Dynamic Programming, Tree-width and Computation on Graphical Models

by
Brian Oliveira Lucena
M.S., Brown University 1998
A.B., Harvard University 1996

Thesis
Submitted in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy
in the Division of Applied Mathematics at Brown University

May 2002

Abstract of "Dynamic Programming, Tree-width and Computation on Graphical Models," by Brian Oliveira Lucena, Ph.D., Brown University, May 2002

Computing on graphical models is a field of diverse interest today, due to the general applicability of these models. This thesis begins by giving background on the generalized Dynamic Programming (DP) method of performing inference computations (Chapter 1). We then (in Chapter 2) demonstrate explicity equivalences between different methods of computation and the importance of a parameter called *tree-width*. We go on to prove a novel method of bounding the tree-width of a graph by using maximum cardinality search. This gives a lower bound on the computational complexity of a graph with respect to standard methods. This bound can be quite weak or quite good. We provide experimental results demostrating both cases. Chapter 3 is concerned with Coarse-to-Fine Dynamic Programming (CFDP), a method which can be faster than the standard methods, but requires special conditions. We prove theorems giving the complexity of CFDP for problems that meet certain criteria. These theoretical results are borne out with applications to specific problems.

This dissertation by Brian Oliveira Lucena is accepted in its present form by
the Division of Applied Mathematics as satisfying the
dissertation requirement for the degree of
Doctor of Philosophy

Date_____ _____

Stuart Geman, Director

Recommended to the Graduate Council

Date_____ _____

David Mumford, Reader

Date_____ _____

Basilis Gidas, Reader

Approved by the Graduate Council

Date_____ _____

Peder J. Estrup

Dean of the Graduate School and Research

The Vita of Brian Oliveira Lucena

Brian Oliveira Lucena was born on January 17, 1975 in Suffern, NY. He attended Spring Valley High School in Spring Valley, NY, and graduated as valedictorian in June 1992. He then entered Harvard University, graduating in 1996 *magna cum laude* in Applied Mathematics / Probability and Statistics. After spending another year in Cambridge, MA doing research and teaching, he entered the Applied Mathematics program at Brown University in September, 1997. He received the Masters degree in Applied Mathematics in May, 1998 and defended this Ph.D. thesis on April 26, 2002.

Acknowledgments

It is an impossible endeavor to thank every person who played a role in my Ph.D. thesis and the long process leading up to its defense, but nonetheless an attempt must be made.

My advisor, Stuart Geman gave me a fantastic combination of guidance and independence from start to finish. His enthusiasm kept me excited throughout the process and he truly served as a role model for me as a scientist, teacher and person.

Various other professors, notably Basilis Gidas, David Mumford and Elie Bienenstock were available for conversations and discussion on a variety of topics, both mathematical and non-mathematical. Laura Leddy, Jean Radican, Roselyn Winterbottom, Trudee Trudell, and the rest of the administrative staff were unfailingly helpful in numerous capacities.

My three roommates at 18 University Avenue each warrant particular mention. Asohan Amarasingham has been a constant presence for me in the 5 years I've spent in Providence, at various times as classmate, roommate, officemate, and above all, as a friend. The seemingly infinite number of conversations on countless subjects we've had in various locations in the Western Hemisphere have been some of the greatest times I've had. Kamran Diba brought the four of us together at 18 University and exposed us to a variety of music, wine, and life theories while making our house a community instead of just a residence. The intensity of his irresolute convictions and his questioning of every facet of life forced me to open my mind to new ideas. Carlos Vicente with his irrepressible energy supplied endless amounts of amusement and entertainment. His kind critiques of my fashion sense made me a better dresser (for a while) and his lust for life never failed to cheer me up.

A couple of fellow students are owed special debts for their direct contributions to this thesis. Matthew Harrison made the original conjecture of the main theorem in Chapter 2, which sent me down a path of discovery which has not quite ended. He also served as my personal reference for questions on MATLAB and Latex. Luis Ortiz provided invaluable references, thoroughly proofread the tree-width results, and helped me see things from "the computer science point of view". Eyal Amir, a postdoc at UC-Berkeley, supplied me with the CYC, HPKB, and CPCS graphs.

I should also thank the countless good friends I've made here, including Phil Weickert, Govind Menon, Joel Middleton, Mickey Inzlicht, Naomi Ball, Rusty Tchernis, Julie Esdale, Stephanie Munson, Andrew Huebner, Danny Trelogan, Sameer Parekh, Nick Costanzino and so many others I can't possibly list them all. You have all made living these 5 years in Providence not only tolerable, but indeed, the happiest in my life to date.

Most importantly, I thank my family. My brother John and sister Anyssa included me in their travels and events, and provided their own "financial aid" to do things I otherwise couldn't afford to do on a graduate student stipend. My parents always encouraged me to go my own way and find my own path. The debt I owe to you can never be repaid.

Contents

List of Figures

Chapter 1

Introduction and Tutorial

1.1 Introduction

The ultimate goal of mathematical modeling is to understand and predict the behavior of systems. In deterministic systems we can more or less predict exactly what will happen. In probabilistic systems, however, there is an inherent uncertainty. Thus our goal is to understand the uncertainty; that is, to learn about the distribution of the variable of interest. This distribution will change and become "less" random as we have more information about the variable. Graphical models such as Markov random fields, Bayes nets, and a host of other variations are attempts to model complex systems of random variables and their interactions, with the goal of understanding and making predictions.

The field of graphical models is one of tremendous research and application today. Dependency graphs arise in fields as diverse as computer vision, speech recognition, expert systems, coding theory and genetics. The reason for this broad range of applications is that these models are very general in their formulation. Using these techniques, we can model any system in which random variables interact and analyze the effects of the variables on one another. Naturally, there are limitations in practice. It may be difficult to create models which accurately reflect reality or on which we can feasibly compute answers. However, between the infinite theoretical possibilities and the trivial problems lie a rich world of tractable and potentially tractable problems.

In practice, there are two main aspects to these models: learning and computation. The learning problem asks: Given some system of random variables in which we are interested, how do we find a mathematical formulation for this system? The formulation we seek is typically a joint probability distribution on all the variables, given in a compact form or factorization. In some cases the distribution may be obvious. Sometimes it is created by a theoretical analysis of the system. Often, it must be learned from data.

Computational aspects involve using the model to get useful answers to questions about the system in which we are interested. The typical questions are to find the most likely configuration of the variables or the marginal distributions of the variables. We would also like to see the effects of evidence on the remaining variables. That is, if we know the values of some variables, we want to compute the effect on the distribution of the remaining variables. There are many standard algorithms for computation which go by different names but are basically equivalent. These include junction-tree propagation, bucket elimination, factor graphs, and generalized dynamic programming.

The limitation of all these methods is that they have a computational complexity which is exponential in a parameter of the graph called *tree-width*. The base of this exponent is the size of the state space. Therefore, on graphs with low tree-width and moderate state space size, these methods are efficient and these problems are effectively solved. The real challenge in computational aspects of graphical models today is how to compute on graphs where the tree-width and/or state space size is sufficiently large to make the standard methods infeasible.

Many important and interesting problems fall into this latter category. Expert systems for medical diagnosis such as the those based on the QMR-DT database are infeasible by standard methods because they have high tree width. Computer vision and image processing models are often based on Markov random fields defined on a lattice. Since an $n \times n$ lattice has tree-width n these problems are typically intractable by our standard methods. Speech recognition algorithms can be posed as an inference problem on a Hidden Markov Model (HMM). An HMM is a type of graphical model which has low tree-width. However, the HMMs powerful enough to be effective in speech recognition often have a

state space which is infeasibly large.

These types of problems can be attacked in several ways. We can approximate our complex models by simpler ones which are amenable to standard methods. We can settle for calculating bounds, instead of exact answers. Finally, we can look for alternate methods of organizing our computation to reduce the complexity. Which approach or combination of approaches proves to be most useful will depend largely on the specifics of the problem at hand. Currently, the state of the art is a collection of methods which are useful in certain situations. These include variational methods, mean-field approximations, and approximations using mixtures of trees. The more "tools" we have at our disposal, the more likely we are to be able to solve a given problem to our satisfaction. Some of these tools may be applicable quite generally, others only in specific problems.

This thesis is composed of three major sections. The first section gives background on the method of generalized dynamic programming. As stated earlier, there are many equivalent methods which go under different names, but can all be viewed as a generalization of the Dynamic Programming principle as originally formulated by Bellman [5]. These methods fall under such headings as "junction-tree propagation" [17], "Bayesian Belief propagation" [19], "factor graphs" [10], "bucket elimination" [9] and "peeling" [7]. While the structure of these methods are usually well-suited to their particular applications, they are all based on the same principle.

We begin with a tutorial of the generalized Dynamic Programming method and show how to perform all of the basic inference computations: marginal distributions, most likely configuration, and expectations, via this method. Generalized Dynamic Programming has the advantage of a simpler, more intuitive framework, and dispenses with the need to actually triangulate the graph.

Chapter 2 concerns notions of computational complexity of these methods. The various algorithms for computing on Markov random fields have the same complexity, and this computational equivalence corresponds to an equivalence of several different properties on graphs. These results are known, but are not found together in the literature in such a concise form. We will introduce the notion of *tree-width* which can be seen as representing the inherent complexity of a graph with respect to our standard computational methods. Finding the tree-width of a graph is not an easy task in general, however. As a result, given a graph it is not always simple to determine the complexity of our inference computations. Much work has been done on computing upper bounds to tree-width, while relatively little has been done on lower bounds. We show that a procedure called *maximum cardinality search* can be used to calculate a lower bound for the tree-width of a graph. This bound is easy to compute, although it is not necessarily a tight lower bound. We go on to use this result to develop more sophisticated methods for computing lower bounds on the tree width of a graph and analyze the instances where the method will and will not yield a good bound. We also apply our methods to various classes of graphs used in practice and interpret the results.

Chapter 3 explores a method called Coarse-to-Fine Dynamic Programming (CFDP) [22]. This is an algorithm which can be applied to finding the most likely configuration of a Markov random field. It requires a certain hierarchical structure in the states of the variables, and the ability to efficiently find an upper bound to a range of state combinations. Even if we are able to apply CFDP, we are not guaranteed it will be any faster than our standard methods. We analyze the performance of CFDP under certain kinds of problems which are discretizations of continuous problems, and thereby give sufficient conditions under which CFDP will be faster than standard DP. In some cases we are able to deter-

mine exactly the asymptotic performance of CFDP. Beyond showing the performance of CFDP for certain classes of problems, these proofs yield general insight into the types of situations where we can expect CFDP to be effective. We then apply CFDP to problems which meet our criteria, and demostrate this computational savings empirically.

1.2 Generalized Dynamic Programming Tutorial

This section is a brief tutorial on how to compute marginal probabilities, the most likely configuration, and expectations on Markov random fields using what we call here the *generalized Dynamic Programming* method. Basically, this is a review of the method given in [12], although the procedure to simultaneously find all marignal distributions is not covered there. It is also quite similar to the "bucket elimination" algorithms given in [9] and similar in spirit to "non-serial" dynamic programming [6].

Suppose we have a joint probability distribution $P(x_1, x_2, \ldots, x_n)$ on n variables and a graph $G = (V, E)$ which represents the structure of this probability distribution in some way. In graphical models, the vertices represent random variables, and we will use interchangeably "X_i", "the vertex which represents X_i", and "vertex i". So $V = \{X_1, X_2, \ldots, X_n\}$ or simply $\{1, 2, \ldots, n\}$. The word "vertex" and "node" will also be used interchangeably. We denote by $\Gamma(X_i)$ the set of nodes which are adjacent to X_i in the graph G. The *degree* of a node is denoted $deg(X_i) = |\Gamma(X_i)|$. A *clique* is a subset of nodes of a graph such that every pair of nodes in the subset is connected by an edge. A *maximal clique* is a clique which is not a subset of any other clique in the graph. We denote by $\mathcal{C}$ the set of all maximal cliques in our graph G. We will assume that all of our X_i take values in the same outcome space S. In practice this is rarely true, but it simplifies the notation and mathematics considerably, while not fundamentally changing the nature of the problem.

The edges of graphical models represent some relationships between the random variables. For example, we could require our probability distribution P to have the following property with respect to our graph G:

$$P(X_i | V \setminus \{X_i\}) = P(X_i | \Gamma(X_i)) \tag{1.2.1.1}$$

This is the Markov random field (MRF) property. More precisely, we say that P is MRF with respect to a graph G if Equation 1.2.1.1 holds. This equation says that X_i is "conditionally independent" of the rest of the nodes (variables) in the graph given the variables that neighbor X_i in the graph. In other words, variables which are not adjacent to X_i in G affect X_i only to the extent that they may affect the neighbors of X_i. So if all the neighbors are known, the other variables are irrelevant to the distribution of X_i. Another way of stating this property is to say that any two non-adjacent variables X_i, X_j are conditionally independent of each other given any set T of variables such that T is an $X_i - X_j$ *separator*.

Definition 1.2.1. A set of vertices T ($X_i, X_j \notin T$) is said to be an X_i-X_j *separator* if any path from X_i to X_j contains some vertex in T.

Another important property some probability distribution may have with respect to a graph is the *Gibbs* property. This says that:

$$P(X_1, X_2, \ldots, X_n) = \prod_{C \in \mathcal{C}} f_C(X_C) \tag{1.2.1.2}$$

This means that the probability distribution can be written as a product of terms, where each term depends only on the variables involved in that clique.

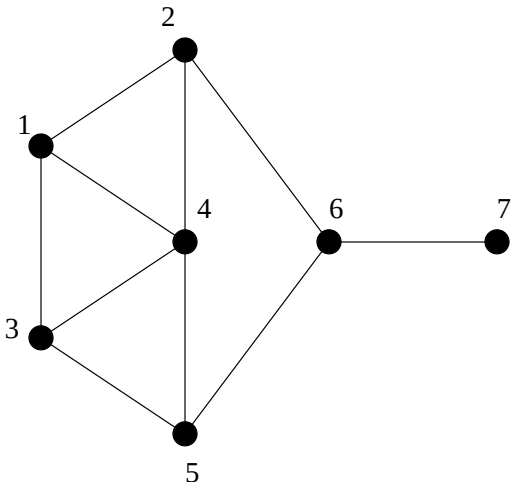

Figure 1.1: A graph G.

By the Hammersley-Clifford theorem [13], we have that if $P > 0$ then P is MRF with respect to G if and only if P is Gibbs with respect to G. For a general function f, we will say that f "respects" the graph G if f can be expressed as a product (or sum) of terms which correspond to the maximal cliques in the graph G.

1.2.1 Computing the Most Likely Configuration

If $P > 0$ then, given a Markov random field, we can invoke the Hammersley-Clifford Theorem and write P as a product of terms, one for each clique in the graph, where the terms for each clique involves only the variables in that clique. Suppose now we want to find the most likely configuration (MLC) of the variables. That is we want to find:

$$\arg \max_{x_1,x_2,\dots,x_n} P(X_1, X_2, \dots, X_n) = \arg \max_{x_1,x_2,\dots,x_n} \prod_{C \in \mathcal{C}} f_C(X_C) \tag{1.2.1.3}$$

We could do this in a brute force manner by evaluating the probability of each of the $|S|^n$ possible state combinations, and choosing the best one. However, the conditional independence structure of the graph enables us to organize our computations in a manner such that we need not examine every single state combination, yet can still ensure that we have the best possible state combination. This is the essence of dynamic programming.

Suppose we are given the graph G in Figure 1.1. We have $n = 7$ random variables and the the conditional independence structure given by G. Specifically we can represent our probability distribution P as a product of clique terms in the following way:

$$P(X_1 = x_1, X_2 = x_2, \dots, X_7 = x_7) = f_{124}(x_1, x_2, x_4) f_{134}(x_1, x_3, x_4) f_{345}(x_3, x_4, x_5) \\ f_{26}(x_2, x_6) f_{56}(x_5, x_6) f_{67}(x_6, x_7)$$

Note that this representation is not unique. For example, we could multiply one clique terms by some constant C and divide another by the same C and we would have a different decomposition.

To use generalized dynamic programming to find the most likely configuration, we must choose an ordering π of the vertices of the graph. Technically, an *ordering* is defined as a bijection:

$$\pi : V \to \{1, 2, \ldots, n\} \tag{1.2.1.4}$$

however, we will often just state $\pi = (v_1, v_2, \ldots, v_n)$ to mean "Label the vertices such that $v_i = \pi^{-1}(i)$". We can then refer to v_i as "the i-th element of the ordering π". The orderings used in dynamic programming are sometimes referred to as *site visitation schedules*.

Some orderings require more computations then others. The complexity depends on the size of the *maximum border* encountered as we progress through the graph. We now give a precise definition of *border* and *maximum border*.

Definition 1.2.2. Let π be an ordering of the vertices of G and let $v_i = \pi^{-1}(i)$. We define the *border* of ordering π at stage i with respect to the graph G as:

$$\begin{aligned}\beta_{\pi,i}(G) = \quad & \{v_j : i{+}1 \leq j \leq n;\ \text{there is a path from } v_i \text{ to } v_j \text{ in } G \\ & \qquad \text{involving only } v_j, v_i, \text{ and } v_1, v_2, \ldots, v_{i-1}\}\end{aligned}$$

Definition 1.2.3. Let the *border size* of an ordering π at stage i with respect to the graph G be given by $B_{\pi,i}(G) = |\beta_{\pi,i}(G)|$.

Definition 1.2.4. Let the *maximum border size* of an ordering π with respect to a graph G be given by $MB_\pi(G) = \max_i B_{\pi,i}(G)$.

The definition for the border is a bit unwieldy, but necessarily so. One way to understand the definition is to consider that the ordering π represents an order of "processing" our vertices. In some sense, we want to define a subset of vertices that come after v_i in the ordering π (i.e. "unprocessed" vertices) which separate the "processed" vertices (i.e. the vertices that come before v_i in the ordering π) from the rest of the unprocessed vertices. However, in reality we don't need a subset to separate *all* of the processed vertices from the unprocessed vertices. We just need to separate the subset of processed vertices which are in the same connected component as v_i from the unprocessed vertices. The above definition must be somewhat complicated to reflect this distinction. Often, our ordering is such that $\{v_1, v_2, \ldots, v_j\}$ is a connected component for all j. In this case the distinction is unnecessary. However, in some situations we will want to choose orderings where this is not the case.

