) - b_i(x_i)$$

$$= \min_{x_\alpha/i} b_\alpha(x_i, x_\alpha/i) - b_i(x_i)$$

$$\leq b_\alpha(x_\alpha) - b_i(x_i)$$

This lemma will be a crucial building block of many of the theorems in this work, and many variants of this lemma have been proven in the literature (e.g. Lemma 4 in [51] and Theorem 1 in [58]). We also note that this lemma continues to hold for any $x^*$ that simultaneously minimizes $b_i$ for all $i$ and $b_\alpha$ for all $\alpha$:

**Definition 5.0.1.** $x^*$ simultaneously minimizes a vector of beliefs, $b$, if $x_i^* \in \arg\min_{x_i} b_i(x_i)$ for all $i$ and $x_\alpha^* \in \arg\min_{x_\alpha} b_\alpha(x_\alpha)$ for all $\alpha$.

Unlike locally decodable beliefs, we may not be able to efficiently construct such an $x^*$ given a vector of admissible and min-consistent beliefs.

Using Lemma 5.0.2 and admissibility, we can convert questions about the optimality of the vector $x$ into questions about the choice of reparameterization. Specifically, we focus on the splitting reparameterization:

$$f(x) = \sum_i c_i b_i(x_i) + \sum_\alpha c_\alpha [b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k)]$$  \hspace{1cm} (5.1)

In Sections 5.1 and 5.2, we show how to choose the parameter vector $c$ in order to guarantee the local or global optimality of any estimate that simultaneously minimizes a vector of admissible and min-consistent beliefs. We conclude this chapter by examining restricted notions of optimality in Section 5.3.

### 5.1 Local Optimality

A function $f$ is said to have a local optimum at the point $x \in \prod_i \mathcal{X}_i$ if there is some neighborhood of the point $x$ such that $f$ does not increase in that neighborhood.
The definition of neighborhood is metric dependent, and in the interest of keeping our results applicable to a wide variety of spaces, we choose the metric to be the Hamming distance. For any two vectors \( x, y \in \prod_i X_i \), the Hamming distance is the number of entries in which the two vectors differ. For the purposes of this paper, we will restrict our definition of local optimality to vectors within Hamming distance one:

**Definition 5.1.1.** \( x \in \prod_i X_i \) is a local minimum of the objective function, \( f \), if for every vector \( y \) that has at most one entry different from \( x \), \( f(x) \leq f(y) \).

We will show that there exist choices of the parameters for which any estimate, extracted from a vector of admissible and min-consistent beliefs, that simultaneously minimizes all of the beliefs is guaranteed to be locally optimal with respect to the Hamming distance. In order to prove such a result, we first need to relate the minima of the admissible and min-consistent beliefs to the minima of the objective function.

Let \( b \) be a vector of admissible beliefs for the function \( f \) with parameter vector \( c \). Define \( -j = \{1, \ldots, n\} \setminus \{j\} \). For a fixed \( x_{-j} \), we can lower bound the optimum value of the objective function as follows:

\[
\min_{x_j} f(x_j, x_{-j}) = \min_{x_j} \left[ \kappa + \sum_i c_i b_i(x_i) + \sum_{\alpha} c_\alpha \left[ b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k) \right] \right] 
\]

\[
= g_j(x_{-j}) + \min_{x_j} \left[ c_j b_j(x_j) + \sum_{\alpha \in \partial j} c_\alpha \left[ b_\alpha(x_\alpha) - c_j b_j(x_j) \right] \right] 
\]

\[
= g_j(x_{-j}) + \min_{x_j} \left[ (1 - \sum_{\alpha \in \partial j} c_\alpha) c_j b_j(x_j) + \sum_{\alpha \in \partial j} c_\alpha b_\alpha(x_\alpha) \right] 
\]

\[
\geq g_j(x_{-j}) + \min_{x_j} \left[ (1 - \sum_{\alpha \in \partial j} c_\alpha) c_j b_j(x_j) + \sum_{\alpha \in \partial j} \min_{x_j} [c_\alpha b_\alpha(x_\alpha)] \right] 
\]

where \( g_j(x_{-j}) \) is the part of the reparameterization that does not depend on \( x_j \).

The last inequality is tight whenever there is a value of \( x_j \) that simultaneously minimizes each component of the sum. If the coefficients of \( b_j \) and each of the \( b_\alpha \) in Equation 5.5 were non-negative, then we could rewrite this bound as

\[
\min_{x_j} f(x) \geq g_j(x_{-j}) + \left(1 - \sum_{\alpha \in \partial j} c_\alpha \right) c_j \min_{x_j} [b_j(x_j)] + \sum_{\alpha \in \partial j} c_\alpha \min_{x_j} [b_\alpha(x_\alpha)] 
\]

which depends on the minima of each of the beliefs. Recall from Lemma 5.0.2 that if \( b \) is locally decodable to \( x^* \), then \( x^* \) must simultaneously minimize \( b_i, b_\alpha \), and, for \( i \in \alpha, b_\alpha - b_i \). So, in general,
we want to know if we can write

\[
(1 - \sum_{\alpha \in \partial j} c_{\alpha} c_{j}(x_{j}) + \sum_{\alpha \in \partial j} c_{\alpha} b_{\alpha}(x_{\alpha}) = d_{j} b_{j}(x_{j}) + \sum_{\alpha \in \partial j} d_{\alpha\alpha} b_{\alpha}(x_{\alpha}) + \sum_{\alpha \in \partial j} d_{\alpha j} [b_{\alpha}(x_{\alpha}) - b_{j}(x_{j})]
\]

for some vector of non-negative constants \(d\). This motivates the following definition:

**Definition 5.1.2.** A function, \(h\), can be written as a conical combination of the beliefs if there exists a vector of non-negative reals, \(d\), such that

\[
h(x) = \kappa + \sum_{i, \alpha \in \alpha} d_{i\alpha} (b_{\alpha}(x_{\alpha}) - b_{i}(x_{i})) + \sum_{\alpha} d_{\alpha\alpha} b_{\alpha}(x_{\alpha}) + \sum_{i} d_{ii} b_{i}(x_{i})
\]

The set of all conical combinations of a collection of vectors in \(R^{n}\) forms a cone in \(R^{n}\) in the same way that a convex combination of vectors in \(R^{n}\) forms a convex set in \(R^{n}\).

The above definition is very similar to the definition of “provably convex” in [58]. There, an entropy approximation is provably convex if it can be written as a conical combination of the entropy functions corresponding to each of the factors. In contrast, our approach follows from a reparameterization of the objective function. Putting all of the above ideas together, we have the following theorem:

**Theorem 5.1.1.** Let \(b\) be a vector of admissible and min-consistent beliefs for the function \(f\) with a corresponding weighting vector of non-zero real numbers, \(c\), such that for all \(i\), \(c_{i} b_{i}(x_{i}) + \sum_{\alpha \in \partial i} c_{\alpha} [b_{\alpha}(x_{\alpha}) - c_{i} b_{i}(x_{i})]\) can be written as a conical combination of the beliefs. If the beliefs are locally decodable to \(x^{*}\), then \(x^{*}\) is a local minimum (with respect to the Hamming distance) of the objective function.

**Proof.** Choose a \(j \in \{1, \ldots, n\}\). By assumption, the portion of the objective function that depends on the variable \(x_{j}\) can be written as a conical combination of the beliefs. By admissibility, we can
\[
\begin{align*}
    f(x^*) &= \kappa + \sum_i c_ib_i(x_i^*) + \sum_{\alpha} c_{\alpha}\left[b_{\alpha}(x_{\alpha}^*) - \sum_{k\in\alpha} c_kb_k(x_k^*)\right] \\
    &= g_j(x_{-j}^*) + c jb_j(x_j^*) + \sum_{\alpha\in\partial j} c_{\alpha}\left[b_{\alpha}(x_{\alpha}^*) - c_j b_j(x_j^*)\right] \\
    &= g_j(x_{-j}^*) + d_{jj} b_j(x_j^*) + \sum_{\alpha\in\partial j} d_{\alpha\alpha} b_{\alpha}(x_{\alpha}^*) + \sum_{\alpha\in\partial j} d_{\alpha j} [b_{\alpha}(x_{\alpha}^*) - b_j(x_j^*)] \\
    &\leq g_j(x_{-j}^*) + d_{jj} b_j(x_j^*) + \sum_{\alpha\in\partial j} d_{\alpha\alpha} b_{\alpha}(x_{\alpha}^*, x_{\alpha\setminus j}^*) + \sum_{\alpha\in\partial j} d_{\alpha j} [b_{\alpha}(x_{\alpha}^*, x_{\alpha\setminus j}^*) - b_j(x_j)] \\
    &= f(x_j, x_{-j}^*)
\end{align*}
\]
for any \(x_j \in X_j\) where the inequality follows from Lemma 5.0.2. We can repeat this proof for each \(j \in \{1, \ldots, n\}\). \(\square\)

Theorem 5.1.1 tells us that, under suitable choices of the parameters, no vector \(x\) within Hamming distance one of \(x^*\) can decrease the objective function.

For a differentiable function \(f\), if \(c_i \neq 0\) for all \(i\) and \(c_\alpha \neq 0\) for all \(\alpha\), then we can always infer that the gradient of \(f\) at the point \(x^*\) must be zero. Consequently, \(x^*\) is either a local minimum, a local maximum, or a saddle point of \(f\). If \(c\) satisfies the conditions Theorem 5.1.1, then by the second derivative test and the observation that the function can only increase in value along the coordinate axes, \(x^*\) is either a local minimum or a saddle point of \(f\). Similarly, for a convex differentiable function \(f\), if \(c_i \neq 0\) for all \(i\) and \(c_\alpha \neq 0\) for all \(\alpha\), then \(x^*\) minimizes \(f\):

**Corollary 5.1.2.** Let \(b\) be a vector of admissible and min-consistent beliefs for a convex differentiable function \(f\) with a corresponding weighting vector of non-zero real numbers, \(c\). If the beliefs are locally decodable to \(x^*\), then \(x^*\) is a global minimum of the objective function.

Lastly, we can check that choosing \(c_i = 1\) for all \(i\) and \(c_\alpha = 1\) for all \(i\) always satisfies the conditions of Theorem 5.1.1. Consequently, if the converged beliefs corresponding to the min-sum algorithm are locally decodable to \(x^*\), then \(x^*\) corresponds to local optima of the objective function:

**Corollary 5.1.3.** Let \(b\) be a vector of admissible and min-consistent beliefs produced by the min-sum algorithm. If the beliefs are locally decodable to \(x^*\), then \(x^*\) is a local minimum (with respect to the Hamming distance) of the objective function.
5.2 Global Optimality

We now extend the approach of the previous section to show that there are choices of the vector \( c \) that guarantee the global optimality of any unique estimate produced from admissible and min-consistent beliefs. As before, suppose \( b \) is a vector of admissible beliefs for the function \( f \). If \( f \) can be written as a conical combination of the beliefs, then we can lower bound the optimal value of the objective function as follows:

\[
\min_x f(x) = \min_x \left[ \kappa + \sum_i c_i b_i(x_i) + \sum_{\alpha} c_{\alpha} \left[ b_{\alpha}(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k) \right] \right]
\]  

(5.7)

\[
= \min_x \left[ \kappa + \sum_{i, \alpha : i \in \alpha} d_{i\alpha} (b_{\alpha}(x_\alpha) - b_i(x_i)) + \sum_{\alpha} d_{\alpha\alpha} b_{\alpha}(x_\alpha) + \sum_i d_i b_i(x_i) \right]
\]  

(5.8)

\[
\geq \kappa + \sum_{i, \alpha : i \in \alpha} d_{i\alpha} \min_x (b_{\alpha}(x_\alpha) - b_i(x_i)) + \sum_{\alpha} d_{\alpha\alpha} \min_x b_{\alpha}(x_\alpha) + \sum_i d_i \min_x b_i(x_i)
\]  

(5.9)

This analysis provides us with our first global optimality result. We note that the following theorem also appears as Theorem 1 in [30], and Theorem 2 in [53] provides a similar proof for the TRMP algorithm.

**Theorem 5.2.1.** Let \( b \) be a vector of admissible and min-consistent beliefs for the function \( f \) with a corresponding weighting vector of non-zero real numbers, \( c \), such that \( f \) can be written as a conical combination of the beliefs. If the beliefs are locally decodable to \( x^* \), then \( x^* \) minimizes the objective function.

**Proof.** Choose \( x \in \prod_i X_i \). Using Definition 5.1.2 and Lemma 5.0.2, we can write

\[
f(x^*) = \kappa + \sum_{i, \alpha : i \in \alpha} d_{i\alpha} (b_{\alpha}(x_\alpha) - b_i(x_i)) + \sum_{\alpha} d_{\alpha\alpha} b_{\alpha}(x_\alpha) + \sum_i d_i b_i(x_i)
\]  

(5.10)

\[
\leq \kappa + \sum_{i, \alpha : i \in \alpha} d_{i\alpha} (b_{\alpha}(x_\alpha) - b_i(x_i)) + \sum_{\alpha} d_{\alpha\alpha} b_{\alpha}(x_\alpha) + \sum_i d_i b_i(x_i)
\]  

(5.11)

\[
= f(x)
\]  

(5.12)

As a corollary, Theorem 5.2.1 also provides us with a simple proof that the standard min-sum algorithm is correct on a tree:

**Corollary 5.2.2.** Suppose the factor graph is a tree. If the admissible and min-consistent beliefs produced by the standard min-sum algorithm are locally decodable to \( x^* \), then \( x^* \) is the global
Proof. Let $b$ be the vector of min-consistent and admissible beliefs obtained from running the standard min-sum algorithm. Choose a node $r \in G$ and consider the factor graph as a tree rooted at a variable node, $r$. Let $p(\alpha)$ denote the parent of factor node $\alpha \in G$. We can now write,

$$f(x) = \sum_i b_i(x_i) + \sum_{\alpha \in A} \left[ b_\alpha(x_\alpha) - \sum_{k \in \alpha} b_k(x_k) \right]$$

$$= b_r(x_r) + \sum_{\alpha \in A} \left[ b_\alpha(x_\alpha) - b_{p(\alpha)}(x_{p(\alpha)}) \right]$$

Hence, we can conclude that $f$ can be written as a conical combination of the beliefs and apply Theorem 5.2.1. □

Given Theorem 5.2.1, starting with the vector $d$ seems slightly more natural than the starting with the vector $c$. Consider any non-negative real vector $d$, we now show that we can find a vector $c$ such that $f$ has a conical decomposition in terms of $d$ provided $d$ satisfies a mild condition. Choose the vector $c$ as follows:

$$c_\alpha = d_{\alpha \alpha} + \sum_{i \in \alpha} d_{i \alpha}$$

(5.13)

$$c_i = \frac{d_{ii} - \sum_{\alpha \in \partial i} d_{i \alpha}}{1 - \sum_{\alpha \in \partial i} c_\alpha}$$

(5.14)

These equations are valid whenever $1 - \sum_{\alpha \in \partial i} c_\alpha \neq 0$. Note that any valid reparameterization must have $c_i \neq 0$ and $c_\alpha \neq 0$ for all $\alpha$ and $i$. Hence, $d_{\alpha \alpha} + \sum_{i \in \alpha} d_{i \alpha} \neq 0$ and $d_{ii} \neq \sum_{\alpha \in \partial i} d_{i \alpha}$.

Now, for this choice of $c$, we have:

$$f(x) = \kappa + \sum_i c_i b_i(x_i) + \sum_{\alpha} c_\alpha \left[ b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k) \right]$$

(5.15)

$$= \kappa + \sum_i c_i (1 - \sum_{\alpha \in \partial i} c_\alpha) b_i(x_i) + \sum_{\alpha} c_\alpha b_\alpha(x_\alpha)$$

(5.16)

$$= \kappa + \sum_i (d_{ii} - \sum_{\alpha \in \partial i} d_{i \alpha}) b_i(x_i) + \sum_{\alpha} (d_{\alpha \alpha} + \sum_{i \in \alpha} d_{i \alpha}) b_\alpha(x_\alpha)$$

(5.17)

$$= \kappa + \sum_i d_{ii} b_i(x_i) + \sum_{\alpha} d_{\alpha \alpha} b_\alpha(x_\alpha) + \sum_{i, \alpha : i \in \alpha} d_{i \alpha} (b_\alpha(x_\alpha) - b_i(x_i))$$

(5.18)

In the case that $1 - \sum_{\alpha \in \partial i} c_\alpha = 0$, $c_i$ can be chosen to be any non-zero real. Again, any valid reparameterization must have $c_i \neq 0$ and $c_\alpha \neq 0$ for all $\alpha$ and $i$. Hence, $d_{\alpha \alpha} + \sum_{i \in \alpha} d_{i \alpha} \neq 0$, 49
but, unlike the previous case, we must have \( d_i - \sum_{\alpha \in \partial_i} d_{i\alpha} = 0 \). The remainder of the proof then follows exactly as above.

We now address the following question: given a factorization of the objective function \( f \), does there exist a choice of the vector \( c \) which guarantees that any estimate obtained from locally decodable beliefs minimizes the objective function? The answer to this question is yes, and we will provide a simple condition on the vector \( c \) that will ensure this.

Again, suppose \( b \) is a vector of admissible beliefs for the function \( f \). We can lower bound \( f \) as

\[
\min_x f(x) = \min_x \left[ \kappa + \sum_i c_i b_i(x_i) + \sum_{\alpha} c_{\alpha} \left[ b_{\alpha}(x_{\alpha}) - \sum_{k \in \alpha} c_k b_k(x_k) \right] \right] \tag{5.19}
\]

\[
= \min_x \left[ \kappa + \sum_i \left( \sum_{\alpha \in \partial_i} \sum_{\alpha} \right) c_i b_i(x_i) + \sum_{\alpha} c_{\alpha} b_{\alpha}(x_{\alpha}) \right] \tag{5.20}
\]

\[
\geq \kappa + \sum_i \min_{x_i} \left[ \left( \sum_{\alpha \in \partial_i} \sum_{\alpha} \right) c_i b_i(x_i) \right] + \sum_{\alpha} \min_{x_{\alpha}} c_{\alpha} b_{\alpha}(x_{\alpha}) \tag{5.21}
\]

Observe that if \( (1 - \sum_{\alpha \in \partial_i} c_{\alpha}) c_i \geq 0 \) for all \( i \) and \( c_{\alpha} \geq 0 \) for all \( \alpha \), then we can further rewrite the bound as:

\[
\min_x f(x) \geq \kappa + \sum_i \left( \sum_{\alpha \in \partial_i} \sum_{\alpha} \right) c_i \min_{x_i} \left[ b_i(x_i) \right] + \sum_{\alpha} c_{\alpha} \min_{x_{\alpha}} \left[ b_{\alpha}(x_{\alpha}) \right] \tag{5.22}
\]

This analysis yields the following theorem:

**Theorem 5.2.3.** Let \( b \) be a vector of admissible and min-consistent beliefs for the function \( f \) with a corresponding weighting vector of non-zero real numbers, \( c \), such that

1. For all \( i \), \( (1 - \sum_{\alpha \in \partial_i} c_{\alpha}) c_i \geq 0 \)

2. For all \( \alpha \), \( c_{\alpha} > 0 \)

If the beliefs are locally decodable to \( x^* \), then \( x^* \) minimizes the objective function.

**Proof.** By admissibility, we can write

\[
f(x) = \sum_i c_i b_i(x_i) + \sum_{\alpha} c_{\alpha} \left[ b_{\alpha}(x_{\alpha}) - \sum_{k \in \alpha} c_k b_k(x_k) \right]
\]

\[
= \sum_i \left[ \left( \sum_{\alpha \in \partial_i} \sum_{\alpha} \right) c_i b_i(x_i) \right] + \sum_{\alpha} c_{\alpha} b_{\alpha}(x_{\alpha})
\]

Now, by assumption, \( (1 - \sum_{\alpha \in \partial_i} c_{\alpha}) c_i \) and \( c_{\alpha} \) are non-negative real numbers. Therefore, by Lemma 50...
5.0.2, we can conclude that the assignment $x^*$ simultaneously minimizes each of the beliefs and hence minimizes the function $f$.

This result is quite general; for any choice of $c$ such that $c_\alpha > 0$ for all $\alpha \in \mathcal{A}$, there exists a choice of $c_i$ for each $i$ such that the conditions of the above theorem are satisfied. The following corollary is an immediate consequence of this observation and Theorem 5.2.3:

**Corollary 5.2.4.** Given any function $f(x) = \sum_i \phi_i(x_i) + \sum_\alpha \psi_\alpha(x_\alpha)$, there exists a choice of a non-zero parameter vector $c$ such that for any vector of admissible and min-consistent beliefs for the function $f$, if the beliefs are locally decodable to $x^*$, then this estimate minimizes $f$.

Up to this point, we have been assuming that the beliefs are locally decodable when, in fact, all of the previous theorems are equally valid for any vector that simultaneously minimizes all of the beliefs. However, finding such a vector may be difficult outside of special cases.

Lastly, we note that there are choices of the parameters for which we are guaranteed local optimality but not global optimality. The difference between Theorem 5.1.1 and Theorem 5.2.1 is that the former only requires that the part of the reparameterization depending on a single variable can be written as a conical combination of the beliefs, whereas the latter requires the entire reparameterization to be a conical combination of the beliefs. The standard min-sum algorithm always guarantees local optimality, and there are applications for which the algorithm is known to produce local optima that are not globally optimal [56].

### 5.3 Other Notions of Optimality

Other notions of local optimality arise when we examine the correctness of the min-sum algorithm when restricted to subgraphs of the factor graph. To make this precise, we need to define a restricted reparameterization.

**Definition 5.3.1.** A factor-induced subgraph of a factor graph $G$ is a subgraph of $G$ formed by taking a subset of the factor nodes and all of their adjacent variable nodes.

Let $G$ be a factor graph. Given a factor-induced subgraph $H \subseteq G$, the reparameterization restricted to $H$, denoted $R(x_H,H)$, is given by

$$R(x_H,H) = \sum_{i \in H} c_i b_i(x_i) + \sum_{\alpha \in H} c_\alpha \left[ b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k) \right]$$  (5.23)
By definition, $R(x_G, G) = f(x)$. Under certain conditions on the choice of subgraph, the converged estimates may correspond to global optima of the restricted reparameterization, even if the estimate is not a global optimum of the objective function.

If the standard min-sum algorithm converges and the beliefs are locally decodable to $x^*$, then $x^*$ is locally optimal in the following sense: for any factor-induced subgraph $H$ containing at most one cycle per connected component, $x_H^*$ minimizes $R(x_H, H)$. This result was originally proven for node induced subgraphs in [56] and subsequently strengthened to general subgraphs in [51]. The general result, known as tree-plus-cycle optimality, for the min-sum algorithm follows directly from Lemma 5.0.2 and the observation that $R(x_H, H)$ can be written as a conical combination of the beliefs for any $H$ containing at most one cycle per connected component.

When the factor graph consists of only pairwise factors, this notion of local optimality coincides with the more traditional definition in Section 5.1. To see this, notice that, for pairwise factor graphs, the collection of nodes formed by taking some variable node $i$ and every node in its two-hop neighborhood forms a subtree, $T$, of the factor graph. This tree is a factor-induced subgraph and it contains every part of the reparameterization that depends on the variable $x_i$. By tree-plus-cycle optimality, $x^*$ minimizes $R(x_T, T)$. As a result, we observe that if we change only the value of $x_i^*$, then we cannot decrease the value of $R$, and because $R(x_T, T)$ contains every portion of $f$ that depends on the variable $x_i$, we cannot decrease the objective function.

However, if the factorization is not pairwise, then the two-hop neighborhood of every node is not necessarily cycle free (see Figure 5.1). Consequently, the notion of tree-plus-cycle optimality need not correspond to Definition 5.1.1 for graphs where the factorization is not pairwise.

Tree-plus-cycle optimality is intrinsically tied to the min-sum algorithm. We can generalize this result to the splitting reparameterization by replacing the notion of cycle free by the notion of a conical decomposition:

**Lemma 5.3.1.** Let $b$ be a vector of admissible and min-consistent beliefs for the objective function
$f$ with parameter vector $c$ such that $b$ is locally decodable to $x^*$. If $H$ is a factor-induced subgraph and $R(x_H, H)$ can be written as a conical combination of the beliefs, then $x^*_H$ minimizes $R(x_H, H)$.

Proof. $R(x_H, H)$ can be written as a conical combination of the beliefs. Consequently, $x^*_H$ must minimize it by Lemma 5.0.2. \qed

Tree plus cycle optimality is then a special case of this theorem for a specific choice of the parameter vector. For different choices of the parameter vector, the estimate produced by the splitting algorithm will be optimal on different subgraphs.
Chapter 6

The Splitting Algorithm

In this chapter, we show how to combine the reparameterizations from Chapter 4 and the optimality conditions from Chapter 5 in order to develop a generalization of the min-sum algorithm that under certain conditions is convergent and correct. In Section 6.1, we describe a distributed, local message passing algorithm for the splitting reparameterization defined in Section 4.2. In Section 6.2, we discuss convergent asynchronous and synchronous versions of the splitting algorithm as well as alternative strategies to maximize the lower bound. Finally, in Section 6.3, we provide a derivation of the splitting algorithm from the standard min-sum algorithm. In subsequent chapters, we will discuss the advantages that the splitting algorithm has over other convergent and correct message passing algorithms as well as its advantages over the traditional min-sum algorithm.

6.1 The Algorithm

Recall that the splitting reparameterization is given by

\[ f(x) = \sum_i c_i b_i(x_i) + \sum_{\alpha} c_{\alpha} \left[ b_{\alpha}(x_{\alpha}) - \sum_{k \in \alpha} c_k b_k(x_k) \right] \]  

(6.1)

where the beliefs are defined via

\[ b_i(x_i) = \frac{\phi_i(x_i)}{c_i} + \sum_{\alpha \in \partial i} c_{\alpha} m_{\alpha \rightarrow i}(x_i) \]  

(6.2)

\[ b_{\alpha}(x_{\alpha}) = \frac{\psi_{\alpha}(x_{\alpha})}{c_{\alpha}} + \sum_{k \in \alpha} c_k [b_k(x_k) - m_{\alpha \rightarrow k}(x_k)] \]  

(6.3)

54
Algorithm 2 Synchronous Splitting Algorithm

1: Initialize the messages to some finite vector.
2: For iteration $t = 1, 2, ...$ update the messages as follows

$$m_{i \rightarrow \alpha}^t(x_i) = \kappa + \frac{\phi_i(x_i)}{c_i} + (c_{\alpha} - 1)m_{\alpha \rightarrow i}^{t-1}(x_i) + \sum_{\beta \in \partial \alpha \setminus \alpha} c_{\beta} m_{\beta \rightarrow i}^{t-1}(x_i)$$

$$m_{\alpha \rightarrow i}^t(x_i) = \kappa + \min_{x_{\alpha \setminus i}} \left[ \frac{\psi_{\alpha}(x_{\alpha})}{c_{\alpha}} + (c_i - 1)m_{i \rightarrow \alpha}^{t-1}(x_i) + \sum_{k \in \alpha \setminus i} c_k m_{k \rightarrow \alpha}^{t-1}(x_k) \right]$$

In light of the results of Chapter 5, we want to design a message passing algorithm for the splitting reparameterization that produces admissible and min-consistent beliefs upon convergence. Given our definition of the beliefs, admissibility is automatic. All that remains is to define the message updates so that the fixed-point beliefs are min-consistent. Specifically, upon convergence, we require that

$$\min_{x_{\alpha \setminus i}} b_{\alpha}(x_{\alpha}) = \kappa + b_i(x_i). \quad (6.4)$$

In principal, there are many different message updates that will guarantee min-consistency upon convergence. For example, we could solve Equation 6.4 for $m_{\alpha \rightarrow i}(x_i)$. In Algorithm 2, we present a pair of message updates that closely mirror the original min-sum message updates (for a more detailed explanation of this relationship see Section 6.3). The fixed points of these message updates are guaranteed to be min-consistent:

**Lemma 6.1.1.** Let $m$ be a fixed point of the message updates in Algorithm 2. The corresponding beliefs, $b$, are min-consistent.
Proof. By admissibility, we can write,
\[
\min_{x_{\alpha \setminus i}} b_\alpha(x_\alpha) = \min_{x_{\alpha \setminus i}} \frac{\psi_\alpha(x_\alpha)}{c_\alpha} + \sum_{k \in \alpha} c_k \left[ \frac{\phi_k(x_k)}{c_k} + (c_\alpha - 1)m_{\alpha \rightarrow k}(x_k) + \sum_{\beta \in \partial_k \setminus \alpha} c_\beta m_{\beta \rightarrow k}(x_k) \right]
\]
\[
= \min_{x_{\alpha \setminus i}} \frac{\psi_\alpha(x_\alpha)}{c_\alpha} + \sum_{k \in \alpha} c_k m_{k \rightarrow \alpha}(x_k)
\]
\[
= \min_{x_{\alpha \setminus i}} \frac{\psi_\alpha(x_\alpha)}{c_\alpha} + \sum_{k \in \alpha \setminus i} c_k m_{k \rightarrow \alpha}(x_k) + c_{i \rightarrow \alpha}(x_i)
\]
\[
= m_{\alpha \rightarrow i}(x_i) + m_{i \rightarrow \alpha}(x_i)
\]
\[
= m_{\alpha \rightarrow i}(x_i) + \left[ \frac{\phi_i(x_i)}{c_i} + (c_\alpha - 1)m_{\alpha \rightarrow i}(x_i) + \sum_{\beta \in \partial_i \setminus \alpha} c_\beta m_{\beta \rightarrow i}(x_i) \right]
\]
\[
= \frac{\phi_i(x_i)}{c_i} + \sum_{\beta \in \partial_i} c_\beta m_{\beta \rightarrow i}(x_i)
\]
\[
= b_i(x_i)
\]

Again, for any real-valued objective function \(f\), there always exists a fixed point of the min-sum message passing updates (see Theorem 2 of [51]). The proof of this statement can be translated almost exactly for our message passing updates, and we will not reproduce it here.

6.1.1 Computation Trees

The computation trees produced by the synchronous splitting algorithm are different from their predecessors. Again, the computation tree captures the messages that would need to be passed in order to compute \(b_i^t(x_i)\). However, the messages that are passed in the new algorithm are multiplied by a non-zero constant. As a result, the potential at a node \(u\) in the computation tree corresponds to some potential in the original graph multiplied by a constant that depends on all of the nodes above \(u\) in the computation tree. We summarize the changes as follows:

1. The message passed from \(i\) to \(\alpha\) may now depend on the message from \(\alpha\) to \(i\) at the previous time step. As such, we now form the time \(t+1\) computation tree from the time \(t\) computation tree by taking any leaf \(u\), which is a copy of node \(v\) in the factor graph, of the time \(t\) computation tree, creating a new node for every \(w \in \partial v\), and connecting \(u\) to these new nodes. As a result, the new computation tree rooted at node \(u\) of depth \(t\) contains at least all of the non-backtracking walks of length \(t\) in the factor graph starting from \(u\) and, at most,
all walks of length $t$ in the factor graph starting at $u$.

2. The messages are weighted by the elements of $c$. This changes the potentials at the nodes in the computation tree. For example, suppose the computation tree was rooted at variable node $i$ and that $b_i$ depends on the message from $\alpha$ to $i$. Because $m_{\alpha \rightarrow i}$ is multiplied by $c_\alpha$ in $b_i$, every potential along this branch of the computation tree is multiplied by $c_\alpha$. To make this concrete, we can associate a weight to every edge of the computation tree that corresponds to the constant that multiplies the message passed across that edge. To compute the new potential at a variable node $i$ in the computation tree, we now need to multiply the corresponding potential $\phi_i c_i$ by each of the weights corresponding to the edges that appear along the path from $i$ to the root of the computation tree. An analogous process can be used to compute the potentials at each of the factor nodes. The computation tree produced by the splitting algorithm at time $t = 2$ for the factor graph in Figure 2.3 is pictured in Figure 6.1. Compare this with computation tree produced by the standard min-sum algorithm in Figure 2.4.

If we make these adjustments and all of the weights are positive, then the belief, $b'_i(x_i)$, at node $i$ at time $t$ is given by the min-marginal at the root of $T_i(t)$. If some of the weights are negative, then $b'_i(x_i)$ is computed by maximizing over each variable in $T_i(t)$ whose self-potential has a negative weight and minimizing over each variable whose self-potential has a non-negative weight. In this way, the beliefs correspond to marginals at the root of these computation trees.
Algorithm 3 Cyclic Coordinate Ascent

1: Choose an initial vector $y^0$.
2: for $t = 1, 2, \ldots$ do
3:   Choose a coordinate to update, $i(t)$.
4:   Construct a new vector $y^t$ by setting $y^t_{i(t)} = y^{t-1}_{i(t)}$ and
5:      $y^t_{i(t)} = \arg \max_{x_i} g(y^t_1, \ldots, y^t_{i(t)-1}, y^t_{i(t)+1}, \ldots, y^t_n)$.
6: end for

Algorithm 4 Asynchronous Splitting Algorithm

1: Initialize the messages uniformly to zero.
2: Choose some ordering of the variables, and perform the following update for each variable $j$
3:   for each edge $(j, \beta)$ do
4:      For all $i \in \beta \setminus j$ update the message from $i$ to $\beta$
5:          $m_{i\rightarrow\beta}(x_i) = \kappa + \frac{\varphi_i(x_i)}{c_i} + (c_\beta - 1)m_{\beta\rightarrow i}(x_i) + \sum_{\alpha \in \partial i \setminus \beta} c_{\alpha} m_{\alpha\rightarrow i}(x_i)$
6:   Update the message from $\beta$ to $j$
7:          $m_{\beta\rightarrow j}(x_j) = \kappa + \min_{x_{\beta\setminus j}} \left[ \frac{\psi_\beta(x_\beta)}{c_\beta} + (c_j - 1)m_{j\rightarrow \beta}(x_j) + \sum_{k \in \beta \setminus j} c_k m_{k\rightarrow \beta}(x_k) \right]$}

6.2 Convergence

Given that the lower bounds discussed in Chapters 4 and 5 are concave functions of the messages, we can employ traditional methods from convex optimization in an attempt to maximize them. One such method is cyclic coordinate ascent. This scheme operates by fixing an initial guess for the solution to the optimization problem and then constructs a better solution by performing an optimization over a single variable. Algorithm 3 illustrates the coordinate ascent method when applied to maximize the function $g(y_1, \ldots, y_n)$. When $g$ is strictly concave, Algorithm 3 is guaranteed to converge to the optimal solution. However, when $g$ is concave but not strictly concave, the algorithm may become stuck at local optima (again, a local optimum with respect to the Hamming distance, see Section 5.1). Despite this drawback, we will attempt to maximize our lower bounds on the objective function via block coordinate ascent, a variant of coordinate ascent where the update is performed over larger subsets of the variables at one time.
6.2.1 A Simple Asynchronous Algorithm

Consider the alternative passing schedule in Algorithm 4. This asynchronous message passing schedule fixes an ordering on the variables and for each $j$, in order, updates all of the messages from each $\beta \in \partial j$ to $j$ as if $j$ were the root of the subtree containing only $\beta$ and its neighbors. We will show that, for certain choices of the parameter vector $c$, this message passing schedule improves a specific lower bound of the objective function at each iteration.

To demonstrate convergence of the algorithm, we restrict the parameter vector $c$ so that $c_i = 1$ for all $i$, $c_\alpha > 0$ for all $\alpha$, and $\sum_{\alpha \in \partial i} c_\alpha \leq 1$ for all $i$. Consider the following lower bound on the objective function:

$$
\min_x f(x) = \min_x \left[ \sum_i c_i b_i(x_i) + \sum_\alpha c_\alpha [b_\alpha(x_\alpha) - \sum_{k \in \alpha} c_k b_k(x_k)] \right]
$$

(6.5)

$$
= \min_x \left[ \sum_i c_i (1 - \sum_{\alpha \in \partial i} c_\alpha) b_i(x_i) + \sum_\alpha c_\alpha b_\alpha(x_\alpha) \right]
$$

(6.6)

$$
\geq \sum_i c_i (1 - \sum_{\alpha \in \partial i} c_\alpha) \min_x b_i(x_i) + \sum_\alpha c_\alpha \min_x b_\alpha(x_\alpha)
$$

(6.7)

Define $LB(m)$ to be the lower bound in Equation 6.7. We will show that, with this restricted choice of the parameter vector, Algorithm 4 can be viewed as a block coordinate ascent algorithm on the lower bound $LB(m)$. In order to do so, we need the following lemma:

**Lemma 6.2.1.** Suppose $c_i = 1$ for all $i$, and we perform the update for the edge $(j, \beta)$ as in Algorithm 4. If the vector of messages is real-valued after the update, then $b_\beta$ is min-consistent with respect to $b_j$.

**Proof.** Let $m$ be the vector of messages before the update and let $m^+$ be the vector of messages after the update. The proof of this lemma is similar to that of Lemma 6.1.1. Observe that for each $i \in \beta \setminus j$,

$$
m^+_{i \rightarrow \beta}(x_i) = \kappa + \phi_i(x_i) + (c_\beta - 1) m_{\beta \rightarrow i}(x_i) + \sum_{\alpha \in \partial i \setminus \beta} c_\alpha m_{\alpha \rightarrow i}(x_i)
$$

$$
= \kappa + \phi_i(x_i) + (c_\beta - 1) m^+_{\beta \rightarrow i}(x_i) + \sum_{\alpha \in \partial i \setminus \beta} c_\alpha m^+_{\alpha \rightarrow i}(x_i)
$$

59
Similarly,

\[ m_{\beta \rightarrow j}^+(x_j) = \kappa + \min_{x_{\beta \setminus j}} \left[ \frac{\psi_\beta(x_\beta)}{c_\beta} + \sum_{k \in \beta \setminus j} m_{k \rightarrow \beta}^+(x_k) \right] \]

Up to an additive constant, we have:

\[
\begin{align*}
\min_{x_{\beta \setminus j}} b_{\beta}^+(x_\beta) &= \min_{x_{\beta \setminus j}} \frac{\psi_\beta(x_\beta)}{c_\beta} + \sum_{k \in \beta \setminus j} m_{k \rightarrow \beta}^+(x_k) + \sum_{\alpha \in \partial \beta \setminus \beta} c_\alpha m_{\alpha \rightarrow \beta}^+(x_\beta) \\
&= \min_{x_{\beta \setminus j}} \frac{\psi_\beta(x_\beta)}{c_\beta} + \sum_{k \in \beta \setminus j} m_{k \rightarrow \beta}^+(x_k) + \sum_{\alpha \in \partial \beta \setminus \beta} c_\alpha m_{\alpha \rightarrow \beta}^+(x_\beta) \\
&= m_{\beta \rightarrow j}^+(x_j) + \left[ \phi_j(x_j) + \sum_{\alpha \in \partial j \setminus \beta} c_\alpha m_{\alpha \rightarrow j}^+(x_j) + (c_\beta - 1)m_{\beta \rightarrow j}^+(x_j) \right] \\
&= b_j^+(x_j)
\end{align*}
\]

Observe that, after performing all of the message updates for a node \( j \) as in Algorithm 4, \( b_\beta \) is min-consistent with respect to \( b_j \) for every \( \beta \in A \) containing \( j \). The most important conclusion we can draw from this is that there is an \( x_j^* \) that simultaneously minimizes \( b_j \), \( \min_{x_{\beta \setminus j}} b_{\beta} \), and \( \min_{x_{\beta \setminus j}} b_{\beta} - b_j \). We have the following theorem:

**Theorem 6.2.2.** Suppose \( c_i = 1 \) for all \( i \), \( c_\alpha > 0 \) for all \( \alpha \), and \( \sum c_\alpha \leq 1 \) for all \( i \). If the vector of messages is real-valued at each step of the asynchronous splitting algorithm, then the algorithm converges to a collection of beliefs such that \( LB(m) \) cannot be improved by further iteration of the asynchronous algorithm. Further, if upon convergence of the lower bound the beliefs are locally decodable to \( x^* \), then \( x^* \) is the global minimum of the objective function.

**Proof.** We will show that the message updates performed for variable node \( j \) cannot decrease the lower bound. Let \( LB_j(m) \) denote the terms in \( LB \) that involve the variable \( x_j \). We can upper
bound \( LB_j \) as follows:

\[
LB_j(m) \leq \min_{x_j} \left( (1 - \sum_{\beta \in \partial_j} c_\beta b_j(x_j) + \sum_{\beta \in \partial_j} c_\beta \min_{x_{x\setminus j}} b_\beta(x_\beta)) \right)
\]

\[
= \min_{x_j} \left( (1 - \sum_{\beta \in \partial_j} c_\beta b_j(x_j) + \sum_{\beta \in \partial_j} c_\beta \min_{x_{x\setminus j}} \frac{\psi_\beta(x_\beta)}{c_\beta} + \sum_{k \in \beta} (b_k(x_k) - m_{\beta \rightarrow k}(x_k))) \right)
\]

\[
= \min_{x_j} \left[ \phi_j(x_j) + \sum_{\beta \in \partial_j} c_\beta \min_{x_{x\setminus j}} \frac{\psi_\beta(x_\beta)}{c_\beta} + \sum_{k \in \beta \setminus j} (b_k(x_k) - m_{\beta \rightarrow k}(x_k)) \right]
\]

Notice that this upper bound does not depend on the choice of the messages from \( \beta \) to \( j \) for any \( \beta \in \partial_j \). As a result, any choice of these messages for which the inequality is tight must maximize \( LB_j \). Observe that the upper bound is tight if there exists an \( x_j \) that simultaneously minimizes \( b_j \) and \( \min_{x\setminus j} b_\beta \) for each \( \beta \in \partial_j \). By Lemma 6.2.1, this is indeed the case after performing the updates in Algorithm 4 for the variable node \( j \). As this is the only part of the lower bound affected by the update, we have that \( LB \) cannot decrease. Let \( m \) be the vector of messages before the update for variable \( j \) and \( m^+ \) the vector after the update. Define \( LB_{-j} \) to be the sum of the terms of the lower bound that do not involve the variable \( x_j \). By definition and the above we have:

\[
LB(m) = LB_{-j}(m) + LB_j(m)
\]

\[
\leq LB_{-j}(m) + \min_{x_j} \left( (1 - \sum_{\beta \in \partial_j} c_\beta b_j(x_j) + \sum_{\beta \in \partial_j} c_\beta \min_{x_{x\setminus j}} b_\beta(x_\beta)) \right)
\]

\[
= LB_{-j}(m^+) + \min_{x_j} \left( (1 - \sum_{\beta \in \partial_j} c_\beta b_j^+(x_j) + \sum_{\beta \in \partial_j} c_\beta \min_{x_{x\setminus j}} b_\beta^+(x_\beta)) \right)
\]

\[
= LB(m^+)
\]

\( LB(m) \) is bounded from above by \( \min_x f(x) \). From this we can conclude that the value of the lower bound converges.