To illustrate this more clearly we define:

$$\begin{aligned}D_{\pi,i}(G) = \quad & \{v_j : 1 \leq j \leq i;\ \text{there is a path from } v_i \text{ to } v_j \text{ in } G \\ & \qquad \text{involving only } v_1, v_2, \ldots, v_i\}\end{aligned}$$

So, $D_{\pi,i}(G)$ is precisely the connected component containing v_i in the subgraph of G generated by the subset $\{v_1, v_2, \ldots, v_i\}$. The border $\beta_{\pi,i}(G)$ is chosen to separate the set

$D_{\pi,i}(G)$ from $V - (\beta_{\pi,i}(G) \cup D_{\pi,i}(G))$. In this way, it is not difficult to verify that an alternate definition for the border is given by:

$$\beta_{\pi,i}(G) = \{v_j : i+1 \leq j \leq n; (v_j, w) \in E \text{ for some } w \in D_{\pi,i}(G)\}$$

Given an ordering π of the vertices, we will "process" the vertices in that order. We will see that the time it takes to process each vertex is $O(|S|^{B_{\pi,i}(G)+1})$. Therefore, we can find the most likely configuration in time $O(n|S|^{MB_\pi(G)+1})$. So we are always looking to find orderings with a small maximum border size. Issues related to this will be discussed further in the next chapter.

Given an ordering of the vertices, how do we find the MLC? The idea is that we progress through the ordering, and at each point in the ordering we compute the best (i.e. most likely) choice for our current vertex for every possible configuration of the border. It is sufficient to consider only the border, because our conditional independence guarantees that the best choice is conditionally independent of the rest of the graph given the border. At the end we can progressively backtrack and choose the best possible configuration.

A more formal description of the algorithm is given here:

Initialize *current clique terms* list to include all clique terms.
for $i = 1$ to n **do**
  Let $v_i = \pi^{-1}(i)$ (i.e. the i-th vertex in the ordering π).
  for every possible configuration of $\beta_{\pi,i}(G)$ **do**
    Let $g_i(v_i, \beta_{\pi,i}(G)) = \prod$(*current clique terms* involving v_i).
    Compute and store $M_i(\beta_{\pi,i}(G)) = \max_{v_i \in S} g_i(v_i, \beta_{\pi,i}(G))$.
    Compute and store $A_i(\beta_{\pi,i}(G)) = \arg\max_{v_i \in S} g_i(v_i, \beta_{\pi,i}(G))$.
  end for
  Remove cliques involving v_i from the *current clique list.*
  Add $M_i(\beta_{\pi,i}(G)$ to the *current clique terms* list (involves "new" clique $\beta_{\pi,i}(G)$).
end for
Let $x^*_{\pi^{-1}(n)} = A_n(\emptyset)$.
for $i = n-1$ to 1 **do**
  Let $x^*_{\pi^{-1}(i)} = A_i(x^*_j : j \in \beta_{\pi,i}(G))$.
end for

At the end of this process $x^*_1, x^*_2, \ldots, x^*_n$ is the most likely configuration.

We will now demonstrate using this algorithm to find the MLC using the ordering $(7, 1, 6, 5, 2, 4, 3)$. Initally, our list of current clique terms is:

$$f_{124}(x_1, x_2, x_4), f_{134}(x_1, x_3, x_4), f_{345}(x_3, x_4, x_5), f_{26}(x_2, x_6), f_{56}(x_5, x_6), f_{67}(x_6, x_7)$$

So first we let $v_1 = 7$. Our border is $\{6\}$ and $g_1(x_6, x_7)$ is the product of all cliques involving x_7. So in this case, $g(x_6, x_7) = f_{67}(x_6, x_7)$. So for every value of x_6, we find the maximum $g(x_6, x_7)$ at every value of x_7, and find the value of x_7 which is best for that x_6. We then store two things.

$$\begin{aligned} M_1(x_6) &= \max_{x_7} g(x_6, x_7) \\ A_1(x_6) &= \arg\max_{x_7} g(x_6, x_7) \end{aligned}$$

We remove $f_{67}(x_6, x_7)$ from the current clique list and add $M_1(x_6)$ to that list. Our current clique list is now:

$$f_{124}(x_1, x_2, x_4), f_{134}(x_1, x_3, x_4), f_{345}(x_3, x_4, x_5), f_{26}(x_2, x_6), f_{56}(x_5, x_6), M_1(x_6)$$

We now move to $i = 2$. So $v_2 = 1$, our border is $\{2, 3, 4\}$, and $g_2(x_1, x_2, x_3, x_4) = f_{124}(x_1, x_3, x_4) f_{134}(x_1, x_3, x_4)$. So for every possible value of x_2,$x_3$and x_4 (that is, for all $|S|^3$ x_2, x_3, x_4 state combinations) we need to find the maximum (and arg max) of g_2 over the range of values of x_1. So we compute and store:

$$\begin{aligned} M_2(x_2, x_3, x_4) &= \max_{x_1} g(x_1, x_2, x_3, x_4) \\ &= \max_{x_1} f_{124}(x_1, x_2, x_4) f_{134}(x_1, x_3, x_4) \\ A_2(x_2, x_3, x_4) &= \arg\max_{x_1} f_{124}(x_1, x_2, x_4) f_{134}(x_1, x_3, x_4) \end{aligned}$$

This time, we remove both f_{124} and f_{134} from the current clique list and add $M_2(x_2, x_3, x_4)$.

Next, at $i = 3$ we have $v_3 = 6$. Our border is $\{2, 5\}$ and:

$$g_3(x_2, x_5, x_6) = f_{26}(x_2, x_6) f_{56}(x_5, x_6) M_1(x_6)$$

So for every possible value of x_2 and x_5 (that is, for all $|S|^2$ x_2, x_5 combinations) we need to find the maximum (and arg max) of g over the range of values of x_6. Therefore we compute and store:

$$\begin{aligned} M_3(x_2, x_5) &= \max_{x_6} g(x_2, x_5, x_6) \\ &= \max_{x_6} f_{26}(x_2, x_6) f_{56}(x_5, x_6) M_1(x_6) \\ A_3(x_2, x_5) &= \arg\max_{x_6} f_{26}(x_2, x_6) f_{56}(x_5, x_6) M_1(x_6) \end{aligned}$$

At our next step, $i = 4$, $v_4 = 5$, $\beta_{\pi,4}(G) = \{2, 3, 4\}$, and:

$$g_4(x_2, x_3, x_4, x_5) = f_{345}(x_3, x_4, x_5) M_3(x_2, x_5)$$

So,

$$\begin{aligned} M_4(x_2, x_3, x_4) &= \max_{x_5} f_{345}(x_3, x_4, x_5) M_3(x_2, x_5) \\ A_4(x_2, x_3, x_4) &= \arg\max_{x_5} f_{345}(x_3, x_4, x_5) M_3(x_2, x_5) \end{aligned}$$

Next, $v_5 = 2$,$\beta_{\pi,5}(G) = \{3, 4\}$, and $g_4(x_2, x_3, x_4) = M_2(x_2, x_3, x_4) M_4(x_2, x_3, x_4)$.

$$\begin{aligned} M_5(x_3, x_4) &= \max_{x_2} g_4(x_2, x_3, x_4) \\ &= \max_{x_2} M_2(x_2, x_3, x_4) M_4(x_2, x_3, x_4) \\ A_5(x_3, x_4) &= \arg\max_{x_2} M_2(x_2, x_3, x_4) M_4(x_2, x_3, x_4) \end{aligned}$$

Finishing the process we compute:

$$\begin{aligned} M_6(x_3) &= \max_{x_4} M_5(x_3, x_4) \\ A_6(x_3) &= \arg\max_{x_4} M_5(x_3, x_4) \end{aligned}$$

$$\begin{aligned} M_7 &= \max_{x_3} M_6(x_3) \\ A_7 &= \arg\max_{x_3} M_6(x_3) \end{aligned}$$

At this point, M_7 is the maximum value of our probability function P. All that remains is to find the variable configuration associated with it. We do this by backtracking through the A_i functions. We let $x_3^* = A_7$. So x_3^* is the value of x_3 at which the maximum value occurs. Then $x_4^* = A_6(x_3^*)$. Continuing we get:

$$\begin{aligned} x_2^* &= A_5(x_3^*, x_4^*) \\ x_5^* &= A_4(x_2^*, x_3^*, x_4^*) \\ x_6^* &= A_3(x_2^*, x_5^*) \\ x_1^* &= A_2(x_2^*, x_3^*, x_4^*) \\ x_7^* &= A_2(x_6^*) \end{aligned}$$

Analyzing the complexity of our algorithm, we see that in the forward part of the algorithm, each step requires us to consider every value of the current variable for every configuration of the border. So the most expensive step will take $|S|^{MB_\pi(G)+1}$ operations. There are n steps, so our complexity must be $O(n|S|^{MB_\pi(G)+1})$ operations.

Remark 1.2.5. Suppose instead of a probability function P which is a product of clique terms we had an arbitrary function Q which is a sum of clique terms:

$$Q(x_1, \ldots, x_n) = \sum_{C \in \mathcal{C}} f_C(x_C) \tag{1.2.1.5}$$

Note that the same procedure for maximizing Q (or indeed minimizing Q) would work with the obvious modifications.

1.2.2 Computing all Marginal Distributions

Another basic computation we can perform is to simultaneously find all the marginal distributions at once. Again we choose an ordering π of the vertices, and then use a "forward-backward" type procedure. We compute conditionals in the forward process, and marginals on the way back. Once again, our complexity will be $O(n|S|^{MB_\pi(G)+1})$.

In pseudocode, here is the algorithm:

initialize *current clique terms* list to include all clique terms
for $i = i$ to n do
  Let $v_i = \pi^{-1}(i)$ (the ith vertex in the ordering π).
  Let $g_i(v_i, \beta_{\pi,i}(G)) = \prod$(*current clique terms* involving v_i).
  Compute and store $h_i(\beta_{\pi,i}(G)) = \sum_{v_i} g_i(v_i, \beta_{\pi,i}(G))$.
  Compute and store $P(v_i|\beta_{\pi,i}(G)) = \frac{g_i(v_i, \beta_{\pi,i}(G))}{h_i(\beta_{\pi,i}(G)}$.
  Remove cliques terms involving v_i from the current clique list.
  Add $h_i(\beta_{\pi,i}(G))$ as a clique term (involves "new" clique $\beta_{\pi,i}(G)$).
end for
for $i = n$ to 1 do
  Let v_i be the ith vertex in the ordering.
  Compute and store $P(\beta_{\pi,i}(G)))$, if not already stored, from previous info.
  Compute and store $P(v_i, \beta_{\pi,i}(G)) = P(v_i|\beta_{\pi,i}(G))P(\beta_{\pi,i}(G))$.
  Compute and store $P(v_i) = \sum_{\beta_{\pi,i}(G)} P(v_i|\beta_{\pi,i}(G))P(\beta_{\pi,i}(G))$.

end for

Again, let us illustrate this procedure in detail using the ordering $(7, 1, 6, 5, 2, 4, 3)$.

In the first step, we want to compute $P(X_7|X_6)$. By our MRF property and basic probability we have:

$$\begin{aligned} P(X_7|X_6) &= P(X_7|X_1, X_2, \ldots, X_6) \\ &= \frac{P(X_1, X_2, X_3, X_4, X_5, X_6, X_7)}{\sum_{X_7} P(X_1, X_2, X_3, X_4, X_5, X_6, X_7)} \\ &= \frac{f_{124} f_{134} f_{345} f_{26} f_{56} f_{67}}{\sum_{X_7} f_{124} f_{134} f_{345} f_{26} f_{56} f_{67}} \\ &= \frac{f_{124} f_{134} f_{345} f_{26} f_{56} f_{67}(x_6, x_7)}{f_{124} f_{134} f_{345} f_{26} f_{56} \sum_{X_7} f_{67}(x_6, x_7)} \\ &= \frac{f_{67}(x_6, x_7)}{\sum_{X_7} f_{67}(x_6, x_7)} \end{aligned}$$

This demostrates why, in computing the conditional probabilities, we need only consider the cliques which involve our current vertex. The others can pull through the summation in the denominator and cancel out with the identical terms in the numerator. This also demonstrates why our overall computational complexity is again $O(n|S|^{MB_\pi(G)+1})$. Whenever we compute and store the conditional probability of a variable given its borders, we must evaluate an expression for every possible value of the border and the current variable. The summations are never more complex than this.

So following our algorithm precisely, our list of current clique terms is:

$$f_{124}(x_1, x_2, x_4), f_{134}(x_1, x_3, x_4), f_{345}(x_3, x_4, x_5), f_{26}(x_2, x_6), f_{56}(x_5, x_6), f_{67}(x_6, x_7)$$

At $i = 1$, $v_1 = 7$, and $\beta_{\pi,i}(G) = \{6\}$. So $g_1(x_6, x_7) = f_{67}(x_6, x_7)$ and then $h_1(x_6) = \sum_{x_7} f_{67}(x_6, x_7)$. We then compute and store:

$$P(X_7|X_6) = \frac{g_1(x_6, x_7)}{h_1(x_6)}$$

for every combination of X_6 and X_7.

Then we remove f_{67} from the list of current clique terms and add $h_1(x_6)$ to that list. Now our list of current clique terms is:

$$f_{124}(x_1, x_2, x_4), f_{134}(x_1, x_3, x_4), f_{345}(x_3, x_4, x_5), f_{26}(x_2, x_6), f_{56}(x_5, x_6), h_1(x_6)$$

We now increment to $i = 2$. Now $v_2 = 1$, and $\beta_{\pi,i}(G) = \{2, 3, 4\}$ so $g_2(x_1, x_2, x_3, x_4) = f_{124}(x_1, x_2, x_4) f_{134}(x_1, x_3, x_4)$. We compute and store:

$$\begin{aligned} h_2(x_2, x_3, x_4) &= \sum_{x_1} g_2(x_1, x_2, x_3, x_4) \\ P(X_1|X_2, X_3, X_4) &= \frac{g_2(x_1, x_2, x_3, x_4)}{\sum_{x_1} g_2(x_1, x_2, x_3, x_4)} \end{aligned}$$

Then we remove f_{124}, f_{134} from our current clique term list and add $h_2(x_2, x_3, x_4)$ to that list.

Continuing in this fashion we compute:

$$\begin{aligned}
g_3(x_2, x_5, x_6) &= f_{26}(x_2, x_6) f_{56}(x_5, x_6) h_1(x_6) \\
h_3(x_2, x_5) &= \sum_{x_6} g_3(x_2, x_5, x_6) \\
P(X_6 | X_2, X_5) &= \frac{g_3(x_2, x_5, x_6)}{h_3(x_2, x_5)}
\end{aligned}$$

$$\begin{aligned}
g_4(x_2, x_3, x_4, x_5) &= f_{345}(x_3, x_4, x_5) h_3(x_2, x_5) \\
h_4(x_2, x_3, x_4) &= \sum_{x_5} g_4(x_2, x_3, x_4, x_5) \\
P(X_5 | X_2, X_3, X_4) &= \frac{g_4(x_2, x_3, x_4, x_5)}{h_4(x_2, x_3, x_4)}
\end{aligned}$$

$$\begin{aligned}
g_5(x_2, x_3, x_4) &= h_4(x_2, x_3, x_4) h_2(x_2, x_3, x_4) \\
h_5(x_3, x_4) &= \sum_{x_2} g_5(x_2, x_3, x_4) \\
P(X_2 | X_3, X_4) &= \frac{g_5(x_2, x_3, x_4)}{h_5(x_3, x_4)}
\end{aligned}$$

$$\begin{aligned}
g_6(x_3, x_4) &= h_5(x_3, x_4) \\
h_6(x_3) &= \sum_{x_4} g_6(x_3, x_4) \\
P(X_4 | X_3) &= \frac{g_6(x_3, x_4)}{h_6(x_3)}
\end{aligned}$$

$$\begin{aligned}
g_7(x_3) &= h_6(x_3) \\
h_7 &= \sum_{x_3} g_7(x_3) \\
P(X_3) &= \frac{g_7(x_3)}{h_7}
\end{aligned}$$

At this point we have finished the "forward" part of our algorithm and, in fact, we already have computed the marginal distribution on X_3. Note that if we are only interested in finding a single marginal, we can just choose an ordering which ends with that variable, and thereby obtain that marginal without needing the "backward" part of the algorithm. Note further that since P is a probability function, then h_n (here h_7) will equal 1, since h_n is just the sum of the probabilities of all possible state combiniations. However, sometimes we have the situation where we only know that:

$$P \propto \prod_{C \in \mathcal{C}} f_C(x_C)$$

or in other words

$$P = \frac{1}{Z} \prod_{C \in \mathcal{C}} f_C(x_C)$$

for some unknown Z. In this case, our procedure still works, and $h_n = Z$.

Working backwards now, we compute:

$$\begin{aligned} P(X_3, X_4) &= P(X_4|X_3)P(X_3) \\ P(X_4) &= \sum_{x_3} P(X_3, X_4) \end{aligned}$$

$$\begin{aligned} P(X_2, X_3, X_4) &= P(X_2|X_3, X_4)P(X_3, X_4) \\ P(X_2) &= \sum_{X_3}\sum_{X_4} P(X_2, X_3, X_4) \end{aligned}$$

$$\begin{aligned} P(X_2, X_3, X_4, X_5) &= P(X_5|X_2, X_3, X_4)P(X_2, X_3, X_4) \\ P(X_5) &= \sum_{X_2}\sum_{X_3}\sum_{X_4} P(X_2, X_3, X_4, X_5) \end{aligned}$$

Up to this point, the step of computing $P(\beta_{\pi,i}(G))$ has been trivial. That is, it was always a computation we had already done in a previous step of the algorithm. In this next step, however, we are required to compute it explicitly.

$$\begin{aligned} P(X_2, X_5) &= \sum_{X_3}\sum_{X_4} P(X_2, X_3, X_4, X_5) \\ P(X_2, X_5, X_6) &= P(X_6|X_2, X_5)P(X_2, X_5) \\ P(X_6) &= \sum_{X_2}\sum_{X_5} P(X_2, X_5, X_6) \end{aligned}$$

$$\begin{aligned} P(X_1, X_2, X_3, X_4) &= P(X_1|X_2, X_3, X_4)P(X_2, X_3, X_4) \\ P(X_1) &= \sum_{X_2}\sum_{X_3}\sum_{X_4} P(X_1, X_2, X_3, X_4) \end{aligned}$$

$$\begin{aligned} P(X_6, X_7) &= P(X_6|X_7)P(X_7) \\ P(X_7) &= \sum_{X_6} P(X_6, X_7) \end{aligned}$$

In this way we are able to compute the marginal distributions of all of our variables. Notice that, our computation at each step in the forward process requires $O(|S|^{B_{\pi,i}(G)+1})$ operations since we compute $P(X_{\pi^{-1}(i)}|\beta_{\pi,i}(G))$ for every value of $X_{\pi^{-1}(i)}$ and $\beta_{\pi,i}(G)$. Going backward, we first compute $P(\beta_{\pi,i}(G))$ by summing over a previously computed joint probability. Since these joint probabilities never involve more than $MB_\pi(G) + 1$ variables, this step takes no more than $O(|S|^{MB_\pi(G)+1})$ operations. We then compute the joint probability $P(X_{\pi^{-1}(i)}, \beta_{\pi,i}(G))$ for every value of $X_{\pi^{-1}(i)}$ and $\beta_{\pi,i}(G)$. So our overall computational complexity is $O(n|S|^{MB_\pi(G)+1})$.