Finally, the lower bound has converged if no single variable update can improve the bound. By the arguments above, this must mean that there exists an \( x_{\star} \) that simultaneously minimizes \( b_j \) and \( \min_{x\setminus j} b_\beta \) for each \( \beta \in \partial_j \). These beliefs may or may not be min-consistent. Now, if there exists a unique minimizer \( x_{\star} \), then \( x_{\star} \) must simultaneously minimize \( b_j \) and \( \min_{x\setminus j} b_\beta \) for each \( \beta \in \partial_j \). From this we can conclude that \( x_{\star} \) simultaneously minimizes all of the beliefs and therefore, using the argument from Theorem 5.2.3, must minimize the objective function. \( \Box \)

Although the restriction on the parameter vector in Theorem 6.2.2 seems strong, we observe
Algorithm 5 Damped Synchronous Splitting Algorithm

1: Fix a finite vector of initial messages, $m^0$.
2: Choose $\delta \in [0, 1]$.
3: for $t = 1, 2, \ldots$ do
4: For each $\alpha$ and $i \in \alpha$, update the message from $i$ to $\alpha$
   $$m^t_{i \rightarrow \alpha}(x_i) = \kappa + \frac{\phi_i(x_i)}{c_i} + (c_i - 1)m^{t-1}_{\alpha \rightarrow i}(x_i) + \sum_{\beta \in \partial_i \setminus \alpha} c_\beta m^{t-1}_{\alpha \rightarrow i}(x_i)$$
5: For each $\alpha$ and $i \in \alpha$, update the message from $\alpha$ to $i$
   $$m^t_{\alpha \rightarrow i}(x_i) = \kappa + (1 - \delta)m^{t-1}_{\alpha \rightarrow i}(x_i) + \delta \min_{x_\alpha \setminus i} \left[ \frac{\psi_\alpha(x_\alpha)}{c_\alpha} + (c_i - 1)m^{t-1}_{\alpha \rightarrow i}(x_i) + \sum_{k \in \alpha \setminus i} c_k m^{t}_{k \rightarrow \alpha}(x_k) \right]$$
6: end for

that for any objective function $f$, we can choose the parameters such that the theorem is sufficient to guarantee convergence and global optimality:

Corollary 6.2.3. Define $c_\alpha = \frac{1}{\max_i |\partial_i|}$ for all $\alpha$ and $c_i = 1$ for all $i$. If the vector of messages is real-valued at each step of Algorithm 4, then the algorithm converges. Further, if the converged beliefs are locally decodable to $x^*$, then $x^*$ is a global minimum of $f$.

We present Algorithm 4 both as a convergent local message passing algorithm and as an example of how the intuition developed from the optimality conditions can be extended to show convergence results: we can achieve global consistency by carefully ensuring a weak form of local consistency. Recall that, like other coordinate ascent schemes, this algorithm can become stuck (i.e. reach a fixed point) that does not maximize the lower bound. For an example of an objective function and choice of parameters such that the algorithm may become stuck, see [21].

### 6.2.2 Synchronous Convergence

By using the asynchronous schedule in Algorithm 4, we seem to lose the distributed nature of the parallel message updates. For some asynchronous schedules, we can actually parallelize the updating process by performing concurrent updates as long as the simultaneous updates do not form a cycle (e.g. we could randomly select a subset of the message updates that do not interfere). We also note that performing updates over larger subgraphs may be advantageous. Other algorithms, such as those in [21, 47], perform updates over larger subtrees of the factor graph.

Asynchronous message passing algorithms can be converted into distributed algorithms by performing multiple asynchronous updates in parallel and then averaging the resulting message vectors.
Unfortunately, this process may require some amount of central control at each step and typically results in slower rates of convergence when compared with the original asynchronous message passing scheme. [47] For example, consider a message vector \( m \). Let \( m' \) be the vector of messages produced by performing the asynchronous update for the variable node \( i \) on the message vector \( m \). For an appropriate choice of the vector \( c \), Theorem 6.2.2 guarantees that \( LB(m') \geq LB(m) \) for all \( i \). Since the lower bound is concave, we must also have that \( LB(\sum_i \frac{m_i}{n}) \geq LB(m) \) where \( n \) is the total number of variable nodes. Let \( m' = \sum_i \frac{m_i}{n} \). We have the following for each \( \alpha \) and \( i \in \alpha \):

\[
m'_{\alpha \rightarrow i}(x_i) = \frac{n-1}{n} m_{\alpha \rightarrow i}(x_i) + \frac{1}{n} m'_{\alpha \rightarrow i}(x_i)
\] (6.8)

This observation can be used to construct the synchronous algorithm described in Algorithm 5. Notice that the factor to variable update in Algorithm 5 is equivalent to the damped message updates from Section 3.4. Algorithm 5 is guaranteed to converge when the parameter vector satisfies the conditions of Theorem 6.2.2 and \( \delta = \frac{1}{n} \). Other choices of \( \delta \) can also result in convergence. For example, instead of constructing \( n \) new message vectors and averaging, we may be able to use only \( k \) such vectors by performing multiple disjoint asynchronous updates to generate the \( k \) new message vectors and then averaging. This produces a synchronous algorithm in the form of Algorithm 5 whenever the asynchronous update is performed exactly once for each variable \( i \) when generating the \( k \) vectors.

### 6.2.3 Subgradient Methods

The fixed points of the splitting algorithm do not necessarily correspond to maxima of the lower bound. As discussed earlier, this problem can occur when using coordinate ascent schemes to optimize concave, but not strictly concave, lower bounds. Other optimization strategies do not suffer from this problem but may have slower rates of convergence. In this section, we discuss one alternative strategy known as subgradient ascent.

The subgradient ascent method is a generalization of the gradient ascent method to functions that are not necessarily differentiable. Let \( g : \mathbb{R}^n \rightarrow \mathbb{R} \), and fix a \( y^0 \in \mathbb{R}^n \). \( h \in \mathbb{R}^n \) is a subgradient of \( g \) at the point \( y^0 \) if for all \( y \in \mathbb{R}^n \), \( g(y) - g(y^0) \geq h \cdot (y - y^0) \). If \( g \) is differentiable, then \( \nabla g(y^0) \) is the only subgradient of \( g \) at \( y^0 \).

The subgradient method to minimize the function \( g \) performs the iteration in Algorithm 6.
Algorithm 6 Subgradient Ascent

1: Let \( g : \mathbb{R}^n \to \mathbb{R} \).
2: Choose an initial vector \( y^0 \).
3: for \( t = 1, 2, \ldots \) do
4: Construct a new vector \( y^t \) by setting
\[
y^t = y^{t-1} + \gamma_t h^t
\]
where \( h^t \) is a subgradient of \( g \) at \( y^{t-1} \) and \( \gamma_t \) is the step size at time \( t \).
5: end for

Unlike gradients, subgradients do not necessarily correspond to ascent directions. However, under certain conditions on the sequence \( \gamma_1, \ldots, \gamma_t \), the subgradient algorithm can be shown to converge [46].

The subgradient algorithm can be converted into a distributed algorithm by exploiting the fact that the subgradient of a sum of functions is equal to the sum of the individual subgradients. Such a strategy has been used to design master/slave algorithms for maximizing the concave lower bounds above [24]. This procedure requires a certain amount of central control, that may not be possible in certain applications. A double loop method that is equivalent to the subgradient method was proposed in [36].

### 6.3 An Alternative Derivation

In this section, we provide an alternative derivation of Algorithm 2 in the case that the \( c_i \) and \( c_\alpha \) correspond to positive integers. This derivation relies upon splitting the factor and variable nodes in the factor graph in order to construct a new message passing scheme.

Suppose \( f \) factorizes over \( \mathcal{A} \) as in equation 2.5. Let \( G \) be the corresponding factor graph. Suppose now that we take one potential \( \alpha \in \mathcal{A} \) and split it into \( k \) potentials \( \alpha_1, \ldots, \alpha_k \) such that for each \( j \in \{1, \ldots, k\} \), \( \psi_{\alpha_j} = \frac{\psi_{\alpha}}{k} \). This allows us to rewrite the objective function, \( f \), as

\[
f(x) = \sum_i \phi_i(x_i) + \sum_{\beta \in \mathcal{A}} \psi_\beta(x_\beta) \quad \text{(6.9)}
\]

\[
= \sum_i \phi_i(x_i) + \sum_{\beta \in \mathcal{A} \setminus \alpha} \psi_\beta(x_\beta) + \sum_{j=1}^k \frac{\psi_{\alpha_j}(x_\alpha)}{k} \quad \text{(6.10)}
\]

\[
= \sum_i \phi_i(x_i) + \sum_{\beta \in \mathcal{A} \setminus \alpha} \psi_\beta(x_\beta) + \sum_{j=1}^k \psi_{\alpha_j}(x_\alpha) \quad \text{(6.11)}
\]
This rewriting does not change the objective function, but it does produce a new factor graph $F$ (see Figure 6.2). Now, take some $i \in \alpha$ and consider the messages $m_{i \rightarrow \alpha}$ and $m_{\alpha \rightarrow i}$ given by the standard min-sum algorithm:

$$m_{i \rightarrow \alpha}^t(x_i) = \kappa + \phi_i(x_i) + \sum_{\beta \in \partial_F \setminus \alpha} m_{\beta \rightarrow i}^{t-1}(x_i)$$

$$m_{\alpha \rightarrow i}^t(x_i) = \kappa + \min_{x_{\alpha \setminus i}} \frac{\psi^\alpha(x_{\alpha})}{k} + \sum_{k \in \alpha \setminus i} m_{k \rightarrow \alpha}^{t-1}(x_k)$$

where $\partial_F i$ denotes the neighbors of $i$ in $F$. Notice that there is an automorphism of the graph that maps $\alpha_l$ to $\alpha_j$. As the messages passed from any node only depend on the messages received at the previous time step, if the initial messages are the same at both of these nodes, then they must produce identical messages at time 1. More formally, if we initialize the messages identically over each split edge, then, at any time step $t \geq 0$, $m_{i \rightarrow \alpha}^t(x_i) = m_{i \rightarrow \alpha}^t(x_i)$ and $m_{\alpha \rightarrow i}^t(x_i) = m_{\alpha \rightarrow i}^t(x_i)$ for any $l \in \{1, ..., k\}$ by symmetry (i.e. there is an automorphism of the graph that maps $\alpha_l$ to $\alpha_j$).

Because of this, we can rewrite the message from $i$ to $\alpha_j$ as:

$$m_{i \rightarrow \alpha_j}^t(x_i) = \kappa + \phi_i(x_i) + \sum_{\beta \in \partial_F \setminus \alpha} m_{\beta \rightarrow i}^{t-1}(x_i)$$

$$= \kappa + \phi_i(x_i) + \sum_{l \neq j} m_{\alpha_l \rightarrow i}^{t-1}(x_i) + \sum_{\beta \in \partial_F \setminus \alpha} m_{\beta \rightarrow i}^{t-1}(x_i)$$

$$= \kappa + \phi_i(x_i) + (k-1)m_{\alpha_j \rightarrow i}^{t-1}(x_i) + \sum_{\beta \in \partial_F \setminus \alpha} m_{\beta \rightarrow i}^{t-1}(x_i)$$

Notice that equation 6.16 can be viewed as a message passing algorithm on the original factor graph. The primary difference then between equation 6.16 and the standard min-sum updates is that the message passed from $i$ to $\alpha$ now depends on the message from $\alpha$ to $i$.

Analogously, we can also split the variable nodes. Suppose $f$ factorizes over $\mathcal{A}$ as in equation
2.5. Let $G$ be the corresponding factor graph. Suppose now that we take one variable $x_i$ and split it into $k$ variables $x_{i1},...,x_{ik}$ such that for each $l \in \{1,...,k\}$, $\phi_{il} = \frac{\phi_i(x_{i})}{k}$. This produces a new factor graph, $F$. Because $x_{i1},...,x_{ik}$ are all the same variable, we must add a constraint to ensure that they are indeed the same. Next, we need to modify the potentials to incorporate the constraint and the change of variables. We will construct $A_F$ such that for each $\alpha \in A$ with $i \notin \alpha$ there is a $\beta = (\alpha \setminus i) \cup \{i_1,...,i_k\}$ in $A_F$. Define $\psi_{\beta}(x_{\beta}) = \psi_{\alpha}(x_{\alpha \setminus i},x_{i_1}) - \log\{x_{i_1} = ... = x_{i_k}\}$ where $\{x_{i_1} = ... = x_{i_k}\}$ is the 0-1 indicator function for the equality constraint. For each $\alpha \in A$ with $i \notin \alpha$ we simply add $\alpha$ to $A_F$ with its old potential. For an example of this construction, see Figure 6.3.

This rewriting produces a new objective function

$$g(x) = \sum_{j \neq i} \phi_j(x_j) + \frac{1}{k} \sum_{l=1}^{k} \phi_{il}(x_{il}) + \sum_{\alpha \in A_F} \psi_{\alpha}(x_{\alpha})$$  \hspace{1cm} (6.17)

Minimizing $g$ is equivalent to minimizing $f$. Again, we will show that we can collapse the min-sum message passing updates over $F$ to message passing updates over $G$ with modified potentials. Take some $\alpha \in A_F$ containing the new variable $i_1$ which augments the potential $\gamma \in A$ and consider the messages $m_{i_1 \rightarrow \beta}$ and $m_{\beta \rightarrow i_1}$ given by the standard min-sum algorithm:

$$m_{i_1 \rightarrow \alpha}^{t}(x_{i_1}) = \kappa + \frac{\phi_i(x_{i})}{k} + \sum_{\beta \in \partial_F \setminus \alpha} m_{\beta \rightarrow i_1}^{t-1}(x_{i_1})$$  \hspace{1cm} (6.18)

$$m_{\alpha \rightarrow i_1}^{t}(x_{i_1}) = \kappa + \min_{x_{\alpha \setminus i_1}} \psi_{\alpha}(x_{\alpha}) + \sum_{k \in \alpha \setminus i_1} m_{k \rightarrow \alpha}^{t-1}(x_k)$$  \hspace{1cm} (6.19)

Again, if we initialize the messages identically over each split edge, then, at any time step $t \geq 0$, $m_{i_1 \rightarrow \alpha}^{t}(x_{i}) = m_{i_1 \rightarrow \alpha}^{t}(x_{i_1})$ and $m_{\alpha \rightarrow i_1}^{t}(x_{i}) = m_{\alpha \rightarrow i_1}^{t}(x_{i_1})$ for any $l \in \{1,...,k\}$ by symmetry. Using
this, we can rewrite the message from \( \alpha \) to \( i_1 \) as:

\[
m_{\alpha \rightarrow i_1}(x_{i_1}) = \kappa + \min_{x_{\alpha \setminus i_1}} \psi_\alpha(x_\alpha) + \sum_{k \in \alpha \setminus i_1} m_{k \rightarrow \alpha}^{t-1}(x_k) \tag{6.20}
\]

\[
= \kappa + \min_{x_{\alpha \setminus i_1}} \psi_\alpha(x_\alpha, x_{i_1}) - \log\{x_{i_1} = \ldots = x_{i_k}\} + \sum_{k \in \alpha \setminus i_1} m_{k \rightarrow \alpha}^{t-1}(x_k) \tag{6.21}
\]

\[
= \kappa + \min_{x_{\alpha \setminus i_1}} \psi_\alpha(x_\alpha, x_{i_1}) - \log\{x_{i_1} = \ldots = x_{i_k}\} + \sum_{l \neq 1} m_{i_1 \rightarrow \alpha}^{t-1}(x_{i_l}) + \sum_{k \in \gamma \setminus i} m_{k \rightarrow \alpha}^{t-1}(x_k) \tag{6.22}
\]

\[
= \kappa + \min_{x_{\alpha \setminus i_1}} \psi_\alpha(x_\alpha, x_{i_1}) + \sum_{l \neq 1} m_{i_1 \rightarrow \alpha}^{t-1}(x_{i_l}) + \sum_{k \in \gamma \setminus i} m_{k \rightarrow \alpha}^{t-1}(x_k) \tag{6.23}
\]

\[
= \kappa + \min_{x_{\alpha \setminus i_1}} \psi_\alpha(x_\alpha, x_{i_1}) + (k-1) m_{i_1 \rightarrow \alpha}^{t-1}(x_{i_1}) + \sum_{k \in \gamma \setminus i} m_{k \rightarrow \alpha}^{t-1}(x_k) \tag{6.24}
\]

By symmetry, we only need to perform one message update to compute \( m_{\alpha \rightarrow i_l}(x_{i_l}) \) for each \( l \in \{1, \ldots, k\} \). As a result, we can think of these messages as being passed on the original factor graph \( G \). The combined message updates for each of these splitting operations are exactly those described in Algorithm 2.

Throughout this discussion, we have assumed that each factor was split into \( k \) pieces where \( k \) was some positive integer. If we allow \( k \) to be an arbitrary non-zero real, then the notion of splitting no longer makes sense. Instead, as described in Section 4.2, these splittings can be viewed more generally as a choice of alternate reparameterizations.
Chapter 7

Graph Covers and Indistinguishability

Thus far, we have demonstrated that certain parameterized variants of the min-sum algorithm allow us to guarantee both convergence and correctness, upon convergence to locally decodable beliefs. If the beliefs are not locally decodable, then the resulting beliefs produce a lower bound on the objective function but seem to tell us very little about the argmin of the objective function.

We have provided relatively little intuition about when we can expect the converged beliefs to be locally decodable. As we saw in Chapter 3, the success of the min-sum algorithm is intrinsically tied to both the uniqueness of the optimal solution and the "hardness" of the optimization problem. For lower bounds such as those in Section 4.3.1, we provide necessary conditions for dual optima to be locally decodable.

These necessary conditions rely on a notion of indistinguishability: the splitting algorithm, in attempting to solve the minimization problem on one factor graph, is actually attempting to solve the minimization problem over an entire family of equivalent (in some sense) factor graphs. The same notion of indistinguishability has been studied for general distributed message passing schemes [4, 5], and we expect ideas similar to those discussed in this and subsequent chapters to be applicable to other iterative algorithms as well (e.g. see Section 10.1.1).

Section 7.1 introduces graph covers and discusses their basic properties. Section 7.2 discusses graph covers in the special case that $f$ satisfies Assumption 2. In Section 7.3, we discuss pseudocodewords and their implications for message passing algorithms. In Section 7.4, we show that,
for functions satisfying Assumption 2, assignments on graph covers correspond to rational feasible points of the MAP LP. As a consequence of this discussion, we show how the MAP LP and graph covers can be used to provide necessary conditions for the splitting algorithm to converge to admissible, min-consistent, and locally decodable beliefs.

7.1 Graph Covers

The above notion of indistinguishability is captured by the formalism of graph covers. Intuitively, if a graph $H$ covers the graph $G$, then $H$ has the same local structure as $G$. Our local message passing schemes depend only on local graph structure and local potentials. The greatest strength of the splitting algorithm, its reliance on only local information, can also be a weakness. Consider the following definition:

Definition 7.1.1. A graph $H$ covers a graph $G$ if there exists a graph homomorphism $h : H \rightarrow G$ such that $h$ is an isomorphism on neighborhoods (i.e. for all vertices $i \in H$, $\partial i$ is mapped bijectively onto $\partial h(i)$). If $h(i) = j$, then we say that $i \in H$ is a copy of $j \in G$. Further, $H$ is a $k$-cover of $G$ if every vertex of $G$ has exactly $k$ copies in $H$.

Graph covers may be connected (i.e. there is a path between every pair of vertices) or disconnected. However, when a graph cover is disconnected, all of the connected components of the cover must themselves be covers of the original graph. For a simple example of a connected graph cover, see Figure 7.1.

Every finite cover is a $k$-cover for some integer $k$. For every base graph $G$, there exists a graph, possibly infinite, which covers all finite, connected covers of the base graph. This graph is known as the universal cover. Throughout this work, we will be primarily concerned with finite, connected covers.

To any finite cover, $H$, of a factor graph $G$ we can associate a collection of potentials derived...
from the base graph; the potential at node \( i \in H \) is equal to the potential at node \( h(i) \in G \). Together, these potential functions define a new objective function for the factor graph \( H \). In the sequel, we will use superscripts to specify that a particular object is over the factor graph \( H \). For example, we will denote the objective function corresponding to a factor graph \( H \) as \( f^H \), and we will write \( f^G \) for the objective function \( f \).

Graph covers, in the context of graphical models, were originally studied in relation to local message passing algorithms [49]. Local message passing algorithms such as the min-sum and splitting algorithms are incapable of distinguishing the two factor graphs \( H \) and \( G \) given that the initial messages to and from each node in \( H \) are identical to the nodes they cover in \( G \): for every node \( i \in G \) the messages received and sent by this node at time \( t \) are exactly the same as the messages sent and received at time \( t \) by any copy of \( i \) in \( H \). As a result, if we use a local message passing algorithm to deduce an assignment for \( i \), then the algorithm run on the graph \( H \) must deduce the same assignment for each copy of \( i \).

Now, consider an objective function \( f \) that factors with respect to \( A \). Let \( G \) denote the corresponding factor graph. For any finite cover \( H \) of \( G \) with covering homomorphism \( h : H \to G \), we can "lift" any vector of beliefs, \( b^G \), from \( G \) to \( H \) by defining a new vector of beliefs, \( b^H \), such that:

- For all variable nodes \( i \in H \), \( b^H_i = b^G_{h(i)} \)
- For all factor nodes \( \alpha \in H \), \( b^H_\alpha = b^G_{h(\alpha)} \)

Analogously, we can lift any assignment \( x^G \) to an assignment \( x^H \) by setting \( x^H_i = x^G_{h(i)} \).

### 7.2 Factorizations and Graph Covers

As we saw in Section 4.4 for objective functions satisfying Assumption 2, any two factorizations of the objective function, \( f^G \), over \( A \) can be related via message reparameterizations. Surprisingly, this correspondence continues to hold for any finite cover \( H \) of \( G \). Let \( f(x) = \sum_i \phi_i(x_i) + \sum_\alpha \psi_\alpha(x_\alpha) = \sum_i \phi'_i(x_i) + \sum_\alpha \psi'_\alpha(x_\alpha) \). Each of these factorizations produces an objective function with respect to the factor graph \( H \) by lifting the potentials from \( G \). Denote these two potentially different objective functions as \( f^H \) and \( g^H \). We can check that \( f^H(x^H) = g^H(x^H) \) for all \( x^H \). This is a direct consequence of Theorem 4.4.1:

**Theorem 7.2.1.** Suppose \( f^G \) satisfies Assumption 2 and \( f^G \) factors over \( A \) in two ways as \( f^G(x) = \)
\[ \sum_i \phi_i(x_i) + \sum_\alpha \psi_\alpha(x_\alpha) = \sum_i \phi'_i(x_i) + \sum_\alpha \psi'_\alpha(x_\alpha). \] The objective functions corresponding to the lift of these potentials from the base graph \( G \) to any finite cover \( H \) of \( G \) agree on every \( x \).

**Proof.** By Theorem 4.4.1, there exists a vector of messages, \( m \), such that for all \( i \), \( \phi_i(x_i) + \sum_{\alpha \in \partial i} m_{\alpha \rightarrow i}(x_i) = \phi'_i(x_i) + \kappa_i \) and for all \( \alpha \phi_\alpha(x_\alpha) - \sum_i \in \partial_\alpha m_{\alpha \rightarrow i}(x_\alpha) = \phi'_\alpha(x_\alpha) + \kappa_\alpha \) for some vector of constants \( \kappa \) such that \( \sum_i \kappa_i + \sum_\alpha \kappa_\alpha = 0 \). Let \( H \) be a finite cover of \( G \) with covering map \( h : H \rightarrow G \), let \( f_H \) be the lift of the primed potentials to \( H \), and let \( g_H \) be the lift of the unprimed potentials to \( H \):

\[
\begin{align*}
    f^H(x^H) &= \sum_{i \in H} \phi_{h(i)}(x^H_i) + \sum_{\alpha \in H} \psi_{h(\alpha)}(x^H_\alpha) \\
    &= \sum_{i \in H} \left[ \kappa_i + \phi_{h(i)}(x^H_i) \right] + \sum_{\alpha \in H} \left[ \kappa_\alpha + \psi_{h(\alpha)}(x^H_\alpha) \right] \\
    &= \sum_{i \in H} \left[ \phi_{h(i)}(x^H_i) + \sum_{\alpha \in \partial i} m_{h(\alpha) \rightarrow h(i)}(x^H_i) \right] + \sum_{\alpha \in H} \left[ \psi_{h(\alpha)}(x^H_\alpha) - \sum_{i \in \alpha} m_{h(\alpha) \rightarrow h(i)}(x^H_i) \right] \\
    &= \sum_{i \in H} \phi_{h(i)}(x^H_i) + \sum_{\alpha \in H} \psi_{h(\alpha)}(x^H_\alpha) \\
    &= g^H(x^H)
\end{align*}
\]

As a consequence of this theorem and Assumption 2, if \( f \) factors over \( \mathcal{A} \) with corresponding factor graph \( G \), then for all finite graph covers \( H \) of \( G \), \( f^H \) is uniquely determined, independent of the chosen factorization. Therefore, for any factorization over \( \mathcal{A} \), we can unambiguously write \( f^H \) to mean the objective function corresponding to the graph cover \( H \) of \( G \). That is, the objective function over finite covers does not depend on the specific factorization corresponding to the base graph.

### 7.3 Pseudocodewords

Surprisingly, minima of the objective function \( f^H \) need not be lifts of the minima of the objective function \( f^G \). Even worse, the minimum value of \( f^G \) does not necessarily correspond to the minimum value of \( f^H \). This idea is the basis for the theory of pseudocodewords in the LDPC (low-density parity-check) community [49, 50]. In this community, assignments on covers that are not lifts of assignments of the base problem are referred to as pseudocodewords.
The existence of pseudocodewords is not unique to coding theory. Consider the maximum weight independent set problem in Figure 7.2. The maximum weight independent set for the graph in Figure 7.2 (a) has weight three. The maximum weight independent set on the 2-cover of this graph in Figure 7.2 (b) has weight seven, which is larger than the lift of the maximum weight independent set from the base graph.

Because local message passing algorithms cannot distinguish a factor graph from its covers and our lower bounds provide lower bounds on the objective function corresponding to any finite cover, we expect that maxima of the lower bounds, at best, correspond to the optimal solution on some graph cover of the original problem. For the splitting algorithm, we can make this intuition precise:

**Theorem 7.3.1.** Let $b$ be a vector of admissible and min-consistent beliefs for the function $f$ with a corresponding weighting vector of non-zero real numbers, $c$, such that $f$ can be written as a conical combination of the beliefs. If $x^G$ simultaneously minimizes the beliefs, then $x^G$ minimizes $f^G$, and for any finite cover $H$ of $G$ with covering homomorphism $h$, $x^H$, the lift of $x^G$ to $H$, minimizes $f^H$. Further, if $y^H$ minimizes $f^H$, then $y^H_i \in \arg \min_{x_i} b_{h(i)}(x_i)$ for all $i$.

**Proof.** The beliefs can be lifted from $G$ to $H$. In other words, the beliefs define a reparameterization of the objective function $f^H$. As $f^G$ can be written as a conical combination of the beliefs, $f^H$ can also be written as a conical combination of the beliefs. The result then follows from Theorem 5.2.3. Because $x^H$ simultaneously minimizes $f^H$ and each of the beliefs, any other minimum of $f^H$ must also satisfy this property. 

As a consequence of Theorem 7.3.1, for any choice of the parameter vector that guarantees correctness, the splitting algorithm can only converge to a locally decodable vector of admissible and min-consistent beliefs if the objective function corresponding to any finite graph cover has a unique optimal solution that is a lift of the unique optimal solution on the base graph.

---

Figure 7.2: An example of a maximum weight independent set problem and a graph cover whose maximum weight independent set is not a copy of an independent set on the original graph. The nodes are labeled with their corresponding weights.
These results highlight the inherent weaknesses of conical decompositions and, as we will see in Section 7.4, the dual approach in general: conical decompositions guarantee the correctness, on every finite cover of the base graph, of any assignment that simultaneously minimizes each of the beliefs. For many applications this guarantee proves to be too strong. This suggests that the convergent and correct strategy outlined in the previous chapters is not as useful as it seems, and for many applications, the standard min-sum algorithm, which only guarantees a form of local optimality on all graph covers, may still be preferred in practice. For an example of such a problem, see [60].

7.4 Graph Covers and the MAP LP

The existence of pseudocodewords is problematic because local message passing algorithms cannot distinguish a graph from its covers. For objective functions over finite state spaces satisfying Assumption 2, we can relate the previous observations to the MAP LP. There is a one-to-one correspondence between the rational feasible points of the MAP LP and assignments on graph covers: every rational feasible point of the MAP LP corresponds to an assignment on some cover, and every assignment on a cover of the base graph corresponds to a rational feasible point of the MAP LP. This relationship was first observed in the coding community [11], but we show how to extend these results to the general case.

Lemma 7.4.1. For objective functions satisfying Assumption 2, there exists a rational feasible point, \( \mu \), of the MAP LP for \( f^G \) such that
\[
\sum_{i \in G} \sum_{x_i} \mu_i(x_i) \phi_i(x_i) + \sum_{\alpha \in G} \sum_{x_\alpha} \mu_\alpha(x_\alpha) \psi_\alpha(x_\alpha) = c
\]
if and only if there exists a \( k \)-cover \( H \) of \( G \) and an assignment \( x^H \) such that
\[
f^H(x^H) = k \cdot c.
\]

Proof. First, suppose \( \mu \) is a rational feasible point of the MAP LP for \( f^G \) that attains the value \( c \). Let \( k \) be the smallest integer such that \( k \cdot \mu_i(x_i) \in \mathbb{Z} \) for all \( i \) and \( x_i \) and \( k \cdot \mu_\alpha(x_\alpha) \in \mathbb{Z} \) for all \( \alpha \) and \( x_\alpha \). Now, we construct a \( k \)-cover \( H \) of \( G \) by creating \( k \) copies of each variable and factor node of \( G \). We will think of each copied variable as corresponding to a particular assignment. For example, consider the \( k \) copies in \( H \) of the variable node \( i \in G \). Exactly \( k \cdot \mu_i(x_i) \) of these copies will be assigned the value \( x_i \). Similarly, each of the \( k \) copies in \( H \) of the factor node \( \alpha \in G \) will correspond to a particular assignment: \( k \cdot \mu_\alpha(x_\alpha) \) of these copies will be assigned the value \( x_\alpha \).

We can now add edges to \( H \) in the following way: for each copy \( \alpha' \in H \) of the factor node \( \alpha \in G \), we look at its corresponding assignment and connect \( \alpha' \) to a subset of the variables in \( H \) that are
copies of some $i \in \alpha$, not already connected to a copy of the node $\alpha$ in $H$, and whose corresponding assignments are consistent with the assignment for $\alpha'$. Note that there is not necessarily a unique way to do this, but after this process, every copy in $H$ of $i \in \alpha$ will be connected to a copy of $\alpha$ in $H$. After repeating this for the remaining factors in $G$, we must have that $H$ is a $k$-cover of $G$, and the assignment $x^H$, given by the chosen assignment corresponding to each variable node in $H$, must satisfy $f^H(x^H) = k \left[ \sum_{i} \sum_{x_i} \mu_i(x_i) \phi_i(x_i) + \sum_{\alpha} \sum_{x_{\alpha}} \mu_\alpha(x_{\alpha}) \psi_\alpha(x_{\alpha}) \right]$.

For the other direction, let $H$ be a $k$-cover of $G$ with cover homomorphism $h : H \rightarrow G$, and let $x^H$ be an assignment to the variables in $H$. Define $\mu_i(x_i) = \frac{1}{k} \sum_{k \in H, h(k) = i} \{ x^H_k = x_i \}$. Observe that $k \cdot \mu_i(x_i)$ is then equal to the number of times in the assignment $x^H$ that some copy in $H$ of the variable $i \in G$ is assigned the value $x_i$. Similarly, define $\mu_\alpha(x_\alpha) = \frac{1}{k} \sum_{\beta \in H, h(\beta) = \alpha} \{ x^H_\beta = x_\alpha \}$. With these definitions, $\mu_\alpha(x_\alpha)$ is sum consistent (min-consistency with the mins replaced by sums) with respect to $\mu_i(x_i)$ for each $i \in \alpha$. This means that $\mu$ is feasible for the MAP LP. Finally, we have the following:

$$f^H(x^H) = \sum_{i \in H} \phi_{h(i)}(x^H_i) + \sum_{\alpha \in H} \psi_{h(\alpha)}(x^H_\alpha)$$

$$= \sum_{i \in G} \sum_{x_i} k \cdot \mu_i(x_i) \phi_i(x_i) + \sum_{\alpha \in G} \sum_{x_\alpha} k \cdot \mu_\alpha(x_\alpha) \psi_\alpha(x_\alpha)$$

$$= k \left[ \sum_{i \in G} \sum_{x_i} \mu_i(x_i) \phi_i(x_i) + \sum_{\alpha \in G} \sum_{x_\alpha} \mu_\alpha(x_\alpha) \psi_\alpha(x_\alpha) \right]$$

Because the polyhedron corresponding to the MAP LP is rational, the optimum of the MAP LP is attained at a rational point. Hence, there is a $k$-cover $H$ of $G$ and an assignment $x^H$ that, in the sense of Lemma 7.4.1, achieves the optimal value of the MAP LP, and this assignment corresponds to a lower bound on the minimum value of any finite cover of $G$ (i.e. for every $k'$-cover $H'$ of $G$ and any assignment $x^{H'}$ on $H'$, $f^{H'}(x^{H'}) \leq \frac{f^H(x^H)}{k}$).

We also note that Lemma 7.4.1 shows that the polytope corresponding to the MAP LP is equivalent to the fundamental polytope defined in [49].

### 7.4.1 Maxima of the Lower Bound

We now address the question of when we can construct a solution to the optimization problem from our lower bounds. As in Section 4.3.2, we will be concerned with the following optimization
problem for objective functions satisfying Assumption 2:

$$\sup_{(\phi, \psi) \in \mathcal{S}} \left[ \sum_i \min_{x_i} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \min_{x_\alpha} \psi_\alpha(x_\alpha) \right]$$  \hspace{1cm} (7.9)$$

We will assume that \( \mathcal{S} \) is chosen such that the above optimization problem is dual to the MAP LP for the factorization \((\phi', \psi')\) with corresponding factor graph \(G\) and that strong duality holds. If this is the case, then the function \( LB \) defined by

$$LB(\phi, \psi) = \sum_i \min_{x_i} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \min_{x_\alpha} \psi_\alpha(x_\alpha)$$  \hspace{1cm} (7.10)$$

is a lower bound on \( f(x) \) for all \((\phi, \psi) \in \mathcal{S}\). Further, observe that for any \((\phi, \psi) \in \mathcal{S}\), if \( x^* \) simultaneously minimizes (i.e. \( x^*_i \in \arg\min \phi_i \) for each \( i \) and \( x^*_\alpha \in \arg\min \psi_\alpha \) for each \( \alpha \)) each term of the lower bound then \( LB(\phi, \psi) = f(x^*) \). Using strong duality and Lemma 7.4.1, we can characterize exactly when such an \( x^* \) exists:

**Theorem 7.4.2.** Let \( f^G \) be an objective function satisfying Assumption 2 with corresponding factor graph \( G \). If \((\phi, \psi) \) maximizes \( LB \), then the following are equivalent:

1. The MAP LP for the factorization \((\phi, \psi)\) has an integral optimum.

2. There is an assignment \( x^* \) that simultaneously minimizes each term in the lower bound.

3. If \( x^G \) minimizes \( f^G \), then for any finite graph cover \( H \) of \( G \), \( x^H \), the lift of \( x^G \) to \( H \), minimizes \( f^H \).

**Proof.** (1 \( \to \) 2) By Lemma 7.4.1, there exists a 1-cover of \( G \) and an assignment \( x^G \) such that \( x^G \) minimizes \( f^G \). Because \((\phi, \psi)\) maximizes the lower bound, we have that \( \min_x f^G(x) = LB(\phi, \psi) \).

Any assignment that minimizes \( f^G \) must also simultaneously minimize each term of the lower bound as \( \min_x f^G(x) \geq LB(\phi, \psi) \) with equality if and only if there exists an assignment that simultaneously minimizes \( \phi_i \) for all \( i \) and \( \psi_\alpha \) for all \( \alpha \).

(2 \( \to \) 3) Suppose that there exists an assignment, \( x^G \), that simultaneously minimizes each component of the lower bound. Let \( H \) be a finite cover of \( G \), and let \( x^H \) be the lift of \( x^G \) to \( H \). \( x^H \) must simultaneously minimize each term of the lower bound \( LB^H \) which implies that

$$f^H(x^H) = LB^H(\phi^H, \psi^H)$$

and that \( x^H \) minimizes \( f^H \).

(3 \( \to \) 1) Suppose that \( f^G \) has a minimum, \( x^G \), and for any graph cover \( H \) of \( G \), \( f^H \) has a minimum at \( x^H \), the lift of \( x^G \) to \( H \). By Lemma 7.4.1 every rational feasible point corresponds to
an assignment on some finite graph cover $H$ of $G$ and vice versa. Because every cover is minimized by a lift of $x^G$, the MAP LP must have an integral optimum.

The equivalence of 1 and 2 also follows from Lemmas 6.2 and 6.4 in [47]. In the case that the optimum of the MAP LP is unique and integral, we have the following corollary:

**Corollary 7.4.3.** Let $f^G$ be an objective function satisfying Assumption 2 with corresponding factor graph $G$. If $b^G$ maximizes $LB^G$, then the following are equivalent:

1. The optimum of the MAP LP for the factorization $(\phi', \psi')$ is unique and integral.
2. $x^G$ is the unique minimum of $f^G$, and for any finite graph cover $H$ of $G$, $x^H$, the lift of $x^G$ to $H$, is the unique minimum of $f^H$.

Notice that these conditions characterize the existence of an $x^*$ that simultaneously minimizes each of the components of the lower bound, but they do not provide a method for constructing such an $x^*$. As we saw earlier, local decodability is one condition that ensures that we can construct a solution to the inference problem. We will say that a $(\phi, \psi) \in S$ is locally decodable to $x^*$ if $\phi_i(x_i)$ is uniquely minimized at $x_i^*$.

**Proposition 7.4.4.** If $(\phi, \psi)$ is locally decodable to $x^G$, then $(\phi, \psi)$ maximizes the lower bound $LB^G$, and $x^G$ must minimize $f^G$.

As we can construct a solution to the inference problem from any locally decodable factorization, we would like to understand when the maxima of the lower bound have this property. Theorem 7.4.2 tells us that the dual optimal solutions cannot be locally decodable unless every graph cover of the base factor graph has a unique solution which is a lift of the solution on the base graph or, equivalently, that the optimum of MAP LP is unique and integral. That is, Theorem 7.4.2 provides necessary, but not sufficient, conditions for dual optima to be locally decodable.
A function \( f \) admits a pairwise factorization if it can be written as a sum of potential functions that depend on at most two variables. Equivalently, we require that there exists a graph \( G = (V,E) \) such that \( f \) factors in the following way:

\[
f(x) = \sum_{i \in V} \phi_i(x_i) + \sum_{(i,j) \in E} \psi_{ij}(x_i, x_j) \tag{8.1}
\]

Recall that the typical factor graph would have a node for each variable \( x_i \), a node for each factor, and an edge between \( x_i \) and \( \psi_{ij} \) for all \( \psi_{ij} \) that are not identically equal to zero. In contrast, we will draw pairwise factor graphs by omitting the factor nodes and directly connecting the variable nodes that would have been connected to the same factor node (see Figure 8.1). In this sense, we think of the pairwise potentials as living on the edges of our new factor graph. The pairwise splitting algorithm is then a message passing algorithm on this simplified factor graph.

We have already seen an example of a problem that naturally admits a pairwise factorization:

\[
\begin{align*}
\text{Figure 8.1: Simplification of pairwise factor graphs.}
\end{align*}
\]
Algorithm 7 Pairwise Synchronous Splitting Algorithm
1: Initialize the messages to some finite vector.
2: For iteration $t = 1, 2, ...$ update the the messages as follows:

$$m_{i \rightarrow j}(x_j) = \min_{x_i} \left[ \frac{\psi_{ij}(x_i, x_j)}{c_{ij}} + \phi_i(x_i) + (c_{ij} - 1)m_{j \rightarrow i}(x_i) + \sum_{k \in \partial i \setminus j} c_{ki}m_{k \rightarrow i}(x_i) \right]$$

the maximum weight independent set problem. In Example 2.2.2, we saw how the message update equations simplify for the maximum weight independent set problem, and a similar strategy can be used to simplify the updates of the splitting algorithm when we restrict the parameter vector, $c$, so that $c_i = 1$ for all $i$. Consider the message passed from the factor $\{i, j\}$ to the variable $i$:

$$m_{\{i, j\} \rightarrow i}(x_i) = \min_{x_i} \left[ \frac{\psi_{ij}(x_i, x_j)}{c_{ij}} + m_{j \rightarrow \{i, j\}}(x_j) \right]$$

$$= \min_{x_i} \left[ \frac{\psi_{ij}(x_i, x_j)}{c_{ij}} + \phi_j(x_j) + \sum_{\{k, j\} \in A \setminus \{i, j\}} c_{kj}m_{k \rightarrow j}(x_j) + (c_{ij} - 1)m_{\{i, j\} \rightarrow j}(x_j) \right] \tag{8.2}$$

We can simplify the message updates by defining $m_{i \rightarrow j}(x_j) \equiv m_{\{i, j\} \rightarrow j}(x_j)$ for all $i$. The simplified message updates appear in Algorithm 7. The message updates in Algorithm 7 are passed directly between variable nodes. The beliefs corresponding to the simplified message updates in Algorithm 7 are given by

$$\tau_{i}^{i}(x_{i}) = \phi_{i}(x_{i}) + \sum_{j \in \partial i} c_{ki}m_{k \rightarrow i}(x_{i}) \tag{8.4}$$

$$\tau_{ij}^{i}(x_{i}, x_{j}) = \phi_{j}(x_{j}) + \phi_{i}(x_{i}) + \frac{\psi_{ij}(x_i, x_j)}{c_{ij}} + \sum_{k \in \partial \setminus i \setminus j} c_{ki}m_{k \rightarrow i}(x_{i}) + (c_{ji} - 1)m_{j \rightarrow i}(x_{i})$$

$$+ \sum_{k \in \partial \setminus i \setminus j} c_{kj}m_{k \rightarrow j}(x_{j}) + (c_{ij} - 1)m_{\{i, j\} \rightarrow j}(x_{j}) \tag{8.5}$$

Note that we switched to using $\tau$ for the beliefs in the pairwise case as the definition uses the simplified messages and should not be confused with the definition in Chapter 6. These new beliefs produce a reparameterization of the objective function in terms of the simplified messages:

$$f(x) = \sum_{i} \tau_{i}(x_{i}) + \sum_{(i, j)} c_{ij} \left[ \tau_{ij}(x_{i}, x_{j}) - \tau_{i}(x_{i}) - \tau_{j}(x_{j}) \right] \tag{8.6}$$

If the parameter vector is chosen such that $c_{i} \neq 1$ for all $i$, then the message update from
Algorithm 8 Pairwise Asynchronous Splitting Algorithm

1: Initialize the messages to some finite vector.
2: Choose some ordering of the variables such that each variable is updated infinitely often, and perform the following update for each variable \( j \) in order
3: for each \( i \in \partial j \) do
4: \[ m_{i \rightarrow j}(x_j) = \min_{x_i} \left[ \frac{u_{ij}(x_i, x_j)}{c_{ij}} + \phi_i(x_i) + (c_{ij} - 1)m_{j \rightarrow i}(x_i) + \sum_{k \in \partial i \setminus j} c_{ki}m_{k \rightarrow i}(x_i) \right] \]
5: end for

Algorithm 2 can still be rewritten in terms of messages passed only between the variable nodes. In this case, the resulting message updates need not correspond to a local message passing scheme; the message passed from \( i \) to \( j \) depends on the messages passed into both \( i \) and \( j \). Consequently, in this chapter, we focus only on the case that \( c_i = 1 \) for all \( i \).