Note that although we refer specifically to computing marginal distributions, we can use the forward part of this process to efficiently (or at least more efficiently) compute the sum of an arbitrary product of functions:

$$\sum_{x_1,\ldots,x_n} \prod_{C\in\mathcal{C}} f_C(x_C)$$

as long as the variables involved in each f_C form a clique in the graph G.

Specifically, suppose in or previous example, we now want to compute:

$$E(q(X_2, X_4, X_6)) = \sum_{x_1,\ldots,x_n} q(x_2, x_4, x_6) \prod_{C \in \mathcal{C}} f_C(x_C)$$

Since $\{2, 4, 6\}$ does not form a clique in G we cannot simply regard q as another clique term and carry out our computations with respect to the graph G. Instead, we make a new graph H by adding the edge $(4, 6)$ to G. Now $\{2, 4, 6\}$ is a clique in H. Therefore if we let $\mathcal{C}'$ be the set of all maximal cliques in H, we can write $f_{246}(x_2, x_4, x_6) = q(x_2, x_4, x_6)$. Now:

$$E(q(X_2, X_4, X_6)) = \sum_{x_1,\ldots,x_n} \prod_{C \in \mathcal{C}'} f_C(x_C)$$

Therefore, by choosing an ordering of the vertices of H, and performing the forward part of the algorithm, we can compute this expectation in time $O(n|S|^{MB_\pi(H)+1})$.

Chapter 2

Tree-width and Computational Complexity

2.1 Introduction

In the last chapter, we described the *generalized DP* method for finding the most likely configuration or marginal probabilities of a Markov random field. We saw that given a probability distribution P, a graph G such that P is MRF with respect to G, and an ordering π of the vertices of G, we can perform an inference computation in time $O(n|S|^{MB_\pi(G)+1})$. We saw how different orderings could yield different computational times. This fact leads to some important questions. Given a graph, how intrinsically difficult is it to perform an inference computation? Given an ordering on a graph, is there another ordering with a lower maximum border? This chapter explores issues that arise from asking these questions. We show explicitly how the *tree-width* of a graph G, as defined implicity in [3], is the parameter which essentially determines the intrinsic complexity of basic computations on that graph. We explore equivalent notions of tree-width and demonstrate explicitly their equivalence. This exploration brings to light interesting analogies between the different methods of performing inference.

We go on to ask questions about determining the tree-width of a graph. It is an NP-complete problem in general to compute tree-width [3], so there has been much work on efficiently computing bounds. Much attention has been paid to upper bounds (e.g. [4], [26], [1]) for a variety of reasons, while work on lower bounds has been relatively sparse [21]. We prove a theorem relating a procedure called *maximum cardinality search* [29] to lower bounds on the tree-width of a graph. We then use this theorem to present a novel method for calculating a lower bound for the tree-width of a graph. We discuss its strengths and weaknesses, improve its performance through heuristics and an iterative method, and then analyze its performance on various classes of graphs.

2.2 Generalized DP, Junction trees, and complexity

Suppose we want to analyze the intrinsic difficulty of a graph G with respect to the generalized DP method. We know that given an ordering π, our computations will take time $O(n|S|^{MB_\pi(G)+1})$. So to measure how intrinsically complex a graph G is with respect to these basic computations using generalized DP, we should look at the minimum of $MB_\pi(G)$ over all possible orderings π. We call this quantity the *minimax border size* of G.

Definition 2.2.1. Let the *minimax border size* of a graph G be given by

$$MMB(G) = \min_{\text{orderings } \pi} MB_\pi(G) \tag{2.2.2.1}$$

So $MMB(G)$ is a measure of the inherent complexity of G with respect to the generalized DP method. For example, if G is an $n \times n$ lattice, then $MMB(G) = n$. The minimum is achieved with a simple ordering of the nodes from left to right in each row, processing the rows top to bottom. The graph in Figure 1.1 in Chapter 1 has minimax border size 3.

We have mentioned that there are other methods to perform these computations, which are basically equivalent. The major family of these approaches is what we will call the *junction tree* approach. Using this method, you must first choose a *triangulation* of G. To be precise, one chooses a graph $H = (V_H, E_H)$ with $V_H = V_G$ and $E_G \subseteq E_H$ such that H is triangulated.

Definition 2.2.2. A graph G is *triangulated* if every cycle $v_1, v_2, \ldots, v_k, v_1$ with $k \geq 4$ has a chord. A *triangulation* of a graph G is a graph H such that $V_H = V_G$, $E_G \subseteq E_H$ and H is triangulated. Given a graph G, we define

$$\mathcal{T}(G) = \{H : H \text{ is a triangulation of } G\} \tag{2.2.2.2}$$

After choosing such a triangulation H, you form a structure called a *junction tree*, which is basically a tree representation of cliques in the graph. Then there are various algorithms to perform inference on the junction tree. I refer the interested reader to [17], [19], or [14] for details. The important issue is that the complexity of the junction tree computations depends on the size of the largest clique in the triangulated graph H.

Definition 2.2.3. For a graph H let $MC(H) =$ the size of the largest clique in H.

Computing the most likely configuration or marginal distributions on a junction tree formed from the triangulation H will have a worst case time complexity $O(n|S|^{MC(H)})$ where n is the number of vertices. So to measure how intrinsically difficult a graph is with respect to this method we must consider the minimum over all possible triangulations H of G of $MC(H)$. We call this the *minimax clique size* of G.

Definition 2.2.4. Let the *minimax clique size* of a graph G, denoted $MMC(G)$ be given by

$$MMC(G) = \min_{H \in \mathcal{T}(G)} MC(H) \tag{2.2.2.3}$$

In other words, if we could find the best triangulation possible, we would be able to do junction tree computations in something which is exponential in $MMC(G)$. So if $MMC(G)$ is large, we can not compute efficiently on a graph using junction tree methods.

2.2.1 Equivalence Results

Here we show explicitly that the junction tree method and generalized DP are equivalent in their computational complexity. Beyond that, we demonstrate a natural correspondence between triangulations of a graph $G = (V_G, E_G)$ and specifying orderings π of the vertices of G. Then we show that computations using generalized dynamic programming with respect to a certain ordering are equivalent in complexity to junction tree propagation with the corresponding triangulation. Specifically, we will define a procedure to create a triangulated graph H given a graph G and π. We show that, in some sense, the "best" triangulations are ones that are derived from orderings and so we need not consider triangulations which do not come from orderings. Then we show a relation between the maximum border size of a graph G with respect to an ordering and the maximum clique size of the triangulation of G which corresponds to that ordering. It is then straightforward to demostrate that $MMB(G) = MMC(G) - 1$. Finally we show the relationship between these quantities and the *tree-width* of a graph. These results are not new, they are implied by the work in [2], [25], [3], [9]. However, in some cases they are not proved explicitly, the terminology varies, and they are scattered in the literature. This section is an attempt to summarize these results and make them all explicit under one framework.

Most of the notation on graphs was introduced in Chapter 1 but we must add a bit here. Given a Graph $G = (V, E)$ and a vertex $v \in V$ we denote the neighborhood of the vertex v by $\Gamma(v) = \{w \in V : (v, w) \in E\}$. The degree of vertex v is denoted by $deg(v) = |\Gamma(v)|$. The *family* of vertex v is defined as $\bar{\Gamma}(v) = \Gamma(v) \cup \{v\}$.

We will now give a procedure to construct a triangulation of a graph G given an ordering π of its vertices. This is known as the *elimination graph* of G with respect to π [29]. It is created by computing the *fill-in* $F_\pi(G)$ and adding the edges in the fill-in to the graph G. The fill-in is usually defined so as to exclude the edges that are already in the graph G, but it simplifies the notation considerably if we let the fill-in include those edges as well. So we will use a modified version of the fill-in for our purposes, but we include the original definition of the fill-in here for completeness.

Definition 2.2.5. Let the fill-in of a graph G with respect to the ordering π be defined as:

$$F_\pi(G) = \{(v_j, v_k) \notin E_G : j < k \text{ and there is a path from } v_j \text{ to } v_k \text{ in } G \text{ using only } v_k, v_j, \text{ and vertices that come before } v_j \text{ in the ordering } \pi\}$$

Rather than working with the fill-in directly, we will work with the *modified fill-in* given by

$$M_\pi(G) = \{(v_j, v_k) : j < k \text{ and there is a path from } v_j \text{ to } v_k \text{ in } G \text{ using only } v_k, v_j, \text{ and vertices that come before } v_j \text{ in the ordering } \pi\}$$

Note that the only difference between the two is that the modified fill-in does not exclude edges which are already in E_G. Furthermore, $E_G \subset M_\pi(G)$ since the edge (v_j, v_k) itself forms a path between v_j and v_k. So $M_\pi(G) = F_\pi(G) \cup E_G$.

The fill-in is related to the border of the graph in the following way. If we let

$$M_\pi^i(G) = \{(v_j, v_k) : i \leq j < k, v_j, v_k \in \beta_{\pi,i}(G)\} \tag{2.2.2.4}$$

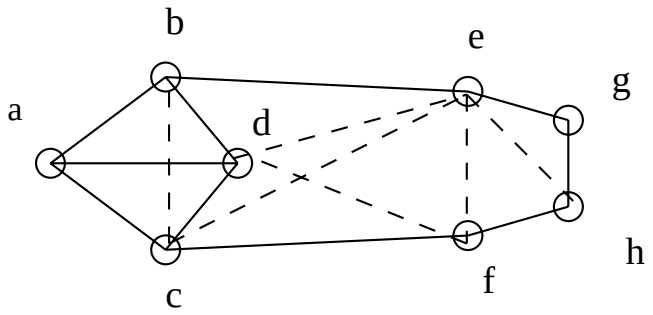

Figure 2.1: $T_\pi(G)$ for $\pi = (a, b, c, d, f, e, g, h)$. The solid edges were in E_G and the dotted edges are in $F_\pi(G)$.

then

$$M_\pi(G) = \bigcup_{i=1}^{n} M_\pi^i(G) \tag{2.2.2.5}$$

This is easily seen once it is noted that an equivalent definition for $M_\pi^i(G)$ is:

$$M_\pi^i(G) = \{(v_j, v_k) : i \le j < k \text{ and there is a path from } v_j \text{ to } v_k \text{ using only vertices } v_k, v_j, \text{ and } v_1, v_2, \ldots, v_i\}$$

Definition 2.2.6. For any graph G and any ordering π of the vertices of G, let the π-*triangulation of* G, denoted $T_\pi(G)$ be given by $T_\pi(G) = (V_G, M_\pi(G))$. Equivalently we can let $T_\pi(G) = (V_G, E_G \cup F_\pi(G))$.

So another way of thinking of the process of creating the π-triangulation of G is the following. At stage 1, we find the border $\beta_{\pi,1}(G)$ and let $F_\pi^1(G)$ be all the edges between vertices of $\beta_{\pi,1}(G)$ that are not already present in E_G. Let $G_1 = (V_G, E_G \cup F_\pi^1(G))$. In general, at stage i we find $\beta_{\pi,i}(G)$ and let $F_\pi^i(G)$ be the edges between vertices of $\beta_{\pi,i}(G)$ that are not present in $E_{G_{i-1}}$. Then we let $G_i = (V_G, E_{G_{i-1}} \cup F_\pi^i(G))$. Following this process to the end, we get that $T_\pi(G) = G_n$.

We illustrate this procedure with an example. In Figure 2.1, we see the triangulation of a graph G with respect to ordering $\pi = (a, b, c, d, f, e, g, h)$ (Note: *not* (a, b, c, d, e, f, g, h)). In the first stage we have $\beta_{\pi,1}(G) = \{b, c, d\}$. The edges (b, d) and (d, c) are already in E_G, but (b, c) is not. So $F_\pi^1(G) = \{(b, c)\}$ ($M_\pi^1 = \{(b, c), (c, d), (b, d)\}$). In the next step, $\beta_{\pi,2}(G) = \{c, d, e\}$. This time, only $(c, d) \in E_{G_1}$, so $F_\pi^2(G) = \{(c, e), (d, e)\}$. Next, $\beta_{\pi,3}(G) = \{d, e, f\}$. Only the edge $(d, e) \in E_{G_2}$ since it was in $F_\pi^2(G)$, so we have $F_\pi^3(G) = \{(d, f), (e, f)\}$. Continuing, $F_\pi^4(G) = \emptyset$, since $\beta_{\pi,3}(G) = \{e, f\}$ and $(e, f) \in F_\pi^3(G)$. Continuing the procedure to the end, we get the result in Figure 2.1.

Definition 2.2.7. [28] An ordering π of a graph G is said to be a *perfect elimination ordering* (also called *zero fill-in*) if $M_\pi(G) = E_G$ or equivalently $F_\pi(G) = \emptyset$.

Lemma 2.2.8. *[27] [29] A graph G is triangulated iff it has a perfect elimination ordering.*

Lemma 2.2.9. *[29] The ordering π is a perfect elimination ordering for $T_\pi(G)$.*

An immediate consequence of these two lemma is that $T_\pi(G)$ is triangulated.

Lemma 2.2.10. *Suppose $G = (V_G, E_G)$ and $H = (V_G, E_H)$ are two graphs defined on the same vertex set and π is some ordering of that vertex set. If $E_G \subset E_H$ then $T_\pi(G) \subset T_\pi(H)$.*

Proof. All we need to show is that $M_\pi(G) \subseteq M_\pi(H)$. Clearly if there is a path from v_j to v_k in G using only $v_1, \ldots, v_j$, and v_k, then such a path exists in H, since $E_G \subset E_H$. So $M_\pi(G) \subset M_\pi(H)$ and therefore $T_\pi(G) \subset T_\pi(H)$. □

Lemma 2.2.11. *Suppose we are given a graph G and any triangulation H of G. Then there exists some ordering π such that $T_\pi(G) \subseteq H$.*

Proof. Let $\Pi(H)$ be the set of perfect elimination orderings π with respect to the graph H. By Lemma 2.2.8 this set is nonempty. Choose any $\pi \in \Pi(H)$. We know $G \subset H$, therefore by Lemma 2.2.10 $T_\pi(G) \subset T_\pi(H) = H$. □

This shows us that the triangulations which do not come from orderings are inferior to those that do come from orderings. More precisely, if H is a triangulation of G that could not be produced by an ordering, then we can remove edges from H to make a graph H' which is also a triangulation of G and such that $H' = T_\pi(G)$ for some π. So the triangulations that come from orderings are "optimal" in that sense. Specifically, it allows us to state the following:

Lemma 2.2.12.

$$\min_{H \in \mathcal{T}(G)} MC(H) = \min_{\text{orderings } \pi} MC(T_\pi(G)) \tag{2.2.2.6}$$

Proof. Clearly we have that the LHS $\leq$ RHS since the LHS minimizes over a bigger set. Suppose the minimum of the LHS is achieved by some H which is not $T_\pi(G)$ for any π. Let α be a perfect elimination order for H. Then $T_\alpha(G) \subseteq H$ and therefore the RHS $\leq$ LHS. □

Theorem 2.2.13. *Let $G = (V_G, E_G)$ be a graph and π be an ordering of its vertices. Let m be the size of the largest clique in $T_\pi(G)$. Then $MB_\pi(G) = m - 1$.*

Proof. Let $v_i = \pi^{-1}(i)$ and let $C \subset V_G$ be the largest clique in $T_\pi(G)$. So $|C| = m$. Let v_j be the first element of C in the ordering π. In other words, $v_j <_\pi v_i$ for all $v_i \in C$, $v_i \neq v_j$. Since for any $v_k \in C - \{v_j\}$, we have $(v_j, v_k) \in T_\pi(G)$ we know that either $(v_j, v_k) \in E_G$ or $(v_j, v_k) \in F_\pi(G)$. In either case there is clearly a path from v_j to v_k which involves only v_j, v_k and $v_1, \ldots, v_{j-1}$. So $C - \{v_j\} \subseteq \beta_{\pi,j}(G)$. So $MB_\pi(G) \geq m - 1$.

Now suppose by contradiction that $MB_\pi(G) \geq m$. This means that there is some j for which $|\beta_{\pi,j}(G)| \geq m$. Let $D = \{v_j\} \cup \beta_{\pi,j}(G)$. We show that D is a clique of size $m + 1$ in $T_\pi(G)$, contradicting our assumption. Take any two vertices $w_1, w_2 \in D$ with $w_1 <_\pi w_2$. If $(w_1, w_2) \in E_G$ then clearly it is in $T_\pi(G)$. Otherwise, we know there is a path from v_j to w_1 which involves only those two vertices and $v_1, \ldots, v_{j-1}$. The same is true for v_j and w_2. Concatenating the two paths gives us a path from w_1 to w_2 involving only vertices which come before them in the ordering. Therefore $(w_1, w_2) \in F_\pi(G)$. So any two vertices in D are adjacent in $T_\pi(G)$. This gives our contradiction and shows that $MB_\pi(G) = m - 1$. □

Corollary 2.2.14. $MMB(G) = MMC(G) - 1$.