Any factor graph can be converted into a pairwise factor graph by modifying the objective function [56]. As such, optimization problems over pairwise factor graphs represent an important special case in the study of message passing algorithms. In this chapter, we explore the special properties of pairwise factor graphs. In Section 8.1, we show that, for pairwise factor graphs, the synchronous splitting algorithm can be viewed as an asynchronous algorithm on a special graph cover. In Section 8.2, we further specialize our results to pairwise binary graphical models. For this restricted class of graphical models, we provide necessary and sufficient conditions under which the fixed points of the splitting algorithm are guaranteed to be locally decodable. These theoretical results are complemented by several applications.

8.1 Asynchronous and Synchronous Convergence

In the pairwise case, we can construct an asynchronous version of Algorithm 7 by using the simplified message updates. The resulting updates are described in Algorithm 8. As before, the asynchronous algorithm converges under appropriate choices of the parameter vector given by Theorem 6.2.2.

Typically, convergence of the synchronous algorithm is proven, for specific problems, by analyzing the computation trees produced at each time step (see, for example, Chapter 3). In contrast, we prove the convergence of the synchronous algorithm by exploiting its relationship to the asynchronous algorithm: the synchronous algorithm can be viewed as an asynchronous algorithm on a specific 2-cover.
First, every pairwise factor graph, $G = (V_G, E_G)$, admits a bipartite 2-cover, $H = (V_G \times \{1, 2\}, E_H)$, called the Kronecker double cover of $G$. We will denote copies of the variable $x_i$ in this 2-cover as $x_{i1}$ and $x_{i2}$. For every edge $(i, j) \in E_G$, $(i_1, j_2)$ and $(i_2, j_1)$ belong to $E_H$. In this way, nodes labeled with a one are only connected to nodes labeled with a two (see Figure 8.2). Note that if $G$ is already a bipartite graph, then the Kronecker double cover of $G$ is simply two disjoint copies of $G$.

We can view the synchronous algorithm described in Algorithm 7 as a specific asynchronous algorithm on the Kronecker double cover where we perform the asynchronous update for every variable in the same partition on alternating iterations (see Algorithm 9).

**Algorithm 9** Bipartite Asynchronous Splitting Algorithm

1: Initialize the messages to some finite vector.
2: Iterate the following until convergence: update all of the outgoing messages from nodes labeled one to nodes labeled two and then update all of the outgoing messages from nodes labeled two to nodes labeled one using the asynchronous update rule:

$$m_{i \rightarrow j}(x_j) = \min_{x_i} \left[ \frac{\psi_{ij}(x_i, x_j)}{c_{ij}} + \sum_{k \in \partial i \setminus j} c_{ki} m_{k \rightarrow i}(x_i) + \phi_i(x_i) + (c_{ij} - 1) m_{j \rightarrow i}(x_i) \right]$$

By construction, the message vector produced by Algorithm 9 is simply a concatenation of two consecutive time steps of the synchronous algorithm. Specifically, for all $t \geq 1$

$$m^t_H = \begin{bmatrix} m_G^{2t-1} \\ m_G^{2t-2} \end{bmatrix}.$$ \hspace{1cm} (8.7)

Therefore, for any pairwise factor graph, the messages passed by the synchronous algorithm are identical to those passed by an asynchronous algorithm on the Kronecker double cover. The proof of convergence, for an appropriate choice of the parameter vector, then follows from the convergence of the asynchronous algorithm.
Theorem 8.1.1. Suppose $c$ is as in Theorem 6.2.2. If the vector of messages is real-valued at each step of the algorithm, then Algorithm 9 converges.

Proof. The update from a node labeled one to a node labeled two only depends on messages received from nodes labeled two. As a result, we can update multiple nodes in the same part of the partition simultaneously. The proof then follows in exactly the same way as Theorem 6.2.2.

Corollary 8.1.2. Suppose $c$ is as in Theorem 6.2.2. If the vector of messages is real-valued at each step of the algorithm, then Algorithm 7 converges.

8.2 Pairwise Binary Graphical Models

In the special case that the state space is binary (i.e. each $x_i$ can only take one of two values) we can strengthen the results of the previous sections. Previous work on pairwise binary graphical models has focused on the relationship between the converged beliefs and solutions to the MAP LP [22, 57]. In this work, we focus on the relationships between the base factor graph and its covers.

In the context of graph covers, the most surprising property of pairwise binary graphical models is that, for any choice of the vector $c$ satisfying the conditions of Theorem 5.2.3, if the algorithm converges, we can always construct a minimizing assignment on a 2-cover:

Theorem 8.2.1. Let $\tau$ be a vector of admissible and min-consistent beliefs for the objective function $f^G$, with factor graph $G$ and parameter vector $c$, obtained from the vector of messages $m$. If the factor graph is pairwise binary and the vector $c$ satisfies the conditions of Theorem 5.2.3, then there exists a 2-cover, $H$, of $G$ and an assignment, $y^*$, on that 2-cover such that $y^*$ minimizes $f^H$.

Proof. Without loss of generality we can assume that $X = \{0, 1\}$. We will construct a 2-cover, $H$, of the factor graph $G$ and an assignment $y^*$ such that $y^*$ minimizes $f^H$. We will index the copies of variable $i \in G$ in the factor graph $H$ as $i_1$ and $i_2$. First, we will construct the assignment. If $\arg \min_{x_i} \tau_i(x_i)$ is unique, then set $y^*_{i_1} = y^*_{i_2} = \arg \min_{x_i} \tau_i(x_i)$. Otherwise, set $y^*_{i_1} = 0$ and $y^*_{i_2} = 1$. Now, we will construct a 2-cover, $H$, such that $y^*$ minimizes each of the beliefs. We will do this edge by edge. Consider the edge $(i, j) \in E$. There are several possibilities:

1. $\tau_i$ and $\tau_j$ have unique argmins. In this case, $\tau_{ij}$ is minimized at $b_{ij}(y^*_{i_1}, y^*_{j_1})$. So, we can add the edges $(i_1, j_1)$ and $(i_2, j_2)$ to $H$. The corresponding beliefs $\tau_{i_1,j_1}$ and $\tau_{i_2,j_2}$ are minimized at $y^*$.
2. \( \tau_i \) has a unique argmin and \( b_j \) is minimized at both 0 and 1 (or vice versa). In this case, we have \( y^*_i = y^*_i, y^*_j = 0, \) and \( y^*_j = 1. \) By min-consistency, we can conclude that \( \tau_{ij} \) is minimized at \((y^*_i, 0)\) and \((y^*_j, 1)\). Therefore, we can add the edges \((i_1, j_1)\) and \((i_2, j_2)\) to \( H \).

3. \( \tau_i \) and \( \tau_j \) are minimized at both 0 and 1. In this case, we have \( y^*_i = 0, y^*_j = 1, y^*_j = 0, \) and \( y^*_j = 1. \) By min-consistency, there is an assignment that minimizes \( \tau_{ij} \) with \( x_i = 0 \) and an assignment that minimizes \( \tau_{ij} \) with \( x_i = 1 \). This means that \( \arg \min_{x_i, x_j} \tau_{ij} \) contains at least one of the sets \{\((0, 0), (1, 1)\)\} or \{\((0, 1), (1, 0)\)\}. In the first case, we can add the edges \((i_1, j_1)\) and \((i_2, j_2)\) to \( H \), and in the second case, we can add \((i_1, j_2)\) and \((i_2, j_1)\) to \( H \).

**Corollary 8.2.2.** Theorem 8.2.1 continues to hold for any vector of messages, \( m \), such that the lower bound, \( \text{LB}_G(m) \), cannot be improved by subsequent iteration of Algorithm 8.

**Proof.** The lower bound has converged if no single variable update can improve the bound. By the arguments in the proof of Theorem 6.2.2, this must mean that for each \( j \) there exists an \( x_j \) that simultaneously minimizes \((1 - \sum_{i \in \partial j} c_{ij})b_j\) and \( \min_{x_i} b_{ij} \) for each \( i \in \partial j \). Notice that these beliefs may or may not be min-consistent. However, as observed in the proof of Theorem 6.2.2, when \( \text{LB}_j \) cannot be improved it is independent of the messages passed from \( i \) to \( j \) for each \( i \in \partial j \). As a result, we may assume that the beliefs are min-consistent as they must have the same minima as the min-consistent beliefs. We can then use this observation to construct a vector \( y^* \) as above: if there is a unique \( x_j \) that simultaneously minimizes \((1 - \sum_{i \in \partial j} c_{ij})b_j\) and \( \min_{x_i} b_{ij} \) for each \( i \in \partial j \), then set \( y^*_j \) and \( y^*_j \) equal to this \( x_j \). Otherwise, set \( y^*_j = 0 \) and \( y^*_j = 1. \) The 2-cover can then be constructed using the vector \( y^* \) as above.

As a consequence of this corollary, we can conclude that Algorithm 7 converges to a vector of messages that maximizes the lower bound (i.e. the coordinate ascent algorithm does not get stuck). As a result, the corollary generalizes Theorem 4 in [22] using graph covers instead of duality (which allows us to apply the result beyond functions satisfying Assumption 2).

Theorem 8.2.1 continues to hold even if the parameter vector, \( c \), does not satisfy the conditions of Theorem 5.2.3. In this case, the assignment on the 2-cover need not minimize the objective function corresponding to the cover. The assignment must, however, simultaneously minimize each of the beliefs on the 2-cover:
Corollary 8.2.3. Let $\tau$ be a vector of admissible and min-consistent beliefs for the objective function $f^G$, with factor graph $G$ and parameter vector $c$, obtained from the vector of messages $m$. If the factor graph is pairwise binary, then there exists a 2-cover, $H$, of $G$ and an assignment, $y^*$, on that 2-cover such that $y^*$ simultaneously minimizes each of the beliefs on $H$.

Given Theorem 8.2.1, we can also explicitly describe when Algorithms 7 and 8 converge to locally decodable beliefs. As every vector of admissible and min-consistent beliefs maximizes a lower bound that is tight on some 2-cover, we can simplify Theorem 7.4.2 for the pairwise binary case.

Corollary 8.2.4. Let $c$ satisfy the conditions of Theorem 6.2.2. Algorithms 8 and 7 converge to locally decodable beliefs if and only if for every 2-cover $H$ of $G$, $f^H$ has a unique minimum.

8.2.1 Partial Solutions

Sometimes the splitting algorithm may converge to a collection of beliefs such that some of the beliefs have a unique argmin while others do not. In this case, we might wonder if the partial solution, generated only from those beliefs that have a unique argmin, can be extended to a global minimum.

More formally, let $\tau$ be a vector of admissible and min-consistent beliefs for the function $f$. Let $F \subset V$ be the set of "fixed" variables (i.e. those variables, $i$, such that $\arg\min_{x_i} \tau_i(x_i)$ is unique), and let $x_i^* = \arg\min_{x_i} \tau_i(x_i)$ for each $i \in F$. If $f$ can be written as a conical combination of these beliefs, then the partial solution $x_F^*$ can always be extended to a global minimum of the objective function. This result was proven independently in [22] and [58]:

Theorem 8.2.5. Let $\tau$ be a vector of admissible and min-consistent beliefs for the function $f$ with parameter vector $c$. If $f$ can be written as a conical combination of the beliefs, then the partial solution $x_F^*$ can be extended to a global minimum of the objective function.

Proof. Let $F_r = V \setminus F$ denote the free variables. $F_r$ and $F$ partition the vertex set into two pieces. There are three different kinds of edges in this graph: those that connect only vertices in $F$, those that connect only vertices in $F_r$, and those edges joining a vertex in $F$ to a vertex in $F_r$.

Let $g_{F_r}(x_{F_r})$ be the portion of the conical decomposition that depends only on the variables in $F_r$. We will extend $x_F^*$ to a global minimum by choosing $x_F^* \in \arg\min_{x_{F_r}} g_{F_r}(x_{F_r})$. 

83
To see that \(x^*\) minimizes \(f\), let \(g_F(x_F)\) be the portion of the conical decomposition that depends only on the variables in \(F\), and let \(g_b(x) = f(x) - g_F(x_F) - g_{F_r}(x_{F_r})\). Because \(x^*_F\) minimizes \(g_F(x_F)\) by min-consistency and \(x^*_r\) minimizes \(g_{F_r}(x_{F_r})\) by construction, we only need to show that \(x^*\) minimizes \(g_b(x)\).

Without loss of generality, we can assume that each \(x_i\) can only take the value zero or one. Choose \(i \in F\) and \(j \in F_r\) such that \((i, j) \in G\). Observe that, by min-consistency, \(\tau_{ij}(x_i, x_j)\) must be minimized at \((x^*_i, 0)\) and \((x^*_i, 1)\). This means that \(x^*\) simultaneously minimizes \(\tau_{ij}(x_i, x_j), \tau_i(x_i), \) and \(\tau_j(x_j)\), and consequently, \(x^*\) must also minimize \(\tau_{ij}(x_i, x_j) - \tau_i(x_i)\) and \(\tau_{ij}(x_i, x_j) - \tau_j(x_j)\).

The result then follows that \(x^*\) minimizes \(f(x) = g_F(x_F) + g_{F_r}(x_{F_r}) + g_b(x)\).

8.2.2 Applications

From Theorem 8.2.1, if the splitting algorithm converges, we can construct an assignment on a 2-cover (a 4-cover in the case of the synchronous algorithm) of our original problem from the converged beliefs whether or not the objective function has a unique minimizing assignment. Hence, given any pairwise binary graphical model, we only need to understand how to transform solutions on a 2-cover to solutions of the original problem. In this section, we discuss several applications of the theoretical tools developed above for pairwise binary graphical models.

Maximum Weight Independent Set

The maximum weight independent set problem on the graph \(G = (V, E)\) with a vector of weights, \(w\), is a solution to the following integer program:

\[
\begin{align*}
\text{maximize} & \quad w^T x \\
\text{subject to} & \quad x_i + x_j \leq 1 \quad \forall (i, j) \in E \\
& \quad x_i \in \{0, 1\} \quad \forall i \in V
\end{align*}
\]

This integer program can be relaxed into a linear program by allowing the \(x_i\) to take values in the interval \([0, 1]\).

As before, we can construct an objective function for this integer program as follows: \(f(x) = -\sum_{i \in V} w_i x_i - \sum_{(i, j) \in E} \log\{x_i + x_j \leq 1\}\) where \(\{x_i + x_j \leq 1\}\) is one if \(x_i + x_j \leq 1\) and zero otherwise. We can check that the LP relaxation for this problem is equivalent to the MAP LP.

By Theorem 8.2.1, we can construct a 2-cover and an assignment, \(y^*\), that minimizes the objec-
tive function on that 2-cover (a 4-cover in the case of the synchronous algorithm) from the converged beliefs. Notice that \( y^* \) must also minimize the linear relaxation corresponding to the 2-cover because the lower bound is tight and the linear program corresponding to the maximum weight independent set problem is half-integral. Now, we can construct a solution to the base linear program by setting \( x_i^* = \frac{x_{i1} + x_{i2}}{2} \), as averaging cannot cause any of the constraints to become violated.

Notice that, in order to construct the assignment \( x^* \), we do not need to construct the covers explicitly. For Algorithm 8, the above procedure is equivalent to setting \( x_i^* \) to the argmin of the \( i^{th} \) belief, if it is unique, and setting it to .5 otherwise. Similarly, in the synchronous case, we set \( x_i^* \) to the argmin of the \( i_1^{th} \) and \( i_2^{th} \) beliefs if they are equivalent and unique and .5 otherwise.

The MAP LP and the LP relaxation do not necessarily correspond if we introduce additional constraints to the integer program:

**Example 8.2.1 (Adding constraints to the maximum weight independent set problem.)**

Adding redundant constraints to the objective function \( f \) can have a surprising effect on the solution to the MAP LP. For example, consider the following integer program corresponding to the maximum weight independent set problem on a cycle:

\[
\begin{align*}
\text{minimize} & \quad -3x_1 - 2x_2 - 2x_3 \\
\text{subject to} & \quad x_1 + x_2 \leq 1 \\
& \quad x_1 + x_3 \leq 1 \\
& \quad x_2 + x_3 \leq 1 \\
& \quad x_1, x_2, x_3 \in \{0, 1\}
\end{align*}
\]

We could convert this integer program into a linear program by relaxing the integrality constraint. We could also construct a MAP LP for this problem in the standard way from the pairwise binary objective function \( f(x) = -3x_1 - 2x_2 - 2x_3 - \log\{x_1 + x_2 \leq 1\} - \log\{x_1 + x_3 \leq 1\} - \log\{x_2 + x_3 \leq 1\} \). We can check that the minimum of the relaxed LP is -3.5, which is also the minimum value of the MAP LP.

Now, consider adding the constraint \(-3x_1 - 2x_2 - 2x_3 \geq -3.5\) to the integer program. This constraint does not change the solution to the integer program or the linear relaxation. However, the minimum of the MAP LP is now -3. To see why this is the case, observe that on any graph cover the constraint \(-3x_i - 2x_j - 2x_k \geq -3.5\) enforces that at most one of any of these nodes can be chosen to be in the independent set. The optimal solution, then, is to set \( x_i = 1 \) for each \( i \) in
the cover that is a copy of node 1 and zero otherwise. This solution is unique on each cover, and is clearly a lift of the MAP assignment $x_1 = 1, x_2 = 0, x_3 = 0$.

---

**Minimum s-t Cuts**

We now turn our attention to a problem that is known to be solvable in polynomial time, the minimum s-t cut problem. An s-t cut in a directed graph, $G = (V, E)$, is a partition of the vertex set into two disjoint sets $A$ and $B$ such that $s \in A$ and $t \in B$. The set of edges cut by this partition is $C = \{(i, j) \in E | i \in A, j \in B\}$. Further, if there is a positive weight associated to each edge of the graph, then the capacity of the cut is defined to be $\sum_{e \in C} w_e$. The minimum s-t cut problem is to find the partition that minimizes the capacity of the cut, and it can be formulated as the following linear program:

\[
\begin{align*}
\text{minimize} & \quad w^T x \\
\text{subject to} & \quad z_j - z_i + x_{ij} \geq 0 \quad \forall (i, j) \in E \\
& \quad z_i \in \mathbb{R} \quad \forall i \in V \\
& \quad z_s = 1, z_t = 0
\end{align*}
\]

Many of the constraints in this LP are not pairwise. However, we still can formulate this problem as a pairwise graphical model. Let $X_i = 1, X_t = 0, X_i = \{0, 1\}$ for each $i \in V \setminus \{s, t\}$ and define $f(x) = \sum_{(i, j) \in E} w_{ij}x_i(1 - x_j)$.

Again, upon convergence of the asynchronous algorithm we can construct a 2-cover and a minimum s-t cut on that 2-cover. Consider the LP problem corresponding to this 2-cover. As a consequence of the max-flow min-cut theorem, any fractional solution to this LP can be converted into integral solution (see Theorem 7.2 in [44]). Therefore, the assignment constructed by Theorem 8.2.1 must maximize the linear program.

Averaging the copies on the 2-cover no longer produces a solution to the linear programming formulation corresponding to the base graph. As before, this solution may be fractional. Again, as a consequence of the max-flow min-cut theorem, any fractional solution to this LP can be converted into integral solution.

We do not need to explicitly construct any of the covers to deduce an assignment. We only need to set $x_i^*$ equal to the argmin of the $i^{th}$ belief if it is unique and zero otherwise. Hence, we can construct a minimum s-t cut from the converged beliefs whether or not the original graph has
Lastly, we explore a classic problem from statistical physics: finding a minimum energy configuration for the Ising model. Specifically, given a graph $G = (V, E)$, the goal is to minimize $f(x) = -\sum_i B_i x_i - \sum_{(i,j) \in E} J_{ij} x_i x_j$ where $x_i \in \{-1, 1\}$. The model is called ferromagnetic if $J_{ij} > 0$ for all $(i, j) \in E$.

Minimizing the energy corresponding to the ferromagnetic Ising model can be formulated as a min-cut problem [15, 35]. This result is not surprising as the binary assignments produce natural partitions of the graph. Consequently, we could formulate a min-cut objective function, as in the previous section, to solve this problem.

However, using the connection with minimum cuts, we can apply the same arguments as in the previous section to show that both the synchronous and asynchronous algorithms converge to the minimizing assignment for the specific factorization given above. Again, we can exploit the relationship between the min-cut LP on the 2-cover guaranteed by Theorem 8.2.1 and the LP corresponding to the original problem to construct an assignment (e.g. for the asynchronous algorithm we set $x_i^*$ equal to the argmin of the $i^{th}$ belief if it is unique and one otherwise).
Chapter 9

Other Convergent and Correct Message Passing Algorithms

Recent work has produced other asynchronous message passing algorithms that are provably convergent: MPLP [14], serial tree-reweighted max-product (TRW-S) [21], max-sum diffusion (MSD) [59], and the norm-product algorithm[16]. Like Algorithm 4, these asynchronous message passing algorithms are convergent in the sense that they can each be viewed as coordinate ascent schemes over concave lower bounds.

All of these algorithms, with the exception of the norm-product algorithm, were shown to be members of a particular family of bound minimizing algorithms [30]. We note that, even when the parameter vector satisfies the conditions of Theorem 6.2.2, Algorithm 4 is still not strictly a member of the family of bound minimizing algorithms. The disparity occurs because the definition of a bound minimizing algorithm as presented therein would require $b_\alpha$ to be min-consistent with respect to $x_j$ for all $j \in \alpha$ after the update is performed over the edge $(i, \alpha)$. Instead, Algorithm 4 only guarantees that $b_\alpha$ is min-consistent with respect to $x_i$ after the update.

In this chapter, we show that all of these message passing algorithms can be seen as coordinate ascent schemes over concave lower bounds. More specifically, their derivations, with, perhaps, the exception of the norm-product algorithm, can be seen to follow the same formula from the previous chapters:

1. Choose a reparameterization.
2. Construct a lower bound.

3. Perform coordinate ascent in an attempt to maximize the bound.

In some cases, the message passing algorithms themselves can be seen as a special case of the splitting algorithm. While in other cases, a slight tweak to the definition of min-consistency allows us to apply the results of the previous chapters. We will divide our analysis into two separate cases: local message passing algorithms and non-local message passing algorithms.

9.1 Local Message Passing Algorithms

Recall that a message passing algorithm is local if each message passed between two nodes of the factor graph depends only on the messages received by the source and the potential at the source (and possibly the potential along the edge in the case of pairwise factorizations). Such message passing schemes are local in the sense that messages passed in this network depend only on local information. The min-sum algorithm and the splitting algorithm are both local message passing schemes. In this section, we describe another local message passing algorithm, the TRMP algorithm, that can be seen as a special case of the splitting algorithm.

9.1.1 TRW-S and TRMP

The tree-reweighted belief propagation algorithm (TRBP) was first proposed in [52], and the application of similar ideas to the MAP inference problem is known as the tree-reweighted max-product algorithm (TRMP) [53]. At the heart of the min-sum analog of the TRMP algorithm is the observation that the objective function can be bounded from below by a convex combination of functions that depend only on factor induced subtrees of the factor graph. As we will see below, the message updates of the TRMP algorithm, defined in [53], are a special case of the splitting algorithm.

Although the TRMP algorithm can be derived for general factor graphs, for simplicity, we consider the algorithm on a pairwise factor graph $G = (V, E)$ with corresponding objective function $f$. Let $\mathcal{T}$ be the set of all spanning trees on $G$ (recall that we omit the factor nodes from the factor graph in the pairwise case), and let $\mu$ be a probability distribution over $\mathcal{T}$ such that every edge has a nonzero probability of occurring in at least one spanning tree. We define $c_i = 1$ for all $i$ and $c_{ij} = Pr_{\mu}[(i,j) \in T]$ corresponding to the edge appearance probabilities. Let $\tau$ be a vector of
admissible and min-consistent beliefs for $f$. We can write the objective function $f$ as

$$f(x) = \sum_{i \in V} \tau_i(x_i) + \sum_{(i,j) \in E} c_{ij} \left[ \tau_{ij}(x_i, x_j) - \tau_i(x_i) - \tau_j(x_j) \right] \quad (9.1)$$

$$= \sum_{T \in \mathcal{T}} \mu(T) \left[ \sum_{i \in V_T} \tau_i(x_i) + \sum_{(i,j) \in E_T} \left[ \tau_{ij}(x_i, x_j) - \tau_i(x_i) - \tau_j(x_j) \right] \right] \quad (9.2)$$

With this rewriting, we can see that this choice of $c$ results in a conical decomposition. For each $T \in \mathcal{T}$, designate a variable node $r_T \in T$ as the root of $T$. Let $p_T^T(\alpha)$ denote the parent of factor node $\alpha \in T$. We can now write,

$$f(x) = \sum_{T \in \mathcal{T}} \mu(T) \left[ \sum_{i \in V_T} \tau_i(x_i) + \sum_{(i,j) \in E_T} \left[ \tau_{ij}(x_i, x_j) - \tau_i(x_i) - \tau_j(x_j) \right] \right] \quad (9.3)$$

$$= \sum_{T \in \mathcal{T}} \mu(T) \left[ \tau_{r_T}(x_{r_T}) + \sum_{i \in V_T, i \neq r_T} \left[ \tau_{p_T^T(i)}(x_i, x_{p_T^T(i)}) - \tau_{p_T^T(i)}(x_{p_T^T(i)}) \right] \right] \quad (9.4)$$

Because $\mu(T) \geq 0$ for all $T \in \mathcal{T}$, we can conclude that $f$ can be written as a conical combination of the beliefs. The TRMP update, defined in [53], is then exactly Algorithm 2 with the the vector $c$ chosen as above. All of the results from the previous chapters can then be applied to this special case. For example, by Theorem 5.2.1, convergence of the TRMP algorithm to locally decodable beliefs implies correctness.

The TRMP algorithm was developed based on the observation that the min-sum algorithm is correct on trees. However, a similar derivation can be made if $\mu$ is a probability distribution over all subgraphs of $G$ containing at most one cycle. In this case, we would obtain a reparameterization of the objective function as a convex combination of subgraphs containing only a single cycle.

Although the TRMP algorithm guarantees correctness upon convergence to locally decodable beliefs, the algorithm need not converge, even if we use Algorithm 4. Specifically, the vector $c$ does not necessarily satisfy the conditions of Theorem 6.2.2. The solution, proposed in [21], is to perform the message updates asynchronously and in a specific order that is guaranteed to improve the lower bound. The resulting algorithm, known as the TRW-S algorithm, is then a convergent and correct version of the TRMP algorithm.
9.2 Non-local Message Passing

Many other, non-local, message passing algorithms have been proposed to solve the inference problem. Some of these algorithms such as the EMPLP algorithm can be viewed as a special case of the splitting algorithm, but with slight modifications that make the final message passing scheme non-local, while others may not even have an interpretation in terms of reparameterizations. Such algorithms are less desirable than local message passing algorithms in certain applications where communication is limited only to immediate neighbors and the potential functions are only known locally (e.g. sensor nets).

9.2.1 MPLP

The MPLP algorithm was designed to provide a convergent message passing scheme for graphical models without the tunable parameters required by the TRMP algorithm [14]. Like TRMP, the MPLP algorithm was originally derived by constructing a special dual of the MAP LP from which a concave lower bound can be extracted. The MPLP algorithm is then a coordinate ascent scheme for this concave lower bound. The MPLP algorithm was initially derived in terms of pairwise factor graphs, but can be extended, with some work, to arbitrary factor graphs.

Again, consider a pairwise factor graph, $G = (V, E)$, with corresponding objective function, $f$. Let $c_i = 1/2$ for all $i$ and $c_{ij} = 1$ for all $i$. Note that since $c_i$ is not equal to 1 for all $i$, we cannot use the message updates in Algorithms 7 and 8. Instead, we will use the messages and beliefs as they appear in Algorithm 2. This choice of $c$ produces the following reparameterization:

$$
\begin{align*}
    f(x) &= \sum_{i \in V} \frac{b_i(x_i)}{2} + \sum_{(i,j) \in E} \left[ b_{ij}(x_i, x_j) - \frac{b_i(x_i)}{2} - \frac{b_j(x_j)}{2} \right] \\
    &= \sum_{i \in V} \frac{b_i(x_i)}{2} + \sum_{(i,j) \in E} \frac{1}{2} \left[ b_{ij}(x_i, x_j) - b_i(x_i) \right] + \left[ b_{ij}(x_i, x_j) - b_j(x_j) \right]
\end{align*}
$$

From Equation 9.6, we can see that this choice of $c$ produces a conical decomposition.

Several variants of the MPLP algorithm were presented in the original paper. One such variant, the EMPLP algorithm, can be seen as a coordinate ascent scheme on the following lower bound:

$$
\begin{align*}
    \min_x f(x) &\geq \sum_{i \in V} \min_{x_i} \frac{b_i(x_i)}{2} + \sum_{(i,j) \in E} \frac{1}{2} \left[ \min_{x_i, x_j} b_{ij}(x_i, x_j) - b_i(x_i) \right] + \min_{x_i, x_j} \left[ b_{ij}(x_i, x_j) - b_j(x_j) \right]
\end{align*}
$$
Consider the pair of message updates given in Algorithm 2. Analogous to the case when \( c_i = 1 \) for all \( i \), we rewrite the update in terms of messages passed directly between the variables as follows:

\[
m_{i \rightarrow j}(x_j) = \min_{x_i} \psi_{ij}(x_i, x_j) - \frac{1}{2} \sum_{k \in \partial j \setminus i} m_{k \rightarrow j}^{t-1}(x_j) + \frac{1}{2} \sum_{k \in \partial i \setminus j} m_{k \rightarrow i}^{t-1}(x_i) \tag{9.8}
\]

Let \( \hat{m}_t \) denote the messages passed by the EMPLP algorithm at time \( t \). The message updates for the EMPLP algorithm are then given by \( \hat{m}_t = 1/2 m_t \) for all \( t \geq 0 \). The condensed message updates are not local: the message from \( i \) to \( j \) depends on messages entering both \( i \) and \( j \). As was the case for the TRMP algorithm, we can extend all of the previous results to the MPLP case.

### 9.2.2 Max-Sum Diffusion

The max-sum diffusion algorithm and a variant known as the augmenting DAG algorithm were designed to solve the max-sum problem (i.e. the negated version of the min-sum problem). Although discovered in the 1970s by Ukrainian scientists, most of the original work on these algorithms remained either in Russian or unpublished until a recent survey article [59]. The augmenting DAG algorithm was suggested in [43] and later expanded in [25]. The max-sum diffusion algorithm was discovered independently by two authors [26, 12], but neither result was ever published.

Here, we derive the min-sum analog of the max-sum diffusion algorithm using the machinery that we have developed for the splitting algorithm. Although the algorithm is a coordinate ascent scheme over a familiar lower bound, the message updates are not an instance of the splitting algorithm primarily because the fixed points are not min-consistent in the sense of Definition 2.2.4.

The max-sum diffusion algorithm was originally described only for pairwise factorizations. However, we will see that the algorithm can be derived for general factor graphs. Consider the reparameterization of the objective function corresponding to the standard min-sum algorithm (i.e. \( c_i = 1 \) for all \( i \) and \( c_\alpha = 1 \) for all \( \alpha \)):

\[
f(x) = \sum_i b_i(x_i) + \sum_\alpha [b_\alpha(x_\alpha) - \sum_{k \in \alpha} m_{\alpha \rightarrow k}(x_k)] = \sum_i \min_{x_i} b_i(x_i) + \sum_\alpha \min_{x_\alpha} [\psi_{\alpha}(x_\alpha) - \sum_{k \in \alpha} m_{\alpha \rightarrow k}(x_k)] \tag{9.9, 10}
\]
This reparameterization can be lower bounded as:

\[
\min_x f(x) \geq \sum_i \min_{x_i} b_i(x_i) + \sum_\alpha \min_{x_\alpha} [\psi_\alpha(x_\alpha) - \sum_{k \in \alpha} m_{\alpha \rightarrow k}(x_k)] \quad (9.11)
\]

The max-sum diffusion algorithm is a coordinate ascent message passing scheme that improves the above lower bound. Unlike the reparameterizations that produced the TRMP and MPLP algorithms, whether or not this reparameterization can be written as a conical combination of the beliefs depends on the underlying factor graph. As such, if we choose an algorithm that converges to a min-consistent vector of beliefs, then we will not be guaranteed correctness.

Instead, the max-sum diffusion algorithm ensures a different form of consistency. Namely, the algorithm guarantees that the fixed points of the message passing scheme satisfy the following for each \( \alpha \) and each \( i \in \alpha \):

\[
\min_{x_\alpha \backslash i} [b_\alpha(x_\alpha) - \sum_{k \in \alpha} b_k(x_k)] = b_i(x_i) \quad (9.12)
\]

Again, there are many message updates that will guarantee this form of consistency upon convergence. The one chosen by the developers of the max-sum diffusion algorithm was:

\[
m_{\alpha \rightarrow i}(x_i) = m_{\alpha \rightarrow i}(x_i) + \frac{1}{2} \min_{x_\alpha \backslash i} [b_\alpha(x_\alpha) - \sum_{k \in \alpha} b_k(x_k) - b_i(x_i)] \quad (9.13)
\]

We can obtain a simpler message update rule that does not depend on the previous iteration:

\[
m_{\alpha \rightarrow i}(x_i) = \frac{1}{2} \min_{x_\alpha \backslash i} [\psi_\alpha(x_\alpha) - \sum_{k \in \alpha \backslash i} m_{\alpha \rightarrow k}(x_k)] - \frac{1}{2} \min_{\beta \in \partial_i \backslash \alpha} [\phi_\beta(x_i) - \sum_{\beta \in \partial_i \backslash \alpha} m_\beta \rightarrow i(x_i)] \quad (9.14)
\]

After computing \( m_{\alpha \rightarrow i}(x_i) \) for each \( x_i \), the lower bound can only increase. Further, we can check that if the algorithm converges to locally decodable beliefs, then this estimate is guaranteed to be correct. This follows by replacing our notion of min-consistency with that of Equation 9.12.

The lower bound in Equation 9.11, can also be shown to be dual to the MAP LP [59]. Finally, note that, like the MPLP algorithm, the message updates are not local.
9.2.3 Norm-Product

The norm-product algorithm, like the above algorithms, is a coordinate ascent scheme for maximizing a concave dual objective function [16]. Unlike the previous algorithms, however, whether or not the norm-product algorithm produces a reparameterization of the objective function remains an open question.

The algorithm is derived by studying the general problem of minimizing a convex objective function having a particular form. The derivation of the algorithm uses more or less standard tools from convex analysis including Fenchel and Lagrangian duality. While the derivation of this algorithm is beyond the scope of this work, it is worth noting that, like the splitting algorithm, the norm-product algorithm is parameterized by a real vector. For some choices of the parameter vector for both algorithms, the norm-product algorithm agrees with the asynchronous splitting algorithm (Algorithm 4). For these choices of the parameter vector, the norm-product algorithm acts like a local message passing algorithm. However, for arbitrary choices of the parameter vector, the norm-product message updates may be non-local.
Chapter 10

Convex Optimization

Much of the previous discussion has been motivated by classic problems in combinatorial optimization. In these cases, the objective function was a mapping from a finite state space to the real numbers. In this chapter, we investigate the properties of the splitting algorithm when it is applied to minimize convex functions over a continuous state space. Let $f : C \to \mathbb{R}$ be a convex function where $C \subseteq \mathbb{R}^n$ is a convex set in $\mathbb{R}^n$. If $C \neq \mathbb{R}^n$, then we can add constraints to ensure that $f$ is finite only over the set $C$. Although some of the previous results were restricted to finite state spaces, we show that many of the same ideas (graph covers, pseudocodewords, concave lower bounds, etc.) will continue to play an important role in our understanding of the behavior of the splitting algorithm for continuous functions.

Much of the work on convex function minimization, with respect to the min-sum algorithm, has focused on the quadratic minimization problem [32, 31, 20, 29]. The quadratic minimization problem is special for two reasons. First, the sum-product algorithm for computing the mean of a multivariate Gaussian distribution, known as Gaussian belief propagation (GaBP), is identical to the min-sum algorithm for minimizing a multivariate quadratic function. Second, the minimization in the message update equations can be computed explicitly. In Section 10.1, we discuss the splitting algorithm for quadratic minimization. We show how certain choices of the parameter vector allow us to solve problems for which the standard min-sum algorithm fails. We also relate the convergence of the splitting algorithm to other iterative algorithms for the quadratic minimization problem.

With the results for the quadratic minimization problem as a guide, we discuss the general problem of minimizing a convex function in Section 10.2. Finally, in Section 10.3, we discuss the
related problem of minimizing a submodular function via the splitting algorithm.

### 10.1 Quadratic Minimization

Let $\Gamma \in \mathbb{R}^{n \times n}$ be a symmetric positive definite matrix and $h \in \mathbb{R}^n$. The quadratic minimization problem is to find the $x \in \mathbb{R}^n$ that minimizes $\frac{1}{2} x^T \Gamma x - h^T x$. The global optimum must satisfy $\Gamma x = h$, and as a result, minimizing a positive definite quadratic function is equivalent to solving a positive definite linear system.

In this section we restrict ourselves, without loss of generality, to symmetric $\Gamma$ as the quadratic function $\frac{1}{2} x^T \Gamma x - h^T x$ is equivalent to $\frac{1}{2} x^T \left[ \frac{1}{2}(\Gamma + \Gamma^T) \right] x - h^T x$ for any $\Gamma \in \mathbb{R}^{n \times n}$:

\[
\frac{1}{2} x^T \Gamma x - h^T x = \frac{1}{2} x^T \left[ \frac{1}{2}(\Gamma + \Gamma^T) + \frac{1}{2}(\Gamma - \Gamma^T) \right] x - h^T x \tag{10.1}
\]

\[
= \frac{1}{2} x^T \left[ \frac{1}{2}(\Gamma + \Gamma^T) \right] x + \frac{1}{2} x^T \left[ \frac{1}{2}(\Gamma - \Gamma^T) \right] x - h^T x \tag{10.2}
\]

\[
= \frac{1}{2} x^T \left[ \frac{1}{2}(\Gamma + \Gamma^T) \right] x - h^T x \tag{10.3}
\]

The matrix $\frac{1}{2}(\Gamma + \Gamma^T)$ is known as the symmetric part of $\Gamma$.

#### 10.1.1 The Gauss-Seidel and Jacobi Algorithms

Because minimizing symmetric positive definite quadratic functions is equivalent to solving symmetric positive definite linear systems, well-studied algorithms such as Gaussian elimination, Cholesky decomposition, etc. can be used to compute the minimum. In addition, many iterative algorithms have been proposed to solve the linear system $\Gamma x = h$: Gauss-Seidel iteration, Jacobi iteration, the algebraic reconstruction technique, etc.

In this section, we focus on the Jacobi and Gauss-Seidel algorithms (Algorithms 10 and 11) both because of their simplicity and their similarities to Algorithms 7 and 8 for this problem. When $\Gamma$ is symmetric positive definite, the objective function, $\frac{1}{2} x^T \Gamma x - h^T x$, is a convex function of $x$. Consequently, we could use a coordinate descent method in an attempt to minimize the objective function. The standard cyclic coordinate descent algorithm for this problem is known as the Gauss-Seidel algorithm.

In the same way that Algorithm 7 is a synchronous version of Algorithm 8, the Jacobi algorithm is a synchronous version of the Gauss-Seidel algorithm. To see this, observe that the iterates
Algorithm 10 Jacobi Iteration
1: Choose an initial vector $x^0 \in \mathbb{R}^n$.
2: For iteration $t = 1, 2, ...$, set
   \[ x_j^t = \frac{h_j - \sum_k \Gamma_{jk} x_{k}^{t-1}}{\Gamma_{jj}} \]
   for each $j \in \{1, ..., n\}$.

Algorithm 11 Gauss-Seidel Iteration
1: Choose an initial vector $x \in \mathbb{R}^n$.
2: Choose some ordering of the variables, and perform the following update for each variable $j$, in order:
   \[ x_j = \frac{h_j - \sum_k \Gamma_{jk} x_k}{\Gamma_{jj}} \]

Produced by the Jacobi algorithm are related to the iterates of the Gauss-Seidel algorithm on a larger problem. Specifically, given a symmetric $\Gamma \in \mathbb{R}^{n \times n}$ and $h \in \mathbb{R}^n$, construct $\Gamma' \in \mathbb{R}^{2n \times 2n}$ and $h' \in \mathbb{R}^{2n}$ as follows:
\begin{align*}
    h'_i &= h_{\lceil h_i / n \rceil} \\
    \Gamma' &= \begin{bmatrix} D & M \\ M & D \end{bmatrix}
\end{align*}

where $D$ is a diagonal matrix with the same diagonal entries as $\Gamma$ and $M = \Gamma - D$.