Proof. For any ordering π we know that $MB_\pi(G) = MC(T_\pi(G)) - 1$. So clearly:

$$MMB(G) = \min_\pi MC(T_\pi(G)) - 1 \tag{2.2.2.7}$$

and by Lemma 2.2.12 the corollary is proved. □

2.2.2 Tree width

At this point, we would like to introduce the definition of a *k-tree* and the associated definition of the *tree-width* of a graph. As we will demostrate shortly, there are many equivalent definitions for the tree-width of a graph yet proofs of equivalence are not always given explicity. The original definition of k-tree was given by Rose [27] in 1970. The tree-width of a graph G was implicitly defined in [3] when Arnborg et al. refer to "the smallest number k such that a given graph is a partial k-tree", without naming this quantity with the label "tree-width". In this article they show that finding this k is an NP-complete problem. However, these definitions are not typically discussed in the literature on graphical models, nor is the concept of a *k-tree*. Instead, most attention is paid to the definition of tree-width in [25] which defines tree-width relative to *tree decompositions* and then asserts (without proof) its equivalence to what we define as $MMC(G) - 1$. They then refer to the simultaneous work in [3] as showing that determining the tree-width of a graph is an NP-complete problem, implying that their definition in [25] is equivalent to the implicit definition in [3]. Again no epxlicit proof is given. In [2] Arnborg defines the *dimension* of a graph and *dimension of a graph with respect to* π. These are easily seen to be equivalent to what are referred to here as $MMB(G)$ and $MB_\pi(G)$ respectively. In that article, it is proved that G has dimension at most k if and only if G is a partial k-tree, which effectively proves the equivalence of $MMB(G)$ and the tree-width of G (denoted $TW(G)$, to be defined later in this section). So while it has been known for some time that the tree-width of a graph G in its various incarnations are all equivalent, there is apparently no single source with a clean statement of these equivalences. It is, therefore, perhaps worthwhile to spell them out here. Later, these results will be used in establishing our lower bound on computational complexity.

Definition 2.2.15. [27] A *k-tree* can be defined recursively in the following way. First, the complete graph on k vertices is a k-tree. Secondly, given a k-tree on n vertices (for $n \geq k$), we can form a k-tree on $n+1$ vertices by connecting our new vertex to k existing vertices which form a complete subgraph in our n-vertex subgraph.

The following alternative definition of a k-tree is due to [28].

Theorem 2.2.16. *The following are necessary and sufficient conditions for a graph G to be a k-tree.*

1. *G is connected*
2. *G contains a k-clique but no $k+2$ clique*
3. *Every minimal x-y separator of G is a k-clique.*

Recall that an *x-y separator* refers to a set of vertices S ($x, y \notin S$) such that any path from vertex x to vertex y must pass through a vertex in S. It is defined only when x and y are non-adjacent. By *minimal* we mean a separator with the smallest possible number of nodes.

Definition 2.2.17. Let G be a k-tree on n vertices and let α be an ordering of the vertices of G. Let $v_i = \alpha^{-1}(i)$. We define $\mu_{\alpha,i}(G) = \Gamma(v_i) \cap \{v_1, v_2, \ldots, v_{i-1}\}$. We say that α is a *construction order* of G if $\{v_1, v_2, \ldots, v_k\}$ is a clique and $\mu_{\alpha,i}(G)$ is a clique of size k for all $k+1 \leq i \leq n$.

It can be easily verified from the definitions that G is a k-tree if and only if there exists some ordering α such that α is a construction order of G. Take a construction order for a k-tree, and let π be the reverse of that ordering. Then π gives a perfect elimination ordering for the k-tree. Therefore k-trees are triangulated by Lemma 2.2.8 (also stated directly in [27]). Note also that $MB_\pi(G) = k$, which means, as we will soon see, that π is an ordering which achieves the minimax border.

Definition 2.2.18. A *partial k-tree* is a graph which is a subgraph of some k-tree.

Lemma 2.2.19. *[28] Let H be a k-tree and let $C = \{w_1, w_2, \ldots, w_k\}$ be a k-clique in H. Then there exists a construction order α on H such that if $v_i = \alpha^{-1}(i)$ then $\{v_1, \ldots, v_k\} = \{w_1, \ldots, w_k\}$. In other words, any clique can be the starting clique for the recursive process of building a k-tree given in the definition.*

This lemma implies, among other things, that an ordering of minimax border can end in any node we choose. Recall from Chapter 1 that to compute a single marginal distribution, we suggested using only the "forward" part of the algortihm to compute all the marginals with an ordering that ends on the node of interest. This lemma says that we do not pay the price of a higher maximum border when we restrict our orderings to end on a specific node.

Definition 2.2.20. The *tree width* of a graph G, denoted $TW(G)$ is defined as the smallest number k for which G is a partial k-tree.

Robertson and Seymour define tree-width through the definition of a *tree-decomposition.*

Definition 2.2.21. [25] A tree decomposition of G is a family $(X_i : i \in I)$ of subsets of V_G, together with a tree T with $V_T = I$ with the following properties:

1. $\bigcup_{i \in I} X_i = V_G$
2. Every edge of G has both its endpoints in some $X_i (i \in I)$
3. For $i, j, k \in I$ if j lies on the path of T from i to k then $X_i \cap X_k \subseteq X_j$

Definition 2.2.22. [25] The *width* of the tree-decomposition is $\max_{i \in I} |X_i| - 1$.

Definition 2.2.23. [25] The *tree-width* of G is the minimum $k \geq 0$ such that G has a tree-decomposition of width $\leq k$.

By using the first definition of tree-width, we avoid having to consider tree-decompositions at all. Instead we can work directly with k-trees which, in the opinion of this author, are simpler and more intuitive. When we refer to $TW(G)$ in this paper, we are working directly with the k-tree definition although the two are equivalent.

Lemma 2.2.24. *Let G be a k-tree and let $A \subset V_G$ such that $|A| \leq k$ and A forms a complete subgraph in G. Then there exists a set D, such that $|D| = k$, D forms a complete subgraph of G and $A \subset D$.*

Proof. G is a k-tree, therefore there exists a construction order α. Let $v_i = \alpha^{-1}(i)$ and consider the element v_i of A which comes last in this construction order. In other words $v_j \in A$ for $j \neq i$ implies $v_j <_\alpha v_i$. If $i \leq k$ then we let $D = \{v_1, v_2, \ldots, v_k\}$ since clearly D is a clique of size k and $A \subset D$. Otherwise consider $C = \mu_{\alpha,i}(G) \cup \{v_i\}$. Clearly C is a clique, since $\mu_{\alpha,i}(G)$ is a clique and v_i is adjacent to every member of $\mu_{\alpha,i}(G)$. We also know that $A \subset C$ since any neighbor v_j of v_i such that $v_j <_\alpha v_i$ is by definition in $\mu_{\alpha,i}$. But $|C|$ is $k+1$ and by the assumption of our lemma $|A| \leq k$. Therefore, $C - A$ is nonempty. Consequently, we can choose any element $w \in C - A$ and let $D = C - \{w\}$. Then D is still a clique, $|D| = k$, and $A \subset D$. □

Theorem 2.2.25. *For any graph G, $MMB(G) = TW(G)$.*

Proof. If G is simply a clique on n nodes then $TW(G) = n-1$ and $MMB(G) = n-1$ by choosing any ordering. So let $TW(G) = k$ and let $|V_G| = n > k$.

First we will show that $MMB(G) \leq TW(G)$. Since G is a partial k-tree, we can add edges to G to form a k-tree H. By Theorem 2.2.16 the graph H has no $k+2$ clique so $MC(H) \leq k+1$. Furthermore, H is a triangulation of G so $MMC(G) \leq k+1$. By Corollary 2.2.14 we have that $MMB(G) \leq k = TW(G)$.

Now we must show $MMB(G) \geq TW(G)$. We will proceed by induction on the number of vertices of G. For our base case $n = 2$, our inequality holds trivially. Now assume it is true for graphs on $n-1$ vertices, we will show it is true for graphs on n vertices. Let G be a graph on n vertices with $MMB(G) = k$. There exists an ordering π such that $MB_\pi(G) = k$. Let $v_i = \pi^{-1}(i)$ and let $A = \Gamma_G(v_1)$. Clearly $|A| \leq k$, since $B_{\pi,1}(G) = deg(v_1)$. Consider the graph H formed by connecting pairwise all the vertices in A and then removing v_1 from the graph. Define an ordering π_1 of the vertices of H by

$$\pi_1 = (v_2, v_3, \ldots, v_n) \tag{2.2.2.8}$$

Claim 2.2.26. $\beta_{\pi_1,i}(H) \subseteq \beta_{\pi,i+1}(G)$

Proof. Let $v_j \in \beta_{\pi_1,i}(H)$. So $j \geq i+2$ and there is a path from v_{i+1} to v_j in H involving only $v_2, v_3, \ldots, v_{i+1}$ and v_j. (We have to shift our indices since π_1 starts at v_2.) If all the edges of this path are in G then we have $v_j \in \beta_{\pi,i+i}(G)$. Otherwise the path involves some edge (v_k, v_l) for $2 \leq k, j \leq i+1$, which is in H but not in G. That implies that v_k, v_l are both adjacent to v_1 in G. So replacing the edge (v_k, v_l) with the edges (v_k, v_1) and (v_1, v_l) gives a path between v_{i+1} and v_j in G involving only $v_1, \ldots v_{i+1}$ and v_j. This implies $v_j \in \beta_{\pi,i+1}(G)$. □

By this claim we know $MB_{\pi_1}(H) \leq MB_\pi(G) = k$. By our inductive assumption, H is a partial k-tree.

So let H' be a k-tree such that $H \subset H'$. Consider again the set A. It is completely connected in H'. By Lemma 2.2.24 there must be a set of vertices B of size k such that B is a complete subgraph of H' and $A \subset B$. Form the graph G' from H' by adding back the vertex v_1 and adding edges from v_1 to every element of B. By Definition 2.2.15, G' is a k-tree. Let e be any edge in G. If $e = (v_1, v_k)$ then $v_k \in A$ therefore $v_k \in B$ and so $(v_1, v_k) \in G'$. Otherwise $e = (v_i, v_j)$ with $i, j \neq 1$. Then $e \in H$ therefore $e \in H'$

therefore $e \in G'$. We can now conclude that $G \subset G'$. Since G' is a k-tree, $TW(G) \leq k = MMB(G)$. □

This result has several consequences. First of all, it shows clearly that partial k-trees are, in some sense, the class of graphs on which we can compute efficiently. If our computational power is such that we can handle, say $|S|^6$ operations but not $|S|^7$, then we are restricting ourselves to the class of 5-trees (with a reasonable number of vertices). We also see clearly a relationship between construction orders on a k-tree and orderings of minimax border. Given an arbitrary graph G, one way to find an ordering of minimax border would be to embed G in a k-tree H with $k = TW(G)$ and then take the reverse of a construction order on H. Of course, finding both $TW(G)$ and such and a k-tree H such that $G \subset H$ are difficult problems. However, we can now see an equivalence between finding k-trees which contain G, finding orderings on G with small max border, and finding triangulations of G with a small largest clique.

2.3 Computing and bounding tree-width

Since the tree-width of a graph is intricately related to the computational complexity of a variety of methods, we would like to be able to compute it directly. However, computing the tree-width of an arbitrary graph is a NP-complete problem [3]. So the best we can hope for is to bound this number.

Much attention has been paid to finding upper bounds for the tree-width of a graph. This is for reasons of simplicity and practicality. Since tree-width can be expressed as a minimum in at least two different ways, finding an upper bound is not too difficult. Choose any ordering π of the vertices of G, compute the maximum border $MB_\pi(G)$ and that is an upper bound for $TW(G)$. Likewise, choosing any triangulation H of G and finding $MC(H) - 1$ also yields an upper bound for tree-width.

A second reason for the attention paid to upper bounds is that an upper bound for a particular problem can demonstrate that it is feasible to solve. If the upper bound for computation is not outlandish, we know that problem is tractable. By contrast, a lower bound can only tell you that a problem is hopeless, at least by conventional methods. So it is a somewhat more pessimistic contribution to be able to confirm that a problem is intractable.

For these reasons, relatively little work has been gone on finding lower bounds for $TW(G)$. Trivially, the minimum degree of the graph is a lower bound. A more sophisticated bound, found in [21] is the minimum over all pairs of non-adjacent vertices of the maximum degree of the two vertices in the pair. Both of these can be easily rendered worthless on a graph with two non-adjacent vertices of low degree. One less than the size of the largest clique in G is also a lower bound. However, this too can be quite weak and it is not terribly efficient to find cliques in a graph anyway. We show that a procedure called maximum cardinality search can efficiently find a lower bound to $TW(G)$. This bound always does at least as well as finding the largest clique in G and typically beats the bound in [21].

2.3.1 Maximum Cardinality Search

We now introduce the procedure called *maximum cardinality search* (MCS). MCS is best known as a method to test whether a graph is triangulated [29]. The following is a succinct description of the procedure:

> Give number 1 to an arbitrary node. Number the nodes consecutively, choosing as the next to number an unnumbered node with a maximum number of previously numbered neighbors. Break ties arbitrarily. [17]

To explain more thoroughly, we start forming a *numbering* by first assigning the number 1 to an arbitrary vertex. Then given that we have numbered i vertices already, we give the number $i+1$ to the unnumbered vertex with the most neighbors in the set of already numbered vertices, breaking ties arbitrarily. Now define an ordering π which maps each vertex to its number. We say that π is an ordering generated by maximum cardinality search, or more concisely, an MCS ordering. We will now give a more formal definition.

Definition 2.3.1. Let $G = (V, E)$ and let $T \subset V$. For $v \in V$, let $d_T(v) = |\{w \in T : (v, w) \in E\}|$.

Definition 2.3.2. Let $G = (V, E)$ be a graph and let π be an ordering of the vertices. Let $v_i = \pi^{-1}(i)$ and let $T_i = \{v_1, v_2, \ldots, v_i\}$. If for all $i = 2, 3, \ldots, n$ we have that $d_{T_{i-1}}(v_i) \geq d_{T_{i-1}}(v_j)$ for all $j = i+1, i+2, \ldots, n$ then we say π is an *MCS ordering* on G.

MCS can be used to test whether a graph is triangulated in the following manner. Let G be a graph and let π_1 be an MCS ordering on G. Now let π_2 be the *reverse* of π_1. That is, let $\pi_2(v) = n + 1 - \pi_1(v)$. Then G is triangulated if and only if π_2 is a perfect elimination ordering on G. [29]

There is some inconsistency in the literature as to whether we should number the vertices in ascending or descending order in the MCS procedure. Some references use the convention that you start by arbitrarily assigning n to a vertex, then $n-1$, etc. This is motivated by the fact that one heuristic to find an ordering of minimum border is to use what we call π_2 in the previous paragraph. This is suggested in, for example [24], among other sources. However, we will use the convention of numbering the nodes in ascending order, as described at the beginning of this section.

2.3.2 Main Result

The main result in this section is that MCS also gives a lower bound on the tree-width of a graph in the following manner.

Theorem 2.3.3. *Let $G = (V_G, E_G)$ be a graph on n vertices. Let $\pi = (v_1, v_2, \ldots, v_n)$ be an MCS ordering on G. Then $TW(G) \geq deg(v_n)$.*

This is our main theorem and the proof will follow shortly. First we show the following corollary is an immediate consequence of Theorem 2.3.3.

Corollary 2.3.4. *Let G be a graph and let $\pi = (v_1, v_2, \ldots, v_n)$ be an MCS ordering on G. Let $T_i = \{v_1, v_2, \ldots, v_i\}$. Then $TW(G) \geq \max_i d_{T_{i-1}}(v_i)$.*

Proof. Let $k = \max_i d_{T_{i-1}}(v_i)$ and $j = \arg\max_i d_{T_{i-1}}(v_i)$. Let H be the subgraph of G generated by the set of vertices T_j. The ordering $v_1, v_2, \ldots, v_j$ is an MCS ordering for H, and in the graph H, the vertex v_i has degree k. So by Theorem 2.3.3, $TW(H) \geq k$ which implies $TW(G) \geq k$. □

Before proceeding to the proof of Theorem 2.3.3 we prove a necessary lemma.

Lemma 2.3.5. *Let $G = (V, E)$ be a graph with $|V| = n$. Suppose we have a partition of the set V into three disjoint sets $X \cup Y \cup S = V$ such that for any $x \in X$ and $y \in Y$, S is an x, y-separator. Let π be an ordering of the vertices generated by maximum cardinality search and let $w_i = \pi^{-1}(i)$. Let $T_i = \{w_1, w_2, \ldots, w_i\}$ for $i = 1, 2, \ldots, n$. Then:*

$$|T_i \cap S| \geq \min\{\max_{v \in X - T_i} d_{T_i}(v), \max_{v \in Y - T_i} d_{T_i}(v)\} \tag{2.3.2.1}$$

Proof. We will prove the lemma by induction on i. Consider first the base case $i = 1$. In order for the right hand side of our inequality to be 1, w_1 must be adjacent to both a vertex in X and a vertex in Y. Clearly, such a vertex must be in S which means the left hand side is also 1. Otherwise the right hand side is 0 and the inequality holds trivially.

We will now assume the inequality holds for i and prove that it must be true for $i+1$. So assume that Equation 2.3.2.1 holds for i and recall that $T_{i+1} = T_i \cup \{w_{i+1}\}$. We will

examine how the two sides of the inequality change as we go form i to $i+1$ under two cases.

Case 1: $w_{i+1} \in S$. Then the LHS increases by 1, and the RHS increases by at most 1. That is, $\max_{v \in X - T_i} d_{T_{i+1}}(v) \leq \max_{v \in X - T_i}(d_{T_i}(v) + 1)$. So the inequality still holds.

Case 2: $w_{i+1} \notin S$. So w_{i+1} cannot border both a vertex in X and a vertex in Y. WLOG assume $w_{i+1} \in X$. This means that $d_{T_i}(w_{i+1}) \geq d_{T_i}(v)$ for all $v \in V - T_i$. So we have:

$$\max_{v \in X - T_i} d_{T_i}(v) \geq \max_{v \in Y - T_i} d_{T_i}(v) \tag{2.3.2.2}$$

Since $w_{i+1} \in X$ (or more precisely, $w_{i+1} \in X - T_i$), we know that:

$$\max_{v \in Y - T_i} d_{T_i}(v) = \max_{v \in Y - T_{i+1}} d_{T_{i+1}}(v) \tag{2.3.2.3}$$

Since the RHS of 2.3.2.1 is the min of two items, the smaller of which does not increase as we go from i to $i+1$, we can conclude that the RHS does not increase as we go from i to $i+1$.

□

We are now ready to prove our main result.