$\Gamma'$ is the analog of the Kronecker double cover discussed in Section 8.1. Let $x^0 \in \mathbb{R}^n$ be an initial vector for the Jacobi algorithm performed on the matrix $\Gamma$ and fix $y^0 \in \mathbb{R}^{2n}$ such that $y^0_i = x_{1+(i-1 \mod n)}$. Further, suppose that we update the variables in the order $1, 2, ..., 2n$ in the Gauss-Seidel algorithm. If $y^t$ is the vector produced after $t$ complete cycles of the Gauss-Seidel algorithm, then $y^t = \begin{bmatrix} x_{2t-1}^t \\ x_{2t}^t \end{bmatrix}$. Also, observe that, for any $y^t$ such that $\Gamma' y^t = h'$, we must have that $\Gamma x_{2t-1}^t + x_{2t}^t = h$.

With these two observations, any convergence result for the Gauss-Seidel algorithm can be extended to the Jacobi algorithm. Consider the following:

**Theorem 10.1.1.** Let $\Gamma$ be a symmetric positive semidefinite matrix with a strictly positive diagonal. The Gauss-Seidel algorithm converges to a vector $x^*$ such that $\Gamma x^* = h$ whenever such a vector exists.

**Proof.** See Section 10.5.1 of [10].
Using our observations, we can immediately produce the following new result:

**Corollary 10.1.2.** Let $\Gamma$ be a symmetric positive semidefinite matrix with positive diagonal and let $\Gamma'$ be constructed as above. If $\Gamma'$ is a symmetric positive semidefinite matrix and there exists an $x^*$ such that $\Gamma x^* = h$, then the sequence $\frac{x^{t+1} - x^t}{2}$ converges to $x^*$ where $x^t$ is the $t^{th}$ iterate of the Jacobi algorithm.

### 10.1.2 Minimization via the Splitting Algorithm

Our goal in this section is to study the properties of the splitting algorithm when it is used to minimize quadratic functions. Every quadratic function admits a pairwise factorization:

$$f(x_1, ..., x_n) = \frac{1}{2} x^T \Gamma x - h^T x$$

where $\Gamma \in \mathbb{R}^{n \times n}$ is a symmetric matrix. The message updates corresponding to this objective function are then given by Algorithms 7 and 8. We note that we abusively write min even though the appropriate notion of minimization for the real numbers is inf.

Because the minimization is being performed over quadratic functions, we can explicitly compute the minimization required by the splitting algorithm at each time step. In this way, the synchronous message update $m^t_{i \rightarrow j}$ can be parameterized as a quadratic function of the form $a^t_{ij} x_j^2 + b^t_{ij} x_j$ where the constants at time $t$ are given by:

$$a^t_{ij} = -\frac{1}{2} \left( \frac{\Gamma_{ij}}{\Gamma_{ii}} \right)^2$$

$$b^t_{ij} = \frac{\left( h_i - \sum_{k \in \partial i \setminus j} c_{ki} \cdot a^t_{ki} \cdot (c_{ji} - 1) a^t_{ji}^{-1} \right) \Gamma_{ij} + 2 \sum_{k \in \partial i \setminus j} c_{ki} \cdot a^t_{ki}^{-1} + 2(c_{ji} - 1) a^t_{ji}^{-1}}{\Gamma_{ii} + 2 \sum_{k \in \partial i \setminus j} c_{ki} \cdot a^t_{ki}^{-1} + 2(c_{ji} - 1) a^t_{ji}^{-1}}$$

These updates are only valid when $\Gamma_{ii} + 2 \sum_{k \in \partial i \setminus j} a^t_{ki}^{-1} + 2(c_{ji} - 1) a^t_{ji}^{-1} > 0$. If this is not the case, then the minimization given in Algorithm 7 is undefined, and we set $a^t_{ij} = b^t_{ij} = -\infty$. For the initial messages, we set $a^0_{ij} = b^0_{ij} = 0$. A similar analysis holds for the asynchronous updates.

From Corollary 5.1.2, we know that, for convex differentiable objective functions, if $c_{ij} \neq 0$ for all $i$ and $j$ and the pairwise splitting algorithm converges to a vector of beliefs that are locally...
decodable to \( x^* \), then \( x^* \) minimizes the objective function.

### Graph Covers

As discussed in Chapter 7, graph covers provide valuable insight into the behavior of message passing algorithms for objective functions over finite state spaces. For continuous state spaces, graph covers continue to provide valuable insight into the behavior of the splitting algorithm. Let \( G \) be the pairwise factor graph for the objective function \( f^G(x_1, \ldots, x_n) = \frac{1}{2} x^T \Gamma x - h^T x \). Let \( H \) be a \( k \)-cover of \( G \) with a corresponding objective function \( f^H(x_{11}, \ldots, x_{1k}, \ldots, x_{nk}) = \frac{1}{2} x^T \tilde{\Gamma} x - \tilde{h}^T x \).

Without loss of generality we can assume that \( f^H \) can be written as

\[
\tilde{\Gamma} = \begin{pmatrix}
\Gamma_{11} P_{11} & \cdots & \Gamma_{1n} P_{1n} \\
\vdots & \ddots & \vdots \\
\Gamma_{n1} P_{n1} & \cdots & \Gamma_{nn} P_{nn}
\end{pmatrix} \tag{10.10}
\]

\[
\tilde{h}_i = h_{[i/k]} \tag{10.11}
\]

where, \( P_{ij} \) is a \( k \times k \) permutation matrix for all \( i \neq j \) and \( P_{ii} \) is the \( k \times k \) identity matrix for all \( i \). The graph covers corresponding to the pairwise factorization are also graph covers for the Gauss-Seidel and Jacobi algorithms.

#### Definition 10.1.1.

Let \( \Gamma_G \) be the matrix corresponding to the objective function \( f^G \) with factor graph \( G \). We say that \( \Gamma_H \) covers \( \Gamma_G \) if \( H \) covers \( G \) and \( \Gamma_H \) is the matrix corresponding to the objective function \( f^H \).

For the quadratic minimization problem, factor graphs and their covers share many of the same properties. Most notably, we can transform critical points on covers to critical points of the original problem. Let \( H \) and \( G \) be as above. We have the following lemma:

**Lemma 10.1.3.** Suppose \( \tilde{\Gamma} x' = \tilde{h} \) for \( x' \in \mathbb{R}^{nk} \). If \( x \in \mathbb{R}^n \) is given by \( x_i = \frac{1}{k} \sum_{j=1}^{k} x'_{ki+j} \) then \( \Gamma x = h \). Conversely, suppose \( \Gamma x = h \). If \( x' \) is given by \( x'_i = x_{[i/k]} \) then \( \tilde{\Gamma} x' = \tilde{h} \).

Notice that these solutions correspond to critical points of the cover and the original problem. Similarly, we can transform eigenvectors of covers to either eigenvectors of the original problem or the zero vector.

**Lemma 10.1.4.** Suppose \( \tilde{\Gamma} x' = \lambda x' \). If \( x \in \mathbb{R}^n \) is given by \( x_i = \frac{1}{k} \sum_{j=1}^{k} x'_{ki+j} \) then either \( \Gamma x = \lambda x \) or \( \Gamma x = 0 \). Conversely, suppose \( \Gamma x = \lambda x \). If \( x' \) is given by \( x'_i = x_{[i/k]} \) then \( \tilde{\Gamma} x' = \lambda x' \).
\[ \Gamma = \begin{pmatrix} 1 & .6 & .6 \\ .6 & 1 & .6 \\ .6 & .6 & 1 \end{pmatrix} \quad \tilde{\Gamma} = \begin{pmatrix} 1 & 0 & .6 & 0 & 0 & .6 \\ 0 & 1 & 0 & .6 & 0 & .6 \\ .6 & 0 & 1 & 0 & .6 & 0 \\ 0 & .6 & 0 & 1 & 0 & .6 \\ .6 & .6 & 0 & 1 & 0 & .6 \\ .6 & 0 & 0 & .6 & 0 & 1 \end{pmatrix} \]

Figure 10.1: An example of a positive definite matrix, \( \Gamma \), which possesses a 2-cover, \( \tilde{\Gamma} \), that has negative eigenvalues.

These lemmas demonstrate that we can average critical points and eigenvectors of covers to critical points and eigenvectors (or the zero vector) of the original problem, and we can lift critical points and eigenvectors of the original problem in order to obtain critical points and eigenvectors of covers.

**Scaled Diagonally Dominant Models**

Unfortunately, even though the critical points of \( \Gamma \) and its covers must correspond via Lemma 10.1.3, the corresponding minimization problems may not have the same solution. The example in Figure 10.1 illustrates that there exist positive definite matrices that are covered by matrices which are not positive definite. This observation seems to be problematic for the convergence of the splitting algorithm. The messages passed by the splitting algorithm are exactly the same for each variable of \( \Gamma \) and each of their copies in \( \tilde{\Gamma} \). As such, the splitting algorithm may not converge to the correct minimizing assignment unless \( \Gamma \) and all of its covers are positive definite. We can exactly characterize the matrices for which this property holds. Consider the following definitions:

**Definition 10.1.2.** \( \Gamma \in \mathbb{R}^{n \times n} \) is walk summable if the spectral radius \( \rho(I - D^{-1/2}\Gamma D^{-1/2}) < 1 \). Here, \( D^{-1/2} \) is the diagonal matrix such that \( D^{-1/2}_{ii} = \frac{1}{\sqrt{\Gamma_{ii}}} \).

**Definition 10.1.3.** \( \Gamma \in \mathbb{R}^{n \times n} \) is scaled diagonally dominant if \( \exists w > 0 \in \mathbb{R}^n \) such that \( |\Gamma_{ii}|w_i > \sum_{j \neq i} |\Gamma_{ij}|w_j \).

Both of these properties are sufficient conditions for the convergence and correctness of the related Gauss-Seidel and Jacobi algorithms (see Section 10.1.1). Surprisingly, they are precisely the conditions that ensure that \( \Gamma \) and all of its covers are positive definite.

**Theorem 10.1.5.** Let \( \Gamma \) be a symmetric matrix with positive diagonal. The following are equivalent:

1. \( \Gamma \) is walk summable.
2. $\Gamma$ is scaled diagonally dominant.

3. All covers of $\Gamma$ are positive definite.

4. All 2-covers of $\Gamma$ are positive definite.

Proof. Without loss of generality, we can assume that $\Gamma$ has a unit diagonal. We break the proof into several pieces:

- (1 $\Rightarrow$ 2) Let $\lambda$ be an eigenvalue of $|I - \Gamma|$ with eigenvector $x > 0$ whose existence is guaranteed by the Perron-Frobenius theorem since without loss of generality we can assume that $|I - \Gamma|$ is irreducible (if not we can make this argument on each of its connected components). For any row $i$, we have:

$$x_i > \lambda x_i = \sum_{j \neq i} |\Gamma_{ij}| x_j$$

Since $\Gamma_{ii} = 1$ this is the definition of scaled diagonal dominance with $w = x$.

- (2 $\Rightarrow$ 3) If $\Gamma$ is scaled diagonally dominant then so is every one of its covers. Scaled diagonal dominance implies that a matrix is symmetric positive definite. Therefore, all covers must be symmetric positive definite.

- (3 $\Rightarrow$ 4) Trivial.

- (4 $\Rightarrow$ 1) Let $\tilde{\Gamma}$ be any 2-cover of $\Gamma$. Without loss of generality, we can assume that $\tilde{\Gamma}$ has the form of equation 10.10.

First, observe that by the Perron-Frobenius theorem there exists an eigenvector $x > 0 \in \mathbb{R}^n$ of $|I - \Gamma|$ with eigenvalue $\varrho(|I - \Gamma|)$. Let $y \in \mathbb{R}^{2n}$ be constructed by duplicating the values of $x$ so that $y_{2i} = y_{2i+1} = x_i$ for each $i \in \{0...n\}$. By Lemma 10.1.4, $y$ is an eigenvector of $|I - \tilde{\Gamma}|$ with eigenvalue equal to $\varrho(|I - \Gamma|)$. We claim that this implies $\varrho(|I - \Gamma|) = \varrho(|I - \tilde{\Gamma}|)$. Assume without loss of generality that $|I - \tilde{\Gamma}|$ is irreducible; if not, then we can apply the following argument to each connected component of $|I - \tilde{\Gamma}|$. By the Perron-Frobenius theorem again, $|I - \tilde{\Gamma}|$ has a unique positive eigenvector (up to scalar multiple), with eigenvalue equal to the spectral radius. Thus, $\varrho(|I - \Gamma|) = \varrho(|I - \tilde{\Gamma}|)$ because $y > 0$.

We will now construct a specific cover $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ is positive definite if and only if $\Gamma$ is walk summable. To do this, we’ll choose the $P_{ij}$ as in equation 10.10 such that $P_{ij} = I$ if
\( \Gamma_{ij} < 0 \) and \( P_{ij} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \) otherwise. Now define \( z \in \mathbb{R}^{2n} \) by setting \( z_i = (-1)^i c y_i \), where the constant \( c \) ensures that \( \|z\| = 1 \).

Consider the following:

\[
\begin{align*}
z^T \tilde{\Gamma} z &= \sum_i \Gamma_{ii} z_i^2 + \sum_{i=1}^n \sum_{j \neq i} \Gamma_{ij} [z_{2i}, z_{2i+1}] P_{ij} \begin{bmatrix} z_{2j} \\ z_{2j+1} \end{bmatrix} \\
&= 1 - 2 \sum_{i>j} |\Gamma_{ij}| c^2 y_i y_j
\end{align*}
\]

Recall that \( y \) is the eigenvector of \( |I - \tilde{\Gamma}| \) corresponding to the largest eigenvalue and \( \|cy\| = 1 \).

By definition and the above,

\[
\varrho(|I - \Gamma|) = \varrho(|I - \tilde{\Gamma}|) = \frac{cy^T |I - \tilde{\Gamma}| cy}{c^2 y^T y} = 2 \sum_{i>j} |\Gamma_{ij}| c^2 y_i y_j
\]

Combining all of the above we see that \( z^T \tilde{\Gamma} z = 1 - \varrho(|I - \Gamma|) \). Now, \( \tilde{\Gamma} \) positive definite implies that \( z^T \tilde{\Gamma} z > 0 \), so \( 1 - \varrho(|I - \Gamma|) > 0 \). In other words, \( \Gamma \) is walk summable.

This theorem has two important consequences. First, it provides us with a new characterization of scaled diagonal dominance and walk-summability. Second, it provides an intuitive explanation for why these conditions should be sufficient for the convergence of the splitting algorithm, the Gauss-Seidel algorithm, and the Jacobi algorithm. Indeed, walk summability and scaled diagonal dominance were independently shown to be sufficient conditions for the convergence of the min-sum algorithm for positive definite matrices [29, 32].

**Beyond Diagonal Dominance**

Empirically, the splitting algorithm can converge to the correct minimizer of the objective function even if the original matrix is not scaled diagonally dominant. The most significant problem when the original matrix is positive definite but not scaled diagonally dominant is that the computation trees may eventually possess negative eigenvalues. If this happens, then some of the beliefs will not be convex, and the corresponding estimate will be negative infinity. This is, of course, the correct
answer on some 2-cover of the problem, but it is not the correct solution to the minimization problem of interest. Our goal, then, is to provide conditions on the choice of the parameter vector such that all of the computation trees produced by the splitting algorithm remain positive definite.

The convexity of $\tau_i$ is determined only by the $a$ vector. As such, we begin by studying the sequence $a^0, a^1, \ldots$ where $a^0$ is the zero vector. We will divide the choice of the parameter vector into two cases: one in which $c_{ij} \geq 1$ for all $i$ and $j$ and one in which $c_{ij} \leq 0$ for all $i$ and $j$.

**Lemma 10.1.6.** If $c_{ij} \geq 1$ for all $i$ and $j$, then for all $t > 0$, $a_{ij}^t \leq a_{ij}^{t-1} \leq 0$ for each $i$ and $j$.

**Proof.** This result follows by induction. First, suppose that $c_{ij} \geq 1$. If the update is not valid, then $a_{ij}^t = -\infty$ which trivially satisfies the inequality. Otherwise, we have:

$$a_{ij}^t = \frac{1}{2} \left( \frac{\Gamma_{ij}}{c_{ij}} \right)^2 \Gamma_{ii} + 2 \sum_{k \in \partial i - j} c_{ki} a_{ki}^{t-1} + 2(c_{ji} - 1) a_{ji}^{t-1} \leq \frac{1}{2} \left( \frac{\Gamma_{ij}}{c_{ij}} \right)^2 \Gamma_{ii} + 2 \sum_{k \in \partial i - j} c_{ki} a_{ki}^{t-1} + 2(c_{ji} - 1) a_{ji}^{t-1} = a_{ij}^{t-1}$$

where the inequality follows from the observation that $\Gamma_{ii} + 2 \sum_{k \in \partial i - j} c_{ki} a_{ki}^{t-1} + 2(c_{ji} - 1) a_{ji}^{t-1} > 0$ and the induction hypothesis.

If we consider only the $a$ vector, then the algorithm may exhibit a weaker form of convergence:

**Lemma 10.1.7.** If $c_{ij} \geq 1$ for all $i$ and $j$ and all of the computation trees are positive definite, then the sequence $a_{ij}^0, a_{ij}^1, \ldots$ converges.

**Proof.** Suppose $c_{ij} \geq 1$. By Lemma 10.1.6, the $a_{ij}^t$ are monotonically decreasing. Because all of the computation trees are positive definite, we must have that for each $i$, $\Gamma_{ii} + 2 \sum_{k \in \partial i - j} c_{ki} a_{ki}^{t-1} + 2c_{ji} a_{ji}^{t-1} > 0$. Hence, the sequence $a_{ij}^0, a_{ij}^1, \ldots$ is monotonically decreasing and bounded from below by $-\frac{1}{2c_{ij}}$. This implies that it converges.

Because the estimates of the variances only depend on the $a_{ij}^t$, if the $a_{ij}^t$ converge, then the estimates of the variances also converge. Therefore, requiring all of the computation trees to be positive definite is a sufficient condition for convergence of the variances. Note, however, that the
Figure 10.2: A positive definite matrix for which the variances in the min-sum algorithm converge but the means do not. [28]

estimates of the means which correspond to the sequence $b^i_{ij}$ need not converge even if all of the computation trees are positive definite (see Figure 10.2).

For now, our strategy will be to ensure that all of the computation trees are positive definite by leveraging our choice of parameters in the splitting algorithm. Specifically, we want to use these parameters to weight the diagonal elements of the computation tree much more than the off-diagonal elements in order to force the computation trees to be positive definite. If we can show that there is a choice of each $c_{ij} = c_{ji}$ that will cause all of the computation trees to be positive definite, then the splitting algorithm over the new factorization will behave almost as if the original matrix were scaled diagonally dominant.

**Theorem 10.1.8.** For any symmetric matrix $\Gamma$ with strictly positive diagonal, $\exists r \geq 1$ and an $\epsilon > 0$ such that the eigenvalues of the computation trees are bounded from below by $\epsilon$ when generated by the pairwise splitting algorithm with $c_{ij} = r$ for all $i$ and $j$.

**Proof.** Let $T_v(t)$ be the depth $t$ computation tree rooted at $v$, and let $\Gamma'$ be the matrix corresponding to $T_v(t)$ (i.e. the matrix generated by the potentials in the computation tree). We will show that the eigenvalues of $\Gamma'$ are bounded from below by some $\epsilon > 0$. For any $i \in T_v(t)$ at depth $d$ define:

$$w_i = \left(\frac{s}{r}\right)^d$$

where $r$ is as in the statement of the theorem and $s$ is a positive real to be determined below. Let $W$ be a diagonal matrix whose entries are given by the vector $w$. By the Geršgorin disc theorem [17], all of the eigenvalues of $W^{-1}\Gamma'W$ are contained in

$$\bigcup_{i \in T_v(t)} \left\{ z \in \mathbb{R} : |z - \Gamma'_{ii}| \leq \frac{1}{w_i} \sum_{j \neq i} w_j |\Gamma'_{ij}| \right\}$$

Because all of the eigenvalues are contained in these discs, we need to show that there is a choice of $s$ and $r$ such that for all $i \in T_v(t)$, $|\Gamma'_{ii} - \frac{1}{w_i} \sum_{j \neq i} w_j |\Gamma'_{ij}| \geq \epsilon$.
Recall from Section 6.1.1 that $|\Gamma'_{ij}| = \eta |\Gamma_{ij}| / r$ for some constant $\eta$ that depends on $r$. Further, all potentials below the potential on the edge $i, j$ are multiplied by $\eta \gamma$ for some constant $\gamma$. We can divide out by this common constant to obtain equations that depend on $r$ and the elements of $\Gamma$. Note that some self-potentials will be multiplied by $r - 1$ while others will be multiplied by $r$. With this rewriting, there are three possibilities:

1. $i$ is a leaf of $T_v(t)$. In this case, we need $|\Gamma_{ii}| > \frac{1}{w_i} |\Gamma_{ip(i)}| w_p(i)$. Plugging in the definition of $w_i$, we have

$$|\Gamma_{ii}| > \frac{|\Gamma_{ip(i)}|}{s}$$  \hfill (10.13)

2. $i$ is not a leaf of $T_v(t)$ or the root. In this case, we need

$$|\Gamma_{ii}| > \frac{1}{w_i} \left[ \frac{|\Gamma_{ip(i)}|}{r} w_p(i) + \frac{s^2(r-1)}{r^3} |\Gamma_{ip(i)}| w_p(i) + \sum_{k \in \partial i - p(i)} |\Gamma_{ki}| w_k \right]$$  \hfill (10.14)

Again, plugging the definition of $w_i$ into the above yields

$$|\Gamma_{ii}| > \frac{|\Gamma_{ip(i)}|}{s} + \frac{s}{r} \left[ \frac{r-1}{r} |\Gamma_{ip(i)}| + \sum_{k \in \partial i - p(i)} |\Gamma_{ki}| \right]$$  \hfill (10.15)

3. $i$ the root of $T_v(t)$. Similar to the previous case, we need $|\Gamma_{ii}| w_i > \sum_{k \in \partial i} |\Gamma_{ki}| w_k$. Again, plugging the definition of $w_i$ into the above yields

$$|\Gamma_{ii}| > \frac{s}{r} \sum_{k \in \partial i} |\Gamma_{ki}|$$  \hfill (10.16)

None of these bounds are time dependent. As such, if we choose $s$ and $r$ to satisfy the above constraints, then there must exist some $\epsilon > 0$ such that smallest eigenvalue of any computation tree is at least $\epsilon$. Fix $s$ to satisfy Equation 10.13 for all leaves of $T_v(t)$. This implies that $(|\Gamma_{ii}| - |\Gamma_{ip(i)}| / s) > 0$ for any $i \in T_v(t)$. Finally, we can choose a sufficiently large $r$ that satisfies the remaining two cases for all $i \in T_v(t)$.

For the case in which $c_{ij} < 0$ for all $i$ and $j$, we have that the computation trees are always positive definite when the initial messages are uniformly equal to zero and the result is much simpler:

105
Lemma 10.1.9. If \( c_{ij} < 0 \) for all \( i \) and \( j \), then for all \( t > 0 \), \( a^t_{ij} \leq 0 \).

Proof. This result follows by induction. First, suppose that \( c_{ij} \geq 0 \). If the update is not valid, then \( a^t_{ij} = -\infty \) which trivially satisfies the inequality. Otherwise, we have:

\[
a^t_{ij} = \frac{1}{2} \left( \frac{\Gamma_{ij}}{c_{ij}} \right)^{2} \left( \sum_{k \in \partial i} c_{k_i} a^t_{k_i} - 2 c_{ji} a^t_{ji} \right) \leq 0
\]

where the inequality follows from the induction hypothesis.

\[\square\]

Theorem 10.1.10. For any symmetric matrix \( \Gamma \) with strictly positive diagonal, if \( c_{ij} < 0 \) for all \( i \) and \( j \), then all of the computation trees are positive definite.

Proof. The computation trees are all positive definite if and only if \( \Gamma_{ii} + 2 \sum_{k \in \partial i} c_{k_i} a^t_{k_i} \geq 0 \) for all \( t \). By Lemma 10.1.9, \( a^t_{ij} \leq 0 \) for all \( t \), and as result, \( \Gamma_{ii} + 2 \sum_{k \in \partial i} c_{k_i} a^t_{k_i} \geq \Gamma_{ii} > 0 \) for all \( t \).

If all of the computation trees remain positive definite, then the beliefs are all positive definite. Hence, if the splitting algorithm converges to a fixed point, then the beliefs must be locally decodable to the correct minimizing assignment. Notice that neither of these theorems require \( \Gamma \) to be positive definite. Indeed, we have already seen an example of a matrix with a strictly positive diagonal and negative eigenvalues (see the matrix in Figure 10.1).

Convergence of the Means

If the variances converge, then the fixed points of the message updates for the means correspond to the solution of a particular linear system \( Mb = d \). In fact, we can show that Algorithm 8 is exactly the Gauss-Seidel algorithm for this linear system. First, we construct the matrix \( M \in \mathbb{R}^{|A| \times |A|} \):

\[
M_{ij,i} = \left[ \Gamma_{ii} + \sum_{k \in \partial i} 2 c_{k_i} a^*_k - 2 a^*_j \right] \text{ for all } i \text{ and } j \in \partial i
\]

\[
M_{ij,k} = \frac{\Gamma_{ij}}{c_{ij}} \text{ for all } i, j, k \in \partial i \text{ such that } k \neq j
\]

\[
M_{ij,j} = \frac{\Gamma_{ij}}{c_{ij}} (c_{ij} - 1) \text{ for all } i \text{ and } j \in \partial i
\]

Here, \( a^* \) is the vector of converged variances. All other entries of the matrix are equal to zero. Next, we define the vector \( d \in \mathbb{R}^{|A|} \) by setting \( d_{ij} = h_i \) for all \( i \) and \( j \in \partial i \).
By definition, any fixed point, \( b^* \), of the message update equations for the means must satisfy \( Mb^* = d \). With these definitions, Algorithm 8 is precisely the Gauss-Seidel algorithm for this matrix. Similarly, Algorithm 7 corresponds to the Jacobi algorithm. Unfortunately, \( M \) is neither symmetric nor diagonally dominant, so the standard results for the convergence of the Gauss-Seidel algorithm do not necessarily apply to this situation. In practice, we have observed that the splitting algorithm converges if each \( c_{ij} \) is sufficiently large (or sufficiently negative). In the next section, we present empirical evidence that the splitting algorithm outperforms the min-sum algorithm on this problem.

**Experimental Results**

Let \( \Gamma \) be the graph below:

\[
\begin{pmatrix}
1 & r & -r & -r \\
r & 1 & -r & 0 \\
-r & -r & 1 & -r \\
-r & 0 & -r & 1
\end{pmatrix}
\]  

(10.17)

Without splitting, the min-sum algorithm converges to the correct solution for \( 0 < r < 0.39865 [29] \). Figure 10.3 illustrates the behavior of the min-sum algorithm, the asynchronous splitting algorithm with \( c_{ij} = 2 \) for all \( i \neq j \), and the synchronous splitting algorithm with \( c_{ij} = 2 \) for all \( i \neq j \).
for different choices of the constant $r$. If we set $c_{ij} = 3$ for all $i \neq j$, then empirically, both the synchronous and asynchronous splitting algorithms converge for all $r \in (-.5, .5)$, which is the entire positive definite region for this matrix. The rate of convergence for all of the algorithms seems to be dominated by the condition number of the matrix. However, judicious choice of the parameter vector can decrease the number of iterations required for convergence.

Although both the synchronous and asynchronous splitting algorithms converge for the entire positive definite region in the above example, damping may be required in order to force the synchronous algorithm to converge over arbitrary graphs, even for sufficiently large $c$. Recall that Algorithm 2 is equivalent to a particular asynchronous update on the Kronecker double cover (see Section 8.1). If the Kronecker double cover is not positive definite, then the synchronous algorithm may not converge. The damped min-sum algorithm with damping factor $\delta = 1/2$ empirically seems to converge if $\Gamma$ is positive definite and all of the computation trees remain positive definite [29]. We make the same observation for the damped version of Algorithm 2.

To see why damping might result in convergence, recall the relationship between the synchronous and asynchronous algorithms described in Section 8.1. A similar relationship holds when $\delta = 1/2$. However, unlike before, the corresponding vector of messages on the Kronecker double cover at time $t$ is the lift of the messages generated by the damped splitting algorithm at time $t$. Consequently, the time $t$ estimates produced by the asynchronous algorithm on the 2-cover must also correspond to a lift of the time $t$ estimates produced by the damped synchronous splitting algorithm. As a result, the asynchronous algorithm cannot reduce the value of the objective function on the 2-cover below the minimum of the original problem (which may have been possible if the Kronecker double cover was not positive semidefinite).

In practice, the damped synchronous splitting algorithm with $\delta = 1/2$ and the asynchronous splitting algorithm appear to converge for all sufficiently large choices of the parameter vector as long as $\Gamma$ is positive definite. We conjecture that this is indeed the case: for all positive definite $\Gamma$ there exists a $c$ such that if $c_{ij} = c$ for all $i \neq j$, then the asynchronous splitting algorithm always converges.

### 10.2 General Convex Optimization

The results of Section 10.1 can, in theory, be applied to minimize general convex functions, but in practice, computing and storing the message vector may be inefficient. Despite this, many of
the previous observations can be extended to general convex functions. In this section, we assume that the objective function, \( f^G \), is an arbitrary real-valued convex function, \( f : C \to \mathcal{R} \) such that \( C \subseteq \mathcal{R}^n \) is a convex set.

As was the case for quadratic minimization, convexity of the objective function \( f^G \) does not necessarily guarantee convexity of the objective function \( f^H \) for every finite cover \( H \) of \( G \). Recall that the existence of graph covers that are not bounded from below can be problematic for the splitting algorithm. For quadratic functions, this cannot occur if the matrix is scaled diagonally dominant or, equivalently, if the objective function corresponding to every finite graph cover is positive definite. This equivalence suggests a generalization of scaled diagonal dominance for arbitrary convex functions.

Following the work of [19] and [28] for quadratic objective functions, we propose a condition on the factorization that is sufficient to guarantee the convexity of the objective function on all finite graph covers:

**Definition 10.2.1.** \( f \) is **convex decomposable** over \( A \) if there exists a factorization of \( f \) over \( A \) such that each of the potential functions in the factorization is convex.

If the objective function is quadratic and the factorization is pairwise, convex decomposability can be shown to be equivalent to scaled diagonal dominance (see Proposition 3.1.8 of [28] where pairwise-normalizable corresponds to our definition of pairwise convex decomposable). For general objective functions, convex decomposability is sufficient to guarantee that all graph covers will be positive definite:

**Lemma 10.2.1.** If \( f \) is convex decomposable over \( A \) with corresponding factor graph \( G \), then for every finite cover \( H \) of \( G \), \( f^H \) is convex.

**Proof.** Since \( f \) is convex decomposable, the objective function \( f^H \) corresponding to a finite cover \( H \) of \( G \) is convex decomposable. As convex decomposability implies convexity, we must have that \( f^H \) is convex for every finite cover \( H \) of \( G \).

Further, if \( c_\alpha \geq 1 \) for all \( \alpha \) and \( c_i = 1 \) for all \( i \), then every computation tree produced by the splitting algorithm for such a factorization is also convex. Consequently, the message updates are well-defined at each time step. As in the quadratic case, this alone is not necessarily enough to guarantee convergence of the splitting algorithm (see Figure 10.2).
Now, if $f$ is convex decomposable over $A$ via the factorization $(\phi', \psi')$, then every assignment on a $k$-cover $H$ of $G$ corresponds to an assignment on $G$. Recall the construction from Lemma 7.4.1:

$$f^H(x^H) = k \left[ \sum_{i \in x^H} \mu_i(x_i) \phi_i(x_i) + \sum_{\alpha \in x^H} \mu_\alpha(x_\alpha) \psi_\alpha(x_\alpha) \right]$$  \hspace{1cm} (10.18)

for some $\mu$ such that for all $i$, $\sum_{x_i} \mu_i(x_i) = 1$ and for all $\alpha$, $\sum_{x_\alpha} \mu_\alpha(x_\alpha) = \mu_i(x_i)$. Let $x^G_i = \sum_{x_i} \mu_i(x_i)$. By the convexity of the decomposition:

$$f^H(x^H) \geq k \left[ \sum_{i \in x^H} \phi_i(\sum_{x_i} \mu_i(x_i) \cdot x_i) + \sum_{\alpha \in x^H} \psi_\alpha(\sum_{x_\alpha} \mu_\alpha(x_\alpha) \cdot x_\alpha) \right]$$  \hspace{1cm} (10.19)

$$= kf^G(x^G)$$  \hspace{1cm} (10.20)

So, minima of graph covers must correspond to minima of the objective function for convex decomposable functions. This continues to hold for convex functions whose graph covers are all convex but are not convex decomposable:

**Lemma 10.2.2.** Let $f$ be a convex function that factorizes over $A$ with corresponding factor graph $G$. Suppose that for every finite cover $H$ of $G$, $f^H$ is convex. If $x^G \in \arg \min_x f(x)$, then for every finite cover $H$ of $G$, $x^H$, the lift of $x^G$ to $H$, minimizes $f^H$.

**Proof.** This follows from the observation that all convex functions are subdifferentiable over their domains, and that $x^H$ is a minimum of $f^H$ iff the zero vector is contained in the subgradient of $f^H$ at $x^H$. \hfill \Box

Even if $f$ is not convex decomposable, we may still be able to use the same trick as in Theorem 10.1.8 in order to force the computation trees to be convex. Let $C \subseteq \mathbb{R}^n$ be a convex set. If $f : C \to \mathbb{R}$ is twice continuously differentiable, then $f$ is convex if and only if its Hessian, the matrix of second partial derivatives, is positive semidefinite on the interior of $C$. The Hessian matrix of a quadratic function, $\frac{1}{2}x^T \Gamma x - h^T x$, is equal to $\Gamma$. For each $x \in C$, Theorem 10.1.8 demonstrates that there exists a choice of the vector $c$ such that all of the computation trees are convex at $x$. For twice continuously differentiable functions, sufficient conditions for the convergence of the min-sum algorithm that are based on a generalization of scaled diagonal dominance are discussed in [32].

110
10.3 Submodular Functions

Lastly, we turn our attention to submodular functions. Submodular functions arise naturally in
the study of matroids, graph cuts, and information theory. Such functions can be minimized in
strongly polynomial time [45, 18]. However, computing the maximum of an arbitrary submodular
function is an NP-hard problem.

**Definition 10.3.1.** Let $S$ be a finite set. A function $f : 2^S \rightarrow \mathbb{R}$ is **submodular** if and only if $f(A \cap B) + f(A \cup B) \leq f(A) + f(B)$ for all $A, B \subseteq S$.

Equivalently, every subset of $A$ of $S$ can be identified with a vector $x^A \in \{0, 1\}^{|S|}$ such that $x^A_i = 1$ if and only if $i \in A$. Minimizing a submodular function can be reformulated as minimizing a convex function. Every submodular function $f$ can be extended to a convex function $\hat{f} : [0, 1]^{|S|} \rightarrow \mathbb{R}$ such that $\hat{f}(y) = f(x)$ for all $y \in \{0, 1\}^{|S|}$. Let $y$ be a vector in $[0, 1]^{|S|}$, and let $P = p|p = y$. If $p_1 > ... > p_k$ are the elements of $P$, then define the sets $A_i = \{j | y_j \geq p_i\}$ for all $i \in \{1, ..., k\}$. Let $\lambda_0 = (1 - p_1)$, $\lambda_k = p_k$, and $\lambda_i = (p_i - p_{i+1})$ for all $i \in \{1, ..., k - 1\}$. The function $\hat{f}(y)$ is defined as follows:

$$\hat{f}(y) = \lambda_0 f(x^\emptyset) + \sum_{i=1}^k \lambda_i f(x^{A_i}) \quad (10.21)$$

Notice that $\sum_{i=0}^k \lambda_i = 1$. In other words, $\hat{f}$ is a convex combination of the values of $f$. $\hat{f}$ is sometimes referred to as the Lovász extension.

**Theorem 10.3.1.** If $f : \{0, 1\}^{|S|} \rightarrow \mathbb{R}$ is submodular if and only if $\hat{f} : [0, 1]^{|S|} \rightarrow \mathbb{R}$ is convex.

**Proof.** See [27].

**Corollary 10.3.2.** If $f : \{0, 1\}^{|S|} \rightarrow \mathbb{R}$ is submodular, then $\min_{x \in \{0, 1\}^{|S|}} f(x) = \min_{y \in [0, 1]^{|S|}} \hat{f}(y)$. Further, if $f$ has a unique minimum, then $\hat{f}$ has a unique minimum.

Now, suppose that $f$ factors over the set $\mathcal{A}$ as $f(x) = \sum_j \phi_j(x_j) + \sum_{a} \psi_{\alpha}(x_a)$. The extension
\( \hat{f} \) factors over \( A \) as well:

\[
\hat{f}(y) = \lambda_0 f(y) + \sum_{i=1}^{k} \lambda_i f(x_i^*)
\]

\[
= \lambda_0 \sum_j \phi_j(y_j) + \sum_\alpha \psi_\alpha(x_\alpha^*) + \sum_{i=1}^{k} \lambda_i \left[ \sum_j \phi_j(x_j^*) + \sum_\alpha \psi_\alpha(x_\alpha^*) \right]
\]

\[
= \sum_j \left[ \lambda_0 \phi_j(y_j) + \sum_\alpha \lambda_i \phi(x_\alpha^*) \right] + \sum_\alpha \left[ \lambda_0 \psi_\alpha(x_\alpha^*) + \sum_{i=1}^{k} \lambda_i \psi_\alpha(x_\alpha^*) \right]
\]

Consequently, if \( \phi_j \) is submodular for all \( j \) and \( \psi_\alpha \) is submodular for all \( \alpha \), then \( \hat{f} \) is convex decomposable over \( A \), and we can apply the results of Section 10.2. Suppose \( f^G \) is submodular decomposable: there exists a factorization of \( f^G \) over \( A \) such that each potential function is submodular. \( f^G \) is submodular, and for any finite cover \( H \) of \( G \), \( f^H \) is submodular. Further, if \( c_\alpha \geq 1 \) for all \( \alpha \) and \( c_i = 1 \) for all \( i \), then the objective function corresponding to every computation tree produced by any of the splitting algorithms is also submodular.

Unlike the convex case, given any vector of admissible and min-consistent beliefs such that \( b_j \) is submodular for all \( j \) and \( b_\alpha \) is submodular for all \( \alpha \), we can always construct an assignment that simultaneously minimizes each of the beliefs, even if the beliefs are not locally decodable.

**Theorem 10.3.3.** Let \( b \) be a vector of admissible and min-consistent beliefs for a submodular function \( f \) with a corresponding weighting vector of non-zero real numbers, \( c \). If for all \( i \), \( b_i \) is submodular and for all \( \alpha \), \( b_\alpha \) is submodular, then the assignments \( x^* \) and \( y^* \), given by

\[
x^*_i = \begin{cases} 
\arg\min b_i & \text{if it is unique} \\
1 & \text{otherwise}
\end{cases}
\]

\[
y^*_i = \begin{cases} 
\arg\min b_i & \text{if it is unique} \\
0 & \text{otherwise}
\end{cases}
\]

for all \( i \), simultaneously minimize each of the beliefs.

**Proof.** Clearly, for all \( i \), \( x^*_i \) and \( y^*_i \) minimize \( b_i \). We need to show that \( x^*_\alpha \) and \( y^*_\alpha \) minimize \( b_\alpha \) for all \( \alpha \). Take any two elements \( x^*_\alpha, x^*_\alpha' \in \arg\min b_\alpha \). Because \( b_\alpha \) is submodular, \( x^*_\alpha \cup y^*_\alpha \) and \( x^*_\alpha \cap y^*_\alpha \) are also elements of the argmin. Let \( x_\alpha = x^*_\alpha \cap \arg\min b_\alpha \) and \( y_\alpha = y^*_\alpha \cup \arg\min b_\alpha \). By submodularity, \( x_\alpha \) and \( y_\alpha \) both minimize \( b_\alpha \). Now, for each \( i \in \alpha \), \( x_i = 1 \) if and only if every minimizing assignment \( x^*_\alpha \) of \( b_\alpha \) has \( x_i' = 1 \). By min-consistency, this occurs only if \( b_i \) has a unique argmin at 1. Consequently, \( x^*_\alpha = x_\alpha \). Similarly, \( y^*_\alpha = y_\alpha \). \( \square \)
Corollary 10.3.4. Let \( b \) be a vector of admissible and min-consistent beliefs for a submodular function \( f \) with a corresponding weighting vector of non-zero real numbers, \( c \), such that \( f \) can be written as a conical combination of the beliefs. If for all \( i, b_i \) is submodular and for all \( \alpha, b_\alpha \) is submodular, then the assignments \( x^* \) and \( y^* \) given in the statement of Theorem 10.3.3 minimize the objective function.

This corollary only applies if the splitting algorithm converges to a collection of submodular beliefs. In general, this does not necessarily happen. However, if \( f \) is submodular and \( f \) admits a pairwise factorization, then each potential function in the factorization must be submodular.

Proposition 10.3.5. If \( f : 2^{|S|} \to \mathbb{R} \) is submodular and \( f \) factors over \( \mathcal{A} \subseteq 2^{\{1,...,|S|\}} \) such that \( |\alpha| \leq 2 \) for all \( \alpha \in \mathcal{A} \), then any factorization \( (\phi, \psi) \in \mathcal{F}_\mathcal{A}(f) \) must have \( \psi_\alpha \) submodular for all \( \alpha \in \mathcal{A} \).

Proof. Fix some \( \{i,j\} \in \mathcal{A} \). Recall that the vectors \( x^{\{i\}} \) and \( x^{\{j\}} \), are indicator vectors for the set \( \{i\} \) and \( \{j\} \) respectively. By the submodularity of \( f \), we have

\[
\begin{align*}
f(x^{\emptyset}) + f(x^{\{i,j\}}) & \leq f(x^{\{i\}}) + f(x^{\{j\}}) \quad (10.26) \\
\psi_\alpha(x^{\emptyset}_\alpha) + \psi_\alpha(x^{\{i,j\}}_\alpha) & \leq \psi_\alpha(x^{\{i\}}_\alpha) + \psi_\alpha(x^{\{j\}}_\alpha) \quad (10.27)
\end{align*}
\]

which proves the submodularity of \( \psi_\alpha \). □

Consequently, for an appropriate choice of the parameter vector, the splitting algorithm can be used to minimize any submodular function that admits a pairwise factorization. Theorem 3 of [22] proves that the TRMP algorithm can always be used to minimize submodular functions that admit pairwise factorizations, but the proof therein exploits the existence of TRMP fixed points instead of the above proposition. An algorithm based on graph cuts can also be used to compute the minimum of a pairwise binary submodular function [23] (see the discussion in Section 8.2.2 for properties of the splitting algorithm over graph cuts).
Bibliography