Proof of Theorem 2.3.3. Let $k = deg(v_n)$. This will be a proof by contradiction. We assume $TW(G) \leq k-1$ and go on to show that this is inconsistent with an ordering π that ends in v_n. Specifically, we will work for a long time to isolate a particular vertex v^* and a set of vertices D which separates v^* from the previously numbered vertex in our maximum cardinality search. We use Lemma 2.3.5 to indicate that certain vertices in D must have already been numbered. We then derive a contradiction by showing that when another vertex, which we call z, is numbered according to the ordering π, it in fact has fewer numbered neighbors than v_n, contradicting our assumption that π is an MCS ordering.

Let $w_1, w_2, \ldots, w_k$ be the k neighbors of v_n labeled such that $l < m \Rightarrow w_l <_\pi w_m$. If $TW(G) \leq k-1$, then there exists a $(k-1)$-tree $H = (V_G, E_H)$ such that $E_G \subseteq E_H$. Let i be the lowest index such that $\{w_{i+1}, w_{i+2}, \ldots, w_k, v_n\}$ form a clique in H. A $(k-1)$-tree cannot contain a $(k+1)$-clique, so we know that $i \geq 1$ and $\{w_k, v_n\}$ form a clique of size 2 so $i \leq k-1$. Therefore i exists and $1 \leq i \leq k-1$. By the definition of i we know that w_i is not adjacent to all of $\{w_{i+1}, w_{i+2}, \ldots, w_k\}$ (it is adjacent to v_n of course). So let j be the smallest index in $i+1, \ldots, k$ such that $(w_i, w_j) \notin E_H$.

We will now define the following sets:

1. $C_1 = \{w_{i+1}, w_{i+2}, \ldots, w_{j-1}\}$
2. $C_2 = \{w_l : j \leq l \leq k, (w_i, w_l) \notin E_H\}$
3. $C_3 = \{w_l : j \leq l \leq k, (w_i, w_l) \in E_H\}$

Note the following straightforward properties of these sets:

1. $C_1 \cap C_2 = C_1 \cap C_3 = C_2 \cap C_3 = \emptyset$

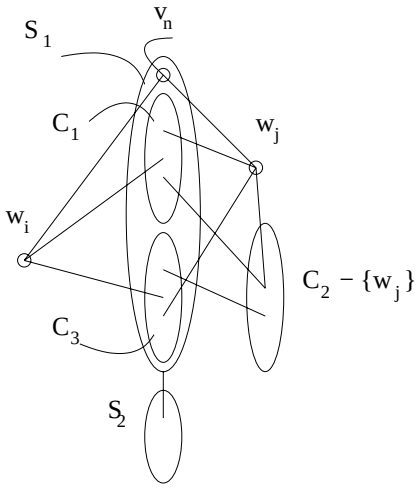

Figure 2.2: Lines demonstrate edges that must exist in H

2. $|C_1 \cup C_2 \cup C_3| = k - i$
3. $v \in C_1 \cup C_2 \cup C_3 \Rightarrow v >_\pi w_i$
4. $v \in C_1 \cup C_3 \Rightarrow (w_i, v) \in E_H$
5. $v \in C_2 \Rightarrow (w_i, v) \notin E_H$
6. $w_j \in C_2$
7. $C_1 \cup C_2 \cup C_3 \cup \{v_n\}$ form a clique in H.

Since $(w_i, w_j) \notin E_H$, by the separation property, we know that there exists a set of vertices S which form a $(k-1)$-clique in H, such that any path from w_i to w_j in H must pass through a vertex in S. Choose such a set S. It is easy to see that $\{v_n\} \cup C_1 \cup C_3 \subseteq S$ since all of those vertices are adjacent to both w_i and w_j in H. Let $S_1 = C_1 \cup C_3 \cup \{v_n\}$, $S_2 = S - S_1$ and let $c_2 = |C_2|$. So $|S_1| = k - i + 1 - c_2$. See Figure 2.2 for a schematic drawing of these various sets.

Now let α be a construction order on H which starts with the clique S as its basis. By Lemma 2.2.19, such an ordering exists. Let v^* be the last element of C_2 with respect to the ordering α. In other words, for all $v \in C_2, v \neq v^*$ we have that $v <_\alpha v^*$. Since $v_j \in C_2$ and v_j is not in the basis for the construction order α, we know that v^* is not in the basis of α. So let $D =$ the $(k-1)$ clique that v^* is adjoined to when H is constructed using the construction order α.

Claim 2.3.6. *D is a w_i, v^*-separator.*

Proof. We know S is a w_i, v^*-separator. This is true because S is a w_i-w_j separator and w_j and v^* are either the same vertex or are adjacent to one another in H. So consider any path from w_i to v^*. It uses some vertex in S. If that vertex is also in D then clearly the path goes through D. Otherwise, since S is our basis in the construction order α, any path from a vertex in $S - D$ to v^* must go through D. So any path from w_i to v^* must go through D. □

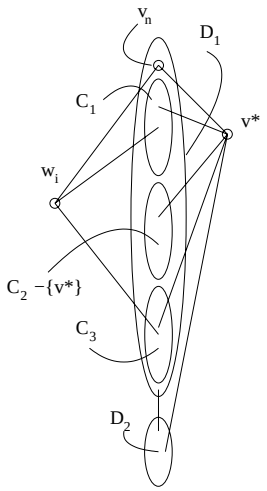

Figure 2.3: Lines demonstrate edges that must exist in H

Now let $D_1 = S_1 \cup (C_2 - \{v^*\})$. Clearly $D_1 \subset D$ since $v \in D_1 \Rightarrow (v, v^*) \in E_H$ and $v <_\alpha v^*$. Note that $|D_1| = k - i$. Let $D_2 = D - D_1$, we have $|D_2| = i - 1$. See Figure 2.3 for another schematic drawing.

Let T_1 be the set of vertices numbered before w_i. Since $\{w_1, \ldots, w_{i-1}\} \subset T_1$ and are all neighbors of v_n, we know $d_{T_1}(v_n) = i - 1$. Since w_i is numbered next, it must have at least as many "numbered neighbors" as v_n. Therefore $d_{T_1}(w_i) \geq i - 1$. If the set D were removed from H, the resulting graph would be disconnected. Let Z be the connected component containing v^* in this disconnected graph. Let $Z_1 = Z - T_1$, so Z_1 is the set of vertices in Z numbered after w_i. We know $v^*, w_j \in Z_1$ since they are both in C_2 and are therefore numbered after w_i (recall that it is possible that $v^* = w_j$). Clearly for any vertex $v \in Z_1$, $d_{T_1}(v) \leq d_{T_1}(w_i)$. Let $m = \max_{v \in Z_1} d_{T_1}(v)$. By Lemma 2.3.5, (with D separating Z from $V - \{Z \cup D\}$) we know at least m vertices of D must already be numbered. Let $D_2 = D - D_1$. Since $|D| = k-1$,$|D_1| = k-i$, and $D_1 \subseteq D$, we know that $|D_2| = i-1$. Let $N = T_1 \cap D$, this is the set of vertices in D which are numbered before w_i. So $|N| \geq m$. Furthermore, $N \subseteq D_2$ since $v \in D_1 \Rightarrow w_i <_\pi v$. In other words, $T_1 \cap D_1 = \emptyset$. So we can conclude that $m \leq |N| \leq |D_2| = i - 1$.

Let $T_2 = T_1 \cup \{w_i\}$. This corresponds to the set of numbered vertices immediately after w_i is numbered. For all $v \in Z_1$ we know that $(v, w_i) \notin E_H$ which of course implies that $(v, w_i) \notin E_G$. So

$$\max_{v \in Z_1} d_{T_2}(v) = \max_{v \in Z_1} d_{T_1}(v) = m \leq i - 1 \tag{2.3.2.4}$$

Meanwhile,

$$d_{T_2}(v_n) = i \tag{2.3.2.5}$$

Now let z be the first vertex in $Z_1 - T_2$ to be numbered. So for all $v \in Z_1, v \neq z$ we have $z <_\pi v$. Let T_3 be the set of vertices numbered before z, i.e. $T_3 = \{v \in V : v <_\pi z\}$. By the assumption of the theorem, v_n comes last in the ordering, so we know $v_n \notin T_3$. Since $z \in Z_1$, by (2.3.2.4) we know

$$d_{T_2}(z) \leq m \tag{2.3.2.6}$$

We have nearly obtained our contradiction. Since z is numbered (and not v_n) at the point when T_3 is the set of numbered vertices we know:

$$d_{T_3}(z) \geq d_{T_3}(v_n) = d_{T_2}(v_n) + d_{T_3-T_2}(v_n) = i + d_{T_3-T_2}(v_n) \tag{2.3.2.7}$$

So we must have

$$d_{T_3}(z) \geq i + d_{T_3-T_2}(v_n) \tag{2.3.2.8}$$

Clearly $d_{T_3}(z) = d_{T_2}(z) + d_{T_3-T_2}(z)$ and we know that $d_{T_2}(z) \leq m$. So substituting this into our previous inequality we get:

$$d_{T_2}(z) + d_{T_3-T_2}(z) \geq i + d_{T_3-T_2}(v_n)$$
$$d_{T_3-T_2}(z) \geq i - d_{T_2}(z) + d_{T_3-T_2}(v_n)$$

and finally

$$d_{T_3-T_2}(z) \geq i - m + d_{T_3-T_2}(v_n) \tag{2.3.2.9}$$

We will obtain a contradiction of Equation 2.3.2.9 by showing that:

$$d_{T_3-T_2}(z) \leq i - 1 - m + d_{T_3-T_2}(v_n) \tag{2.3.2.10}$$

Since we chose z to be the first element of Z_1 to be numbered after w_i, it is clear that $(T_3 - T_2) \cap Z_1 = \emptyset$. Also we know that any vertex which borders z is either in Z or D. Therefore,

$$d_{T_3-T_2}(z) = d_{(T_3-T_2)\cap D}(z) = d_{(T_3-T_2)\cap D_1}(z) + d_{(T_3-T_2)\cap D_2}(z) \tag{2.3.2.11}$$

We know that $|D_2| = i - 1$ and $|T_2 \cap D_2| = m$. So we can assert that

$$d_{(T_3-T_2)\cap D_2}(z) \leq i - 1 - m \tag{2.3.2.12}$$

Furthermore, since $(v_n, v) \in E_G$ for all $v \in D_1$ we know

$$d_{(T_3-T_2)\cap D_1}(z) \leq d_{(T_3-T_2)\cap D_1}(v_n) \tag{2.3.2.13}$$

So we have that $d_{T_3-T_2}(z) \leq i - 1 - m + d_{T_3-T_2}(v_n)$, which contradicts (2.3.2.9) and proves the theorem.

□

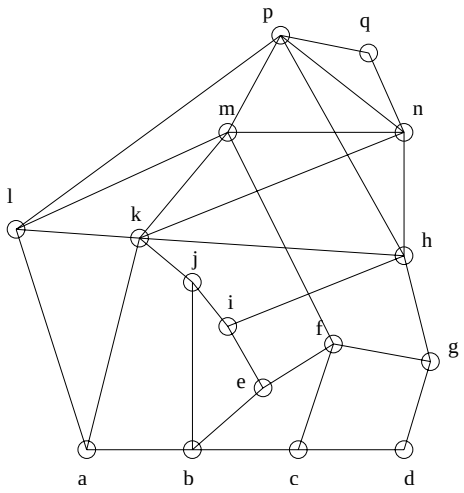

Figure 2.4: A graph

2.4 The MCS lower bound

By our result, any MCS ordering gives us a series of bounds. Choosing the best of these bounds yields the following definition:

Definition 2.4.1. Let the MCS lower bound for a graph G and an MCS ordering π on G be given by:

$$MCSLB_\pi(G) = \max_i d_{T_{i-1}}(v_i)$$

In this way we get a bound for each MCS ordering. Recall that there are many arbitrary choices made in the course of the MCS procedure, and therefore there are many possible MCS orderings. It is generally not feasible to examine all possible MCS orderings, but it is not unreasonable to find a few and see the bounds that arise.

Consider the graph shown in Figure 2.4. There are 16 nodes, many edges, and it is certainly not apparent what the tree width of the graph is. We look for an ordering with a low maximum border and come up with

$$\pi_1 = (d, c, g, f, q, n, p, l, a, k, j, b, e, i, h, m)$$

This ordering has a maximum border of 4 meaning our basic computations will take time $O(16|S|^5)$ using the generalized DP method. Equivalently, if we use this ordering to generate a triangulated graph H (by computing the fill-in), then H will have a 5-clique, giving the same computational complexity. Either way, it is a huge improvement over brute force.

Still we may wonder if we can do better. Maybe there is an ordering with a maximum border of 3 or less, enabling us to reduce our computations further. The bound given in [21] tells us that the tree-width of our graph is greater than or equal to 2, since vertices d and q are non-adjacent vertices, each of degree 2, giving us a lower bound of 2 for $TW(G)$. Looking for large cliques in the graph, we find many 3-cliques, but no 4-cliques, again

telling us that we must incur a border of size 2. But we still don't know if it is possible to find an ordering with a maximum border of exactly 2 or 3, rather than 4.

It is here that the lower bound provided by maximum cardinality search proves useful. Using our result, we know that when you run maximum cardinality search on a graph, the "maximum number of previously numbered neighbors" of an unnumbered vertex is a lower bound for the tree-width of a graph. If at some point in a maximum cardinality search on a graph G, we number a vertex with m previously numbered neighbors, this means that the tree-width of the graph is greater than or equal to m.

We illustrate this bound on our example graph. We arbitrarily number node a first. Now we must next number an unnumbered vertex with a maximum number of neighbors in the set $\{a\}$. In other words, we must choose a neighbor of a so our choices are b, l, k. Arbitrarily we choose b. Next we must choose an (unnumbered) vertex with as many neighbors as possible in the set $\{a, b\}$. Since no vertex borders them both, we can choose any neighbor of either a or b; arbitrarily, we choose c. Suppose, continuing in this fashion, we choose d and then f, so the current set of of numbered vertices is $\{a, b, c, d, f\}$. Now nodes g and e each have two neighbors in this set, while no other (unnumbered) vertex has more than one. So g or e must be numbered next. Furthermore, the existence of an unnumbered vertex with 2 numbered neighbors tells us that the tree-width of our graph must be greater than or equal to 2. As we proceed through the maximum cardinality search, this bound may improve. Indeed, suppose we choose e followed g, i, j, and h. These choices all conform to the maximum cardinality search. Now node k has 3 neighbors in the set $\{a, b, c, d, f, e, g, i, j, h\}$, so we can assert that the tree width of our graph is greater than or equal to three. If we continue by choosing k, l, m, and n, we see that now vertex p has 4 neighbors in the set $\{a, b, c, d, f, e, g, i, j, h, k, l, m, n\}$ which proves that the tree-width of this graph must be at least 4. This means that any ordering of the vertices must incur a border of size at least 4, or equivalently any triangulation must have a clique of size 5. Since we already have an ordering with maximum border 4, we now know it is useless to try to find a better ordering or triangulation. We have managed to calculate that the tree width of our graph is exactly 4.

2.4.1 Properties of the MCS Lower Bound

Of course, our bound is not tight in every case and in fact has some severe limitations. Most glaringly, any MCS lower bound is bounded above by the vertex of highest degree. So in the case of an $n \times n$ lattice, we can't possibly get a lower bound better than 4 (in fact, we always get 2) when the true tree-width is n. So the bound can be arbitrarily weak. On the bright side, the bound always does at least as well as finding the largest clique in G. This is apparent when you consider that the last of the m nodes in an m-clique will have at least $m - 1$ numbered neighbors when it gets numbered. Moreover, as our example has shown, there are non-trivial cases where this bound is tight and informative.

2.5 Improving the Bound

This bound can also be improved in a couple of different ways. The first is by using heuristics in our arbitrary choices. Since different orderings can yield different bounds, we would like to iterate MCS in a fashion that gives us the highest possible lower bounds. This means using a heuristic in our arbitrary choices. Perhaps the first heuristic that comes to mind is: given a number of possible vertices to choose from, choose the one with

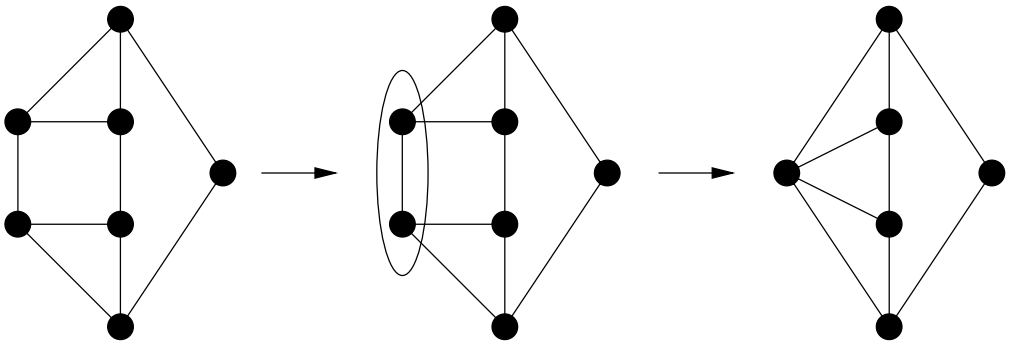

Figure 2.5: An example of edge contraction.

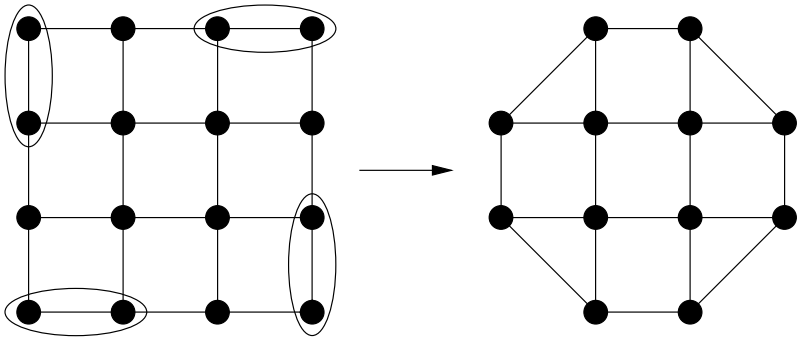

Figure 2.6: Edge contraction of the corners of a lattice

the lowest degree. This gives the higher degree vertices a chance to "accumulate" more neighbors before they are chosen, possibly resulting in a higher bound.

2.5.1 Edge Contraction

Another way to improve the bound is by the use of *edge contraction*. Contracting an edge $e = (v, w)$ on G means to make a new graph G' where v and w are identified as the same vertex (see Figure 2.5). In other words, we combine v and w into one vertex, whose set of neighbors is the union of the neighbors of v and the neighbors of w. To put it precisely, consider this definition where the vertex w is combined into v. We let:

$$V_{G'} \quad = \quad V_G - \{w\}$$

$$\begin{array}{lll} (a,b) \in E_{G'} & \text{if} & (a,b) \in E_G \text{ for } a, b \neq v, w \\ (v,b) \in E_{G'} & \text{if either} & (v,b) \in E_G \text{ or } (w,b) \in E_G \end{array}$$

There are two properties of edge contraction which make it particularly useful in conjuction with the MCS lower bound. First of all, if G' is an edge contraction of G, then $TW(G') \leq TW(G)$. The second is that G' may have vertices of higher degree than G, since the combined vertex accumulates more neighbors.

A clear example of how this may help is illustrated by a simple lattice. It turns out that any MCS ordering on a lattice will have a corner vertex as the last vertex, and have

an MCSLB of exactly 2. So the MCSLB will be just 2 for any MCS ordering on this graph. However, if we consider the graph formed by contracting one edge coming off each of the 4 corner vertices, we see that the resulting graph has minimum degree 3 which implies that the MCSLB for any ordering will now be at least 3. This is illustrated in Figure 2.6.

So one can imagine strategies where an MCS ordering is chosen on a graph, the corresponding lower bound is computed and then an edge is contracted to form a new graph. We can iterate this procedure repeatedly until we have reduced to graph to a single node and keep track of the best bound achieved. One question is: Which edge do we contract? Intuition from the lattice example suggests that the edge we contract should involve the last node in the MCS ordering. So we can arbitrarily choose a neighbor of that node, or use one of several heuristics. Since we know that the MCS lower bound does at least as well as the minimum degree of the graph, this may suggest choosing the neigbor of lowest degree. Note that in general, creating a single node of high degree does not improve the MCS lower bound, as that node will typically be chosen early. It is more important to have many nodes of relatively high degree.

2.5.2 Empirical results

In this section we put forth an algorithm which combines the MCS lower bound with selective edge contraction to provide an improved procedure to lower bound the tree-width of a graph. We also show results of this procedure applied to various classes of graphs. These include lattices, graphs arising from low density parity check codes, and graphs used in other practical examples such as medical networks.

First we present the exact implementation of the algorithm:

1. Randomly choose a starting node v.
2. Perform MCS on your graph G starting with the vertex v. In breaking ties, choose a vertex of lowest degree. In case of a tie for the vertex of lowest degree, randomly choose among the vertices of lowest degree.
3. Let w_1 be the last vertex chosen by MCS. Choose w_2 randomly among the neighbors of w_1 of minimum degree.
4. Contract the edge (w_1, w_2).
5. Iterate steps 2 through 4 until the graph is reduced to a single vertex.
6. Report the highest bound achieved at any point along this process.

It is not difficult to do a rudimentary analysis of the complexity of this algorithm. First node that we start with a graph on n nodes and gradually contract it down to a single node. For each graph in this process, we run one iteration of MCS. To do a single MCS iteration on a graph with m nodes, we maintain a list the unnumbered vertices with the best "score" (i.e. with the most numbered neighbors, breaking ties by smallest degree) and then randomly choose from this list. So the single MCS iteration takes $O(m^2)$ operations on a graph with m nodes. Choosing the edge to contract takes an additional $O(m)$ operations, which can be ignored. Since we iterate MCS n times, each of which costs no more than $O(n^2)$ operations, our overall complexity for this algorithm is $O(n^3)$. In practice, running this algorithm in uncompiled MATLAB on a graph of 500 nodes takes about 5 minutes.

Lattices

Since the inspiration for the edge-contraction came from analyzing MCS on lattices, this is the first class of graphs we consider. The advantage to considering this group of graphs is that we know the answer exactly: an $n \times n$ lattice has tree-width exactly n. However, it also represents a situation where we can expect poor performance from this algorithm. When we begin, we have many vertices, all of degree ≤ 4. Furthermore, we know that even after edge contraction we will not create any large cliques in our graph. Since MCS always does at least as well as finding the largest clique in the graph, if edge contraction yields a large clique, then MCS would do at least that well. However, since the lattice is a planar graph, we know it contains no K_5 minor (where K_5 is the complete graph on 5 vertices). In other words, edge contraction could not possibly yield a clique of size larger than 4. We consider both square lattices and triangular lattices. Examples of both are given in Figure 2.7. Both have actual tree-width of n for an $n \times n$ lattice, although we would expect our bounds to be better on the triangular lattice since it has more edges.

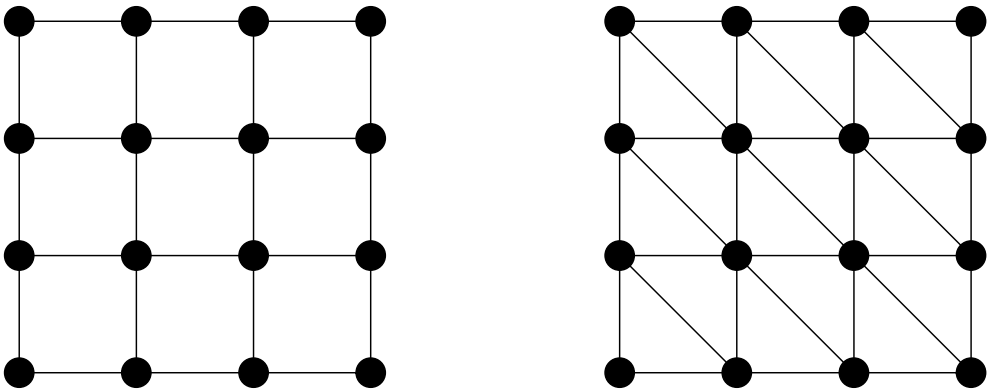

Figure 2.7: Examples of square and triangular lattices for n=4

The following is a table of our results. Recall that there are random choices involved in our algorithm, therefore the lower bounds given here are the best lower bound achieved in 5 trials.

Lattice width	Sq. lattice LB	Tri. Lattice LB
10	5	5
15	5	5
20	5	5
25	5	6

Here we see the results on both square and triangular lattices for various widths n. This clearly demonstrates the weakness of the MCS lower bound. Even with our edge contraction procedure, the bounds barely increase even as we increase the width of the lattice a great deal. However, as stated previously, these graphs represent the worst case of this procedure. On a positive note, these results show that our lower bounding technique is doing something more than just detecting large clique minors. None of the edge contractions has any clique bigger than 4, yet we get bounds of 5 and 6 for these graphs.

2.5.3 Low Density Parity Check code graphs

Recently in the coding theory community, much attention has been paid to Low Density Parity Check (LDPC) code graphs. It has been known since their introduction in 1963 [11] that these codes can theoretically approach the Shannon limit. However, until the recent development of so-called "loopy" belief propagation techniques for decoding, there was no hope of any efficient practical scheme for decoding.

Exact maximum likelihood coding can be viewed as an inference problem on a Markov random field. For a LDPC code, the associated Markov Random field can be described in the following way. Let n be the block length of the code, and m be the number of parity checks. So the rate of the code is $R = 1 - \frac{m}{n}$. Let k be the number of bits involved in each parity check. Equivalently, this is the number of "1"s in each row of the parity-check matrix. Let $1, 2, \ldots, n$ be the n bits in the codeword that is sent over the channel and let $C_1, \ldots, C_m$ be the subset of bits involved in each of the m parity checks. Then $G = (V, E)$ is the associated Markov random field for this problem where:

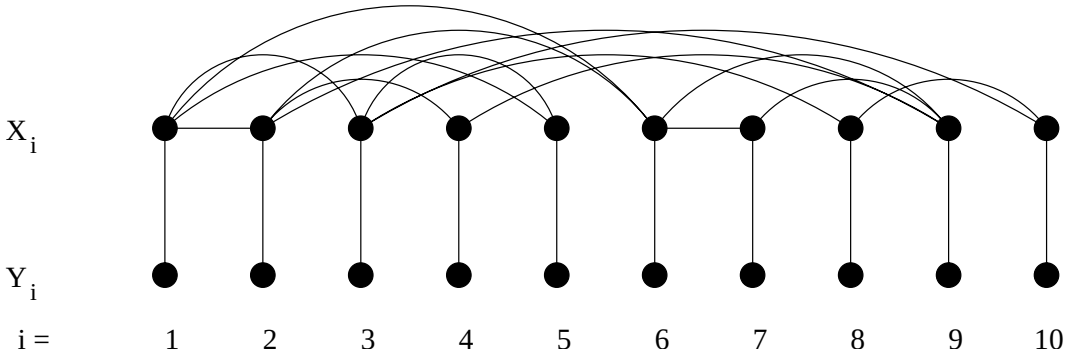

Figure 2.8: An MRF for a LDPC code with n=10, m=5, k=3

$$V = \{X_i : i = 1, \ldots, n\} \cup \{Y_i : i = 1, \ldots, n\}$$
$$(X_i, Y_i) \in E \text{ for } i = 1, \ldots, n$$
$$(X_i, X_j) \in E \text{ if there exists } q \text{ such that } i, j \in C_q$$

The X_i represent the bits sent and the Y_i the bits received. So the Y_i are a corrupted version of the X_i, which is why X_i is adjacent to Y_i. We assume a "memoryless" channel, so the Y_i are conditionally independent of each other given the X_i. Additionally, the bits involved in any parity check must form a clique, which leads to the last condition. An example for $n = 10, m = 5, k = 3$ is shown in Figure 2.8. In this figure the parity checks (cliques) C_q are $\{1, 3, 5\}, \{1, 2, 6\}, \{2, 4, 9\}, \{3, 8, 10\}, \{6, 7, 9\}$.

Exact decoding of these codes involves computing either the most likely configuration (to minimize the word-error rate) or the maximum of the marginal distributions on each individual bit (to minimize the bit-error rate). In either case, we know that the complexity of these problems is $O(n|S|^{TW(G)+1})$. Since the nodes represent bits, $|S| = 2$.

We used our procedure to lower bound graphs of this type for various values of n, m, k. We considered graphs formed by a random choice of the parity checks. In other words, C_j was a random subset of size k from $1, \ldots, n$. The results are in the following table.

n	m	k	TW Lower bound
100	50	6	18
200	100	6	23
300	150	6	27
500	250	6	35
1000	500	6	48
250	125	8	38
500	250	8	53
1000	500	8	71

So from these results we can see, for example, that to perform exact coding on our randomly chosen LDPC graph with $n = 500, m = 250, k = 8$ would require at least $2^{53} \approx 9 \times 10^{15}$ operations, and possibly much more since these are only lower bounds for tree-width. Furthermore, we would expect the tree-width of these graphs to grow linearly

with n (for a given rate R and density k), though this has not been proven. Our lower bound seems to exhibit the linear growth as well. This is some evidence that our bound may be performing reasonably well for this class of graphs.

Specific Graphs from Expert Systems

We obtained 5 graphs that arise from various applications and tested the results of our lower bound. CYC is a subset of a first order logic knowledge base pertaining to spatial relationships. [18] The CPCS graphs come from Bayesian networks related to medical diagnosis. [20] The HPKB graphs are from the High-powered Knowledge Bases project by SRI International related to world politics and affairs. [8]

The exact tree-widths for these graphs are not known, but we list the best known upper bounds. These upper bounds were obtained using the min-degree heuristic [27] [15] which finds an ordering by iterating the process of choosing a node of lowest degree, forming a clique from the neighbors of that node, and removing that node from the graph. These upper bounds were reported in [1].

Graph	Nodes	Edges	TW Upper Bound	TW Lower Bound
CYC1	142	169	13	9
CPCS1	360	1036	20	18
CPCS2	421	1704	23	21
HPKB1	446	2637	36	20
HPKB2	570	3840	40	22

These results show both the power and the limitations of our lower bound algorithm. Combined with a simple heuristic for an upper bound, we are able to determine a narrow range for the tree-widths of the CYC and CPCS graphs. For the HPKB graphs, the upper and lower bounds are farther apart, showing a weakness in either the upper or lower bounding process (or both).

2.6 Conclusions

We have demonstrated equivalent notions of the tree-width of a graph and related these equivalences to the computational equivalence of various methods of inference on graphical models. We then proved a theroem that the maximum cardinality search procedure can be used to get a lower bound on the tree-width of a graph. We gave simple examples to show that this bound can be quite good or quite weak. We then suggested an algorithm which combines maximum cardinality search with edge contraction and wise heuristics to improve the lower bound. This algorithm was implemented and run on lattices, LDPC graphs, and several specific graphs from expert systems. The bounds on the lattices were quite weak, but on graphs used in practice, the bound seemed to perform well. In a couple of cases, the lower bound was within 2 of known upper bounds to the tree-width of the graph. This suggests that practioners of inference on graphical models may find this bound to be a useful tool.

Chapter 3

Complexity Results and Applications for Coarse-to-Fine Dynamic Programming

3.1 Introduction

The previous chapter dealt with issues of complexity that arise from Markov Random fields. We saw the limitations of computing on graphs using standard Dynamic Programming techniques and other equivalent methods. These methods are feasible as long as the tree-width of the underlying graph and the state space at each node is not too large. Precisely, the computational time required is $O(n|S|^{k+1})$, where n is the number of nodes, S is the state space at each node, and k is the tree-width of the underlying graph. In practical applications, the linear dependence on n is rarely a problem, but a large k renders many problems infeasible, as we saw in the previous chapter. Even if $k = 1$ a huge state space can cause $|S|^2$ to be computationally intractable. This is the case with many of the speech recognition systems based on Hidden Markov Models.

Coarse-To-Fine Dynamic Programming (CFDP) is a variation on DP that can be used in various kinds of optimization problems [22]. Specifically, it is applicable to finding the most likely configuration on a Markov random field, and has also been used to numerically solve some calculus of variations problems. In the MRF setting, the idea is that when the state space is large, rather than examine each state individually, we group the states into large groups called superstates. We then define a new MRF using efficiently computable upper bounds for the values of the clique functions across the various state combinations in our superstates. By using standard DP, we find the most likely superstate configuration on this new MRF, and since the state space is small, it will run efficiently. We then use the result of this DP step to make a more "refined" graph. That is, we break some of the superstates into smaller groups and iterate DP again. The surprising fact is that this can be done in a way so that we find the exact most likely configuration. The catch is that the speed of CFDP depends on the structure of the grouping and the nature of the problem. In the best case, CFDP yields a large computational savings over standard DP; in the worst case, it will actually be slower. Furthermore, to implement CFDP, we must be able to efficiently compute a maximum of the clique functions over a subset of states. CFDP has been effectively used in problems from image processing [23], calculus of variations [22], and algebraic coding theory [16].

3.2 Explanation of the method

Before giving a rigorous definition of CFDP, we will demonstrate its use on a simple example. In Figure 3.1 we are given a Markov random field with respect to a simple chain graph. Below it, we have a trellis where each column of nodes represents the different state possibilites for the random variables in the Markov random field. As usual, we assume, for the sake of simplicity, that all 5 variables $X_1, X_2, \ldots, X_5$ take values in the same set S. We have a probability function P defined on $\bar{X} = (X_1, X_2, X_3, X_4, X_5)$. We want to find the most likely configuration of this Markov random field, that is to find:

$$\bar{x}' = \arg\max_{\bar{x} \in S^5} P(\bar{x}) \tag{3.2.3.1}$$

One canonical problem that is solved by dynamic programming is to find the "maximum path" across a trellis. Suppose we are given a trellis such as the one in Figure 3.1 and each of the edges is given a certain value. Using DP, we can efficiently find the path from the leftmost column to the rightmost column which maximizes the sum of the edge

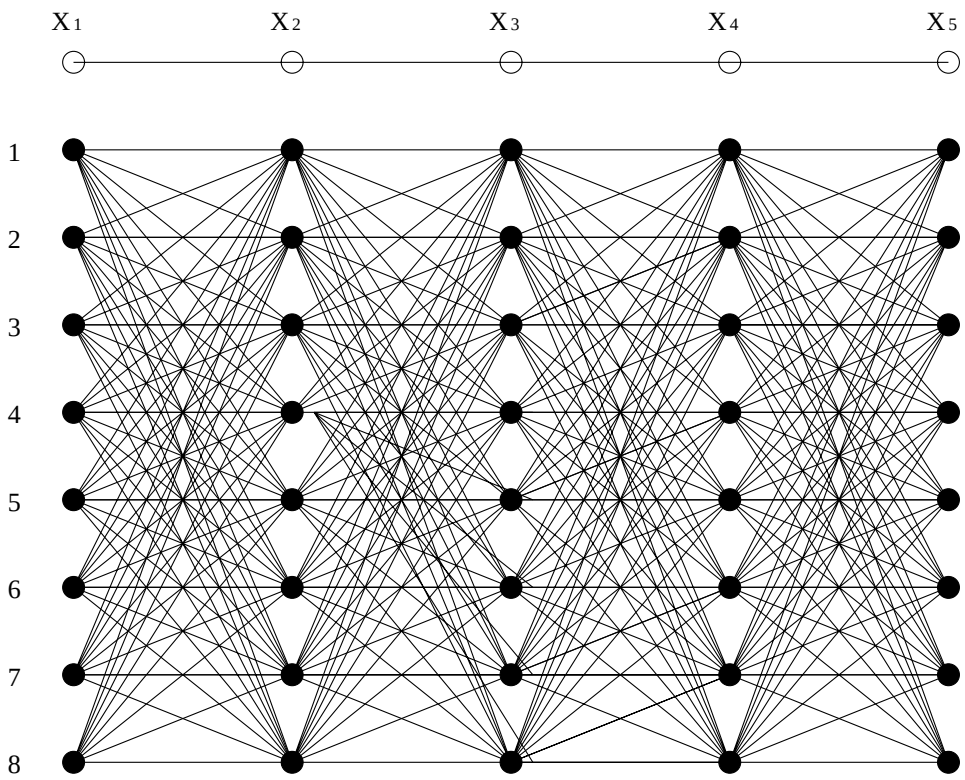

Figure 3.1: A MRF with respect to a chain graph and the associated trellis.

values. It turns out that this is equivalent to finding the most likely configuration of the MRF. We will now show this explicity.

Using elementary probability and our MRF property, we can write the following:

$$P(X_1, X_2, X_3, X_4, X_5) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)P(X_5|X_4)$$

If we let:

$$\begin{aligned} g_1(x_1, x_2) &= P(x_1)P(x_2|x_1) \\ g_2(x_2, x_3) &= P(x_3|x_2) \\ g_3(x_3, x_4) &= P(x_4|x_3) \\ g_4(x_4, x_5) &= P(x_5|x_4) \\ \text{and} & \\ f_i(x_i, x_{i+1}) &= \log g_i(x_i, x_{i+1}) \text{ for } i = 1, 2, 3, 4 \end{aligned}$$

we can write:

$$\begin{aligned}
\bar{x}' &= \arg\max_{\bar{x}\in S^5} P(\bar{x}) \\
&= \arg\max_{\bar{x}\in S^5} \log P(\bar{x}) \\
&= \arg\max_{\bar{x}\in S^5} \log \prod_{i=1}^{4} g_i(x_i, x_{i+1}) \\
&= \arg\max_{\bar{x}\in S^5} \sum_{i=1}^{4} \log g_i(x_i, x_{i+1}) \\
&= \arg\max_{\bar{x}\in S^5} \sum_{i=1}^{4} f_i(x_i, x_{i+1})
\end{aligned}$$

So if we label the edge from node j in column i to node k in column $i+1$ with the value $f_i(j,k)$ then the maximum path corresponds to the most likely configuration under the distribution P.

Solving this problem is a straightforward application of standard DP. For each node in the second column, we consider every node in the first column. At every node in the second column, we store the node in the first column which gives the best path to the second column, and the value of the path to that point. We then move to the third column, and for each node there, consider which node in the second column leads to the best path. Given the second column, the best choice for the third column is independent of the first column so we need not look back further. We continue this process until we complete the last column. At this point we examine every node in the last column to see which ending point gives the highest value. We then backtrack from that node using our stored information to find the path which led to this highest value. The number of operations this requires is $(n-1)S^2$.

Suppose, however, that for any pair of groups of nodes in neighboring columns, we could quickly get the value of the edge of highest value between any two nodes in the two groups. In other words, in our problem we can find:

$$\max_{j\in S_1\subset S; k\in S_2\subset S} f_i(j,k) \tag{3.2.3.2}$$

without having to actually examine every single one of the $|S_1| \times |S_2|$ possibilities. We could then use the following procedure to find the maximum path across the trellis. First, we split the nodes in each column into two groups called "superstates". We then put edges between these superstates and label them with the maximum value of all of the edges connecting any of the nodes contained in the superstates. We then iterate standard DP on this superstate graph, to find the best path across this "optimistic" graph. The value of a path in this superstate trellis is just an upper bound on the value of any single node path through nodes contained in those superstates. So the best path through the superstate trellis is just the region of the original trellis with the best upper bound, using the overestimates defined by the current superstate trellis. We then "refine" along this best path. That is, the superstates that were on this "best" path are divided into two smaller superstates. Now a new graph is formed where each column has 3 superstates instead of 2. We repeat this process of finding paths and refining until we find a path

which is composed entirely of single nodes (once a superstate is divided into single nodes it is divided no further). At that point we have found the best path through the entire original graph, since the value of our single node path is higher than upper bounds for all other possible paths. This process is illustrated in Figure 3.2.

Although we are guaranteed to find the best path through the graph eventually, we have no guarantee in general that we will save time in this process. For example, if the entire graph must be divided into individual nodes before the algorithm terminates, we have spent a great deal of time just to perform standard DP on the original graph as our final step. However, if large chunks of the trellis remain unrefined, we will save time. This is because we avoid examining every possibility in those sections of the trellis. In the best case, we never choose a path of superstates that does not contain the true best path. In that case, we would iterate standard DP $(\log S)$ times and the most expensive of these iterations would take $(n-1)(\log S)^2$ operations. So our computational time will be $O(n(\log S)^3)$ which is considerably less than nS^2. In the worst case, we iterate DP $O(S)$ times, the most expensive of which costs nS^2, giving a time complexity which is $O(nS^3)$, which is clearly worse than nS^2.

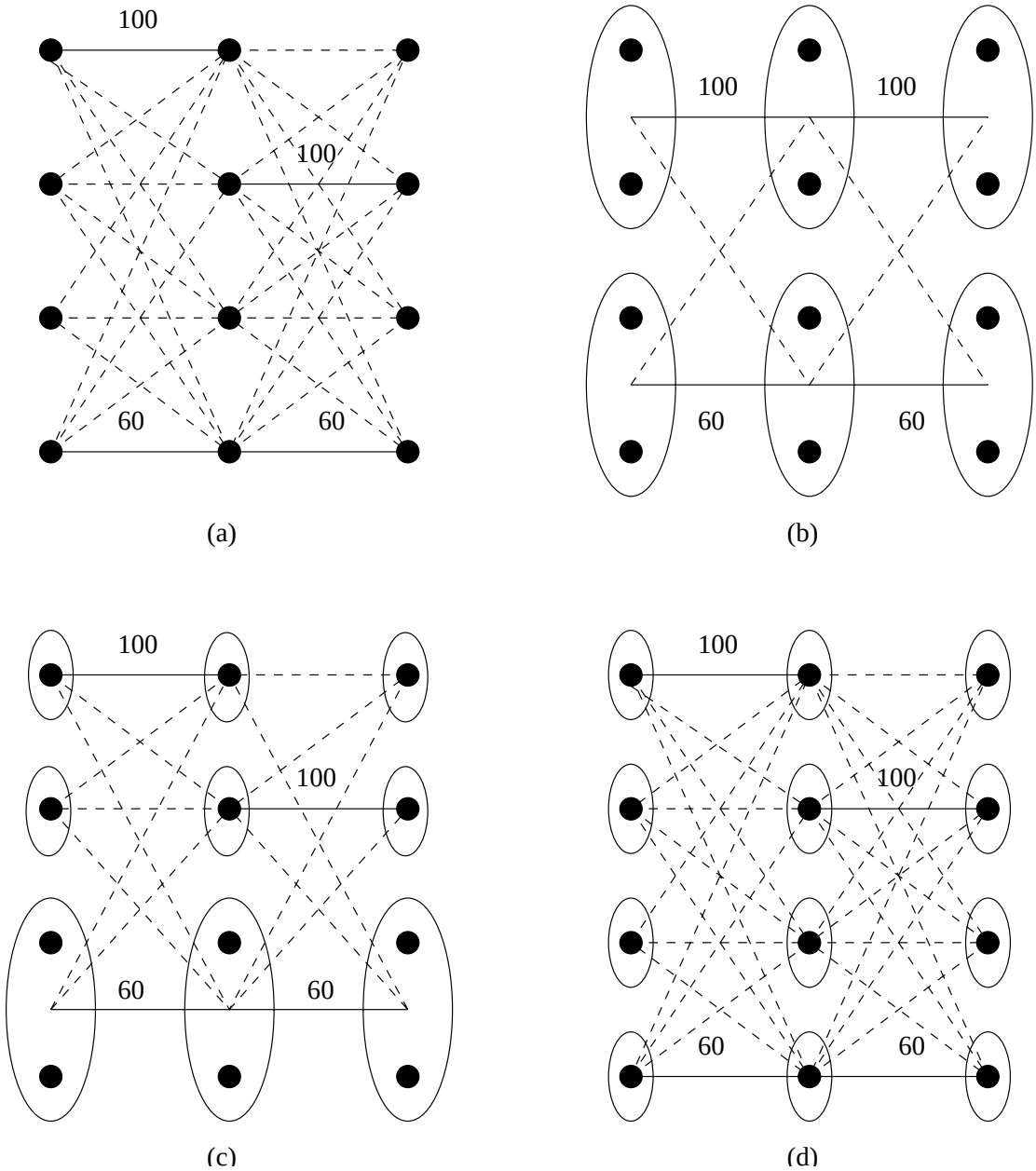

Figure 3.2: Four trellises in various stages of the CFDP process. Dotted edges represent edges with a value of 0.

3.3 Continuous Framework

We would like to know when CFDP will save time over standard DP. From a theoretical perspective, we would like to have sufficient conditions, which if met, would ensure a specific time complexity for CFDP which is significantly less than the time complexity of standard DP for the same problem. Such a theoretical result would yield insight on the types of problems for which CFDP would be a practical method.

The key to the efficiency of CFDP is that the coarse graph must reveal something about the true best path across the trellis. CFDP will save time if there are large areas of the trellis which remain at a coarse resolution. So rather than explore each and every one of these "bad" possibilities, CFDP is able to dismiss them all very quickly. However, CFDP can easily be fooled into searching and resolving an area of the trellis which has, in fact, no good path. Consider using CFDP on the trellis in Figure 3.2(a). On the first iteration it chooses the path across the top half of the trellis, since the optimistic estimate is higher there, as illustrated in part (b) of the figure. The value of the top path in the coarse graph is 200. After it refines along that top path we have the trellis shown in part(c). We find that the two 100 value edges do not actually connect, and the price to switch one of them turns out to be quite large. After part (c) we choose the bottommost path across the trellis, and refine those superstates so that in part (d) we are back to our original trellis configuration, having completely refined all our superstates.

This example is trivial in the sense that we only have 4 nodes in each column and could not expect CFDP to yield much of a savings to begin with. However, it highlights a situation where CFDP may run into trouble. We might expect CFDP to perform better if the nodes that were grouped together were similar in some sense. To be more precise, suppose there were some notion of "distance" between nodes and that edges between nearby nodes were close in value. Examining again the situation in Figure 3.2 we see that a problem arises because two edges from the same nodes to "nearby" nodes (that is, nodes that are grouped together as long as possible) have drastically different values, namely 100 and 0. Thus we have no guarantee that the result at the coarse resolution has anything to do with the true best path. However, if we knew, for example, that the edge values could not differ very much between "nearby edges" (i.e. edges connecting "nearby" nodes) we could be sure that there is a true path somewhat close in value to the "optimistic" estimate.

This is all very intuitive and informal. To make this into a mathematically precise framework, we consider a continuous analog to our generally discrete framework. To begin we choose a *maximum resolution* $R = 2^p$ for some positive integer p. As R gets bigger, we are more closely approximating the true continuous problem. We choose R as a dyadic number since we are using a binary splitting rule (i.e. we divide our superstates in half). In this setting, our discrete columns of nodes are represented by the interval $[0, 1]$, individual nodes, correspond to intervals of some *minimum length* $l = \frac{1}{R}$ and superstates are represented by intervals of a longer length. The edge values between nodes and/or superstates correspond to the maximum of a function f over the intervals. In this way, we have a natural notion of distance between nodes. Furthermore, by putting conditions on our function f we can ensure that if two pairs of nodes are "near" to each other then the edges connecting those pairs of nodes will also be close in value. This setting enables us to make precise statements about the run time of CFDP under certain conditions of the function f. Continuing the correspondence, paths of superstates will correspond to "rectangles" in $[0, 1]^n$, while paths of single states will be rectangles of minimum length.

This will be made precise shortly.

In this paper, we will concern ourselves with how the CFDP run time grows with R. The results are of an asymptotic nature (for R "big enough"). For this reason, we will take n to be a constant of the problem and ignore the dependence of the run time on n. This is reasonable since the issue of large state space or tree width is typically the more important computational issue than the number of nodes in the graph.

3.3.1 Definitions and Notation

We state precisely many definitions that will be required to perform this mathematical analysis. It is useful to keep in mind the correspondences stated in the previous paragraph. For the time being we will restrict ourselves to only simple chain graphs. Later in the paper we will extend these results to general graph structures. This section is very technical, yet this is necessary to create a mathematical framework under which precise statements can be proven.

Most of the mathematics in this paper will take place on the space $[0,1]^n$. In general we will use lowercase letters with a bar over them, such as $\bar{x}$ to refer to points in $[0,1]^n$. So $\bar{x} = (x_1, x_2, \ldots x_n)$. An uppercase letter with a bar over it, such as $\bar{I}$ will refer to a "rectangle" in $[0,1]^n$. For example, $\bar{I} = (I_1, I_2, \ldots, I_n)$ where $I_j = [a_j, b_j]$, a closed interval in $[0,1]$. The n dimensions of our space may be referred to as coordinates, and will usually be indexed by the letter j.

Proceeding with the continuous formalism, we specify a function f

$$f : [0,1]^n \longrightarrow I\!R \tag{3.3.3.1}$$

We assume this function respects our chain graph structure. That is, we assume

$$f(\bar{x}) = f_1(x_1, x_2) + f_2(x_2, x_3) + \ldots + f_{n-1}(x_{n-1}, x_n) \tag{3.3.3.2}$$

where the f_j are functions from $[0,1] \times [0,1] \longrightarrow I\!R$.

With some abuse of notation, we can extend the functions f, f_j to accept rectangles in the following way. First let

$$f_j(I_j, I_{j+1}) = \sup_{x_j \in I_j; x_{j+1} \in I_{j+1}} f_j(x_j, x_{j+1}) \tag{3.3.3.3}$$

Then

$$f(\bar{I}) = f_1(I_1, I_2) + f_2(I_2, I_3) + \ldots + f_{n-1}(I_{n-1}, I_n) \tag{3.3.3.4}$$

Recall that rectangles will represent paths through our trellis. It will be important to quantify the size of a rectangle. We denote the length of an interval I_j by $m(I_j)$. For our purposes, it is most convenient to classify the size of a rectangle by the size of its largest dimension.

Definition 3.3.1. We define the *size* of a rectangle $\bar{I}$ by

$$|\bar{I}| = \max_j m(I_j) \tag{3.3.3.5}$$

If $|\bar{I}| = l (= \frac{1}{R})$ then $\bar{I}$ is a rectangle which represents a choice of a single state for each variable.

Definition 3.3.2. Let the *set of minimum length rectangles* $\mathcal{M} = \mathcal{M}(R)$ be the set of all such rectangles. More precisely,

$$\bar{I} \in \mathcal{M}(R) \Longleftrightarrow I_j = \left[\frac{c_j - 1}{R}, \frac{c_j}{R}\right] \tag{3.3.3.6}$$

where c_j is an integer in $1, 2, \ldots R$ for all $j = 1, 2, \ldots, n$.

For simplicity we assume a unique point maximum $\bar{x}^*$ of f and that (for R big enough) there exists a unique $\bar{I}^*(R) \in \mathcal{M}(R)$ which maximizes $f(\bar{I})$ over all $\bar{I} \in \mathcal{M}(R)$. [1]

Then the solution to our continuous problem can be restated as finding

$$\bar{I}^*(R) = \arg \max_{\bar{I} \in \mathcal{M}(R)} f(\bar{I}) \tag{3.3.3.7}$$

where $\bar{I}^*$ is the unique solution to the problem.

Let us review how our continuous formulation leads to a discrete graph trellis. We specify set of functions f, f_j as in Equation 3.3.3.2. We then specify the maximum resolution $R = 2^p$ for some positive integer p, which implies that our minimum length intervals will be of size $l = \frac{1}{R}$. We then represent each minimum length interval as a node, and let the edge between minimum length intervals in adjacent columns j and $j+1$ be given by the maximum of the function f_j over every possible pair of points in those two intervals. In this way, we have a trellis, and can use standard DP or CFDP to find our answer. For every value of R we have a different trellis, with more nodes as R increases. The question is, how does our run time increase with R. We know that using standard DP, our run time will grow like R^2. How will the CFDP run time grow as we increase R? That question will be answered in this paper, under certain conditions for the functions f_j.

So far, we have defined precisely the continuous analogs to states, superstates, columns of nodes, and edge values. We still need to have a representation for the pattern of our trellis. That is, which superstates exist in which columns at a given iteration of CFDP. To do this, we introduce the notions of *interval partitions, interval decompositions* and *resolution patterns.*

Definition 3.3.3. An *interval partition* P is an ordered set of numbers in the interval $[0, 1]$; $P = \{a_1, a_2, \ldots, a_k\}$ with $0 < a_i < a_{i+1}$ and $a_k = 1$. Define the corresponding *interval decomposition* P' to be the set of k intervals $\{[0, a_1], [a_1, a_2], \ldots, [a_{k-1}, 1]\}$.

The difference between the interval partition and interval decomposition is somewhat semantic. The first is a set of points while the second is a set of intervals.

Definition 3.3.4. A *resolution pattern* $\mathcal{P}$ is a set of n interval partitions; one for each coordinate. $\mathcal{P} = (P_1, P_2, \ldots, P_n)$

We will also define inclusion operations between resolution pattern.

Definition 3.3.5. We will say $\mathcal{P}^* \subset \mathcal{P}$ if $P_j^* \subset P_j^*$ for $j = 1, 2, \ldots, n$. Another way of saying this is that $\mathcal{P}$ is more *refined* than $\mathcal{P}^*$.

[1] This assumption avoids the problems that we face when the true point maximum is a dyadic point (e.g. $x_j^* = \frac{3}{4}$ for all j). When that happens, we often have a singular situation where the rectangles $\bar{I}^{1,a}$ (defined by $I_j^{1,a} = [\frac{3}{4} - \frac{1}{2^a}, \frac{3}{4}]$ for all j) and $\bar{I}^{2,a}$ defined by $I_j^{2,a} = [\frac{3}{4}, \frac{3}{4} + \frac{1}{2^a}]$ for all j) are such that $f(\bar{I}^{1,a}) = f(\bar{I}^{2,a})$ for all a and there is not a unique $\bar{I}^*$ for any R. We make this assumption to avoid dealing with such details.

For our purposes, it is most convenient to judge the size of a resolution pattern by the size of its largest interval partition. In this way we can use the size of the resolution pattern to bound the complexity of a DP iteration on that pattern.

Definition 3.3.6. The *size* of a resolution pattern $\mathcal{P}$, denoted $|\mathcal{P}|$ will be given by

$$|\mathcal{P}| = \max_j |P_j| \tag{3.3.3.8}$$

Definition 3.3.7. The *basic resolution pattern* $\mathcal{P}_b$ is the pattern given by $P_j = \{\frac{1}{2}, 1\}$ for all j.

Definition 3.3.8. For any resolution pattern $\mathcal{P}$ define the corresponding *set of available rectangles* $\mathcal{A}(\mathcal{P})$ in the following way:

$$\bar{I} \in \mathcal{A} \iff I_j \in P'_j \text{ for } j = 1, 2, \ldots, n$$

So $\mathcal{A}$ represents all the possible paths through the graph at the current state of resolution.

We can now state the CFDP procedure in a mathematically precise manner. Given the f_j, f, a starting resolution pattern $\mathcal{P}^1$, and R; CFDP uses the following two-step iterative process to find $\bar{I}^*$. First CFDP finds

$$\bar{I}^i = \arg \max_{\bar{I} \in \mathcal{A}(\mathcal{P}^i)} f(\bar{I}) \tag{3.3.3.9}$$

through standard dynamic programming. Here we are not necessarily guaranteed a unique answer, so we just break ties arbitrarily. If $\bar{I}^i \in \mathcal{M}(R)$ then the algorithm stops and outputs $\bar{I}^* = \bar{I}^i$. Otherwise the resolution pattern $\mathcal{P}^{i+1}$ is formed by adding the midpoint of I^i_j to P^i_j for all j such that $m(I^i_j) > l$. This procedure then repeats until an answer is found.

Lemma 3.3.9. *If* $\bar{I}^i = \arg\max_{\bar{I} \in \mathcal{A}(\mathcal{P}^i)} f(\bar{I})$ *and* $\bar{I}^i \in \mathcal{M}(R)$ *then:*

$$\bar{I}^i = \bar{I}^* = \arg \max_{\bar{I} \in \mathcal{M}(R)} f(\bar{I}) \tag{3.3.3.10}$$

Proof. Take any $\bar{I} \in \mathcal{M}(R)$. There exists a $\bar{I}' \in \mathcal{A}(\mathcal{P}^i)$ such that $\bar{I} \subseteq \bar{I}'$. Therefore $f(\bar{I}') \geq f(\bar{I})$. Furthermore, $f(\bar{I}^i) \geq f(\bar{I}')$ since $\bar{I}^i$ is the $\arg\max_{\bar{I} \in \mathcal{A}(\mathcal{P}^i)} f(\bar{I})$. Therefore $f(\bar{I}^i) \geq f(\bar{I})$ for all $\bar{I} \in \mathcal{M}(R)$. Therefore, it must be the case that $\bar{I} = \bar{I}^* = \arg\max_{\bar{I} \in \mathcal{M}(R)} f(\bar{I})$. □

We can speak of running CFDP to completion, that is until CFDP finds $\bar{I}^*$, or we can just run CFDP for a fixed number of iterations and analyze $\bar{I}^i$ and $\mathcal{P}^i$. We can also specify no minimum length and have CFDP output an infinite sequence of rectangles $\bar{I}^i$.

Note that the resolution patterns created by CFDP have a certain structure. Let $\mathcal{P}$ be the resolution pattern resulting from some number of iterations of CFDP on some problem with the initial resolution pattern $\mathcal{P}_b$ (the basic resolution pattern defined above). For example, consider a particular P_j. If $\frac{3}{8} \in P_j$ then $\frac{1}{2}, \frac{1}{4} \in P_j$ as well. This is a result of the coarse-to-fine progression of our intervals. If $\frac{3}{8}$ is in P_j, that means that the interval

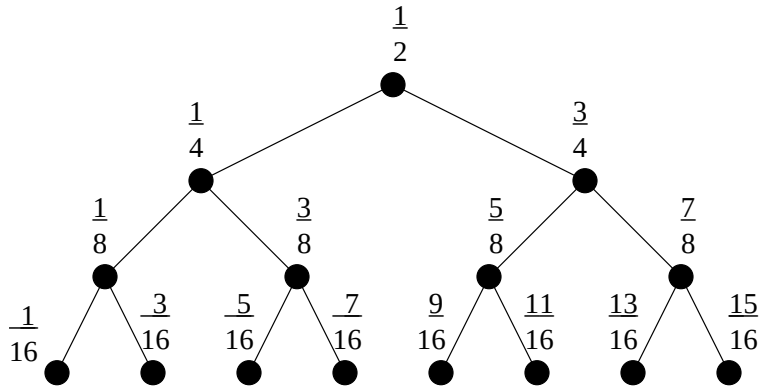

Figure 3.3: The "legality" tree of height 4.

$[\frac{1}{4}, \frac{1}{2}]$ must have been chosen by CFDP, which implies that those points must have been in the interval partition already. In general, for any $x_j \in P_j$, $x_j \neq 0, 1$ we can write $x_j = \frac{2^m c_o}{R}$ for some odd integer c_o. If $\mathcal{P}$ is a resolution pattern created by CFDP from the basic resolution pattern, then P_j will have the following property: if $x_j = \frac{2^m c_o}{R} \in P_j$ then $\frac{2^m(c_o-1)}{R}, \frac{2^m(c_o+1)}{R} \in P_j$. We will now make this definition explicit.

Definition 3.3.10. A resolution pattern $\mathcal{P}$ is said to be *legal* if for all $j = 1, 2, \ldots, n$ we have that if $\frac{2^m c_o}{R} \in \mathcal{P}_j$ for some odd integer c_o then $\frac{2^m(c_o-1)}{R}, \frac{2^m(c_o+1)}{R} \in \mathcal{P}_j$.

Another way to understand legality is to consider the tree in Figure 3.3, where each node is identified with a number. The legality requirement is equivalent to saying that if some number is in P_j than all its ancestors on the "legality" tree must be as well.

Definition 3.3.11. Let $\bar{I}' = \arg\max_{\bar{I} \in \mathcal{A}(\mathcal{P})} f(\bar{I})$. A resolution pattern $\mathcal{P}$ is said to be *sufficient* for a particular problem at a particular resolution R, if $\bar{I}' \in \mathcal{M}(R)$.

Note that by Lemma 3.3.9 we can conclude that if $\mathcal{P}$ is sufficient, then:

$$\bar{I}' = \arg \max_{\bar{I} \in \mathcal{A}(\mathcal{P})} f(\bar{I}) = \arg \max_{\bar{I} \in \mathcal{M}(R)} = \bar{I}^* \tag{3.3.3.11}$$

We can already begin to prove a few basic lemmas about these objects we have defined.

Lemma 3.3.12. *Fix a maximum resolution R. Given two resolution patterns, $\mathcal{P}^1$ and $\mathcal{P}^2$, if $\mathcal{P}^1$ is more refined than $\mathcal{P}^2$ (i.e. $P_j^2 \subseteq P_j^1$ for all j), and $\mathcal{P}^2$ is sufficient, then we have that $\mathcal{P}^1$ is sufficient.*

Proof. Since $\mathcal{P}^2$ is sufficient, we know that $\bar{I}^* \in \mathcal{A}(\mathcal{P}^2)$. Since, by definition $\bar{I}^* \in \mathcal{M}(R)$ we know that $\bar{I}^* \in \mathcal{A}(\mathcal{P}^1)$. So all we need to show is that given any other $\bar{I} \in \mathcal{A}(\mathcal{P}^1)$, $\bar{I} \neq \bar{I}^*$, we have that $f(\bar{I}^*) > f(\bar{I})$. Suppose, by contradiction, we had an $\bar{I} \in \mathcal{A}(\mathcal{P}^1)$, $\bar{I} \neq \bar{I}^*$ such that $f(\bar{I}) \geq f(\bar{I}^*)$. There must exist some $\bar{I}^2 \in \mathcal{A}(\mathcal{P}^2)$ such that $\bar{I} \subseteq \bar{I}^2$. Therefore $f(\bar{I}^2) \geq f(\bar{I}) \geq f(\bar{I}^*)$ which contradicts the sufficiency of $\mathcal{P}^2$. □

Lemma 3.3.13. *Let $\bar{I}^1, \bar{I}^2, \ldots$ be a sequence of rectangles chosen by CFDP. Then for all i, $f(\bar{I}^i) \geq f(\bar{I}^{i+1})$ and $f(\bar{I}^i) \geq f(\bar{I}^*) \geq f(\bar{x}^*)$.*

Proof. Consider $\bar{I}^{i+1} = (I_1^{i+1}, I_2^{i+1}, \ldots, I_n^{i+1})$. There exists some $\bar{I} = (I_1, I_2, \ldots, I_n)$ in $\mathcal{A}(\mathcal{P}^i)$ such that $I_j^{i+1} \subseteq I_j$ for all j. Therefore $f(\bar{I}) \geq f(\bar{I}^{i+1})$, since each maximum is taken over at least as big a set in $\bar{I}$ as it is in $\bar{I}^{i+1}$. But we know that $f(\bar{I}^i) \geq f(\bar{I})$ since $\bar{I}^i$ is the arg max over the set $\mathcal{A}(\mathcal{P}^i)$. Therefore $f(\bar{I}^i) \geq f(\bar{I}^{i+1})$. Since $\bar{I}^*$ is the result of the last iteration, it follows that $f(\bar{I}^i) \geq f(\bar{I}^*)$.

We know that there exists $\bar{I} \in \mathcal{M}$ such that $\bar{x}^* \in \bar{I}$. By definition of f on intervals $f(\bar{I}) \geq f(\bar{x}^*)$. It then follows that $f(\bar{I}^*) \geq f(\bar{I})$ by definition of arg max. □

Suppose that we specify no minimum length and run CFDP to get an infinite sequence of rectangles $\bar{I}^i$. It is shown in [22] that if the f_j are continuous, then $\bar{I}^i$ *converges* to $\bar{x}^*$ in the sense that if $\bar{x}^i$ is any sequence of points satisfying $\bar{x}^i \in \bar{I}^i$ then $\bar{x}^i \to \bar{x}^*$.

3.3.2 Main results (chain graph)

Our goal is to give a limit on the CFDP run time for a particular problem; that is, a particular set of f_j. Here are the two main results we prove for the 1-dimensional "chain graph" setting. They show that, in our continuous framework, we realize a tremendous computational savings when the f_j are linear, and a considerable savings when the f_j "look quadratic" around the maximum of f. Furthermore, these results are tight. That is, there exists problems which meet our assumptions which require the same order of computation that we give as an upper bound.

Theorem 3.3.14. *Suppose the f_j are continuous and piecewise linear, f has a unique point maximum $\bar{x}^*$, and there exists some R_1 such that $R > R_1$ implies $\bar{I}^*$ is unique. Then coarse-to-fine dynamic programming will find the rectangle $\bar{I}^*(R)$ in time $O((\log R)^3)$.*

Theorem 3.3.15. *Suppose the f_j are C^2 and that f has a unique maximum $\bar{x}^*$. Assume further that the Taylor expansion of f at $\bar{x}^*$ has a negative-definite matrix coefficient in the quadratic term. Finally, assume there exists some R_1 such that $R > R_1$ implies $\bar{I}^*$ is unique. Then coarse-to-fine dynamic programming will find the rectangle $\bar{I}^*(R)$ in time $O(R^{\frac{3}{2}})$.*

The proofs to these theorems (in a more general form) are in the Appendix. For now we will just make a few remarks:

1. These bounds are both tight with respect to R. That is, there are f_j that are piecewise linear that require at least $K(\log R)^3$ operations and f_j that meet the conditions of Theorem 3.3.15 which require at least $KR^{\frac{3}{2}}$ operations. The tightness in the piecewise linear case is apparent since even if CFDP is perfect (i.e. only chooses the superstates which contain the true solution), it will take $(\log R)^3$ (plus lower order terms) operations. The tightness for the C^2 case can be easily confirmed analytically in simple cases, take, for example, when $n = 3$, $f_1(x_1, x_2) = x_1 + x_2^2 + 4x_2$, $f_2(x_2, x_3) = -(\frac{5}{2})x_2^2 - 2x_2 + x_3$. So $x_1^*, x_3^* = 1$ and the only interesting aspect of the problem is the behavior of x_2. In this case, elementary calculus can show that any sufficient resolution pattern $\mathcal{P}$ must have $|P_2| > K\sqrt{R}$. This becomes clearer after reading the proof.

2. In practical examples, we are often unable to compute:

$$f_j(I_j, I_{j+1}) = \sup_{x_j \in I_j; x_{j+1} \in I_{j+1}} f_j(x_j, x_{j+1}) \tag{3.3.3.12}$$

exactly. Instead we settle for finding heuristics h_j:

$$h_j(I_j, I_{j+1}) \geq \sup_{x_j \in I_j; x_{j+1} \in I_{j+1}} f_j(x_j, x_{j+1}) \tag{3.3.3.13}$$

and define

$$h(\bar{I}) = h_1(I_1, I_2) + h_2(I_2, I_3) + \ldots + h_{n-1}(I_{n-1}, I_n) \tag{3.3.3.14}$$

As long as $h(\bar{I}) - f(\bar{I}) \leq K|\bar{I}|$ for some constant K and $h(\bar{I}) = f(\bar{I})$ for $\bar{I} \in \mathcal{M}(R)$, then the results of our theorems still hold when using CFDP on the h_j to find the maximum for f.

3. Our setup for CFDP has been geared towards finding maxima. An analogous procedure can be used to find minima instead.

4. We see that in the linear case we get quite a large savings over standard DP – from $O(R^2)$ to $O((logR)^3)$.For the general C^2 case, we get considerable savings, $O(R^{\frac{3}{2}})$ versus R^2. This is potentially a huge savings. Suppose we have a problem where we want $R \approx 10^6 \approx 2^{20}$. For standard dynamic programming we would need 10^{12} operations, that is, a trillion operations. If the problem is piecewise linear, using CFDP we perform the same computation in about 8,000 operations (using $\log_2$). If the problem is just C^2, we still cut the computation to 10^9 operations, from a trillion down to a billion.

3.4 Example – the isoperimetric problem

To illustrate this computational savings in a 1-D setting, we consider a simple calculus of variations problem: the so-called "isoperimetric" problem. The problem is; given a curve of a fixed length, starting at the origin and ending somewhere on the positive x-axis, to enclose the largest area possible between the curve and the x-axis. The answer is simple – a semicircle.

We formulate this as a continuous trellis problem in the following way. We fix the length of our curve, and divide it up into n equal pieces and we let y_j represent the height of the curve above the x-axis after j segments. We form a trellis with $n+1$ columns of nodes, numbering them from 0 to n. Column 0 and column n each have only one point, corresponding to the fact that we must start and end on the x-axis. We can think of these columns as being represented by the interval $[0,0]$. Columns 1 and $n-1$ are represented by the interval $[0,1]$. Since clearly it is inefficient to go below the x-axis, we consider only curves that stay above the axis. Since our segments are of unit length, the highest we could possibly be right after the first segment (or right before the last segment) is 1. Following this logic, columns 2 and $n-2$ are respresented by the interval $[0,2]$ and in general, for $j = 0, 1, 2, \ldots \lfloor \frac{n}{2} \rfloor$, we represent columns j and $n-j$ by the interval $[0,j]$. This differs slightly from the setup in our theorem, where every column of nodes was represented by an equally sized interval, but it is easy to see the the result of Theorem 3.3.14 is still in force.

Note that our progression in the direction of the x-axis (i.e. away from the y-axis) is constrained by our choices of the y_j, since our segments are of unit length. In other words, consider this problem for $n = 6$ and consider the path given by $y_0 = 0, y_1 = 1, y_2 = 2, y_3 = 3, y_4 = 2, y_5 = 1, y_6 = 0$. This path through our trellis corresponds to a curve which goes straight up from the origin to the point $(0,3)$ and then straight back down to the origin, enclosing no area. This is because our segments are of length 1 and therefore if we increase our height (i.e. y-coordinate) by 1 then we do not increase our x-coordinate at all. The path given by $y_0 = 0, y_1 = \frac{\sqrt{2}}{2}, y_2 = \sqrt{2}, y_3 = \frac{3\sqrt{2}}{2}, y_4 = \sqrt{2}, y_5 = \frac{\sqrt{2}}{2}, y_6 = 0$ corresponds to a curve consisting of two linear segments, the first from the origin to $(\frac{3\sqrt{2}}{2}, \frac{3\sqrt{2}}{2})$, and the second from $(\frac{3\sqrt{2}}{2}, \frac{3\sqrt{2}}{2})$ to $(3\sqrt{2}, 0)$. This curve encloses an area of size $\frac{9}{2}$.

We need to define our functions f and f_j to reflect this area function. The value of an edge between columns j and $j+1$ should correspond to the area enclosed under that segment (see Figure 3.4). Using simple geometry, our function f_j is given by:

$$f_j(y_j, y_{j+1}) = \begin{cases} ((y_j + y_{j+1})/2)\sqrt{(1-(y_j - y_{j+1})^2)} & \text{if } |y_j - y_{j+1}| \leq 1, \\ -\infty & \text{otherwise.} \end{cases} \tag{3.4.3.1}$$

Here the root term represents the width of the trapezoidal region we are considering and first term represents the "average" height. We let $f(y_j, y_{j+1}) = -\infty$ when $|y_j - y_{j+1}| > 1$ since it is impossible to change the height by more than 1 between the start and end of a segment.

So if we let:

$$f(\bar{y}) = f(y_0, y_1, \ldots, y_n) = \sum_{j=0}^{n-1} f_j(y_j, y_{j+1}) \tag{3.4.3.2}$$

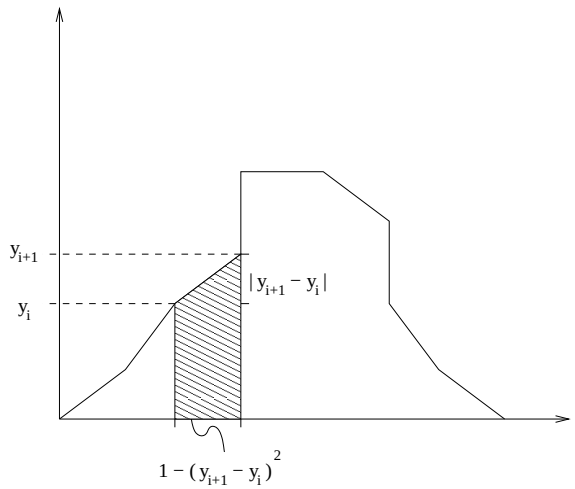

Figure 3.4: The area enclosed by one segment.

then we are looking for $\bar{y}^*$ such that

$$f(\bar{y}^*) = \sup_{\bar{y} \in D} f(\bar{y}) \tag{3.4.3.3}$$

where the domain D is defined by $y_j, y_{n-j} \in [0, j]$ for $j = 0, 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$. Since the domain is compact we know the sup is achieved and $\bar{y}$ is continuous.

Now that we have defined precisely the mathematical problem and shown how it translates into finding the maximum path on a "continuous" trellis, the coarse-to-fine approach should be fairly straightforward. We define a resolution $R = 2^b$ for some integer b and let our minimum length l be given by $l = \frac{1}{R}$. So given R we have a discrete trellis, where the nodes represent intervals of length l and the edge value between two intervals is given by:

$$f_j(I_j, I_{j+1}) = \sup_{y_j \in I_j; y_{j+1} \in I_{j+1}} f_j(y_j, y_{j+1}) \tag{3.4.3.4}$$

Recall that a key practical requirement for implementation of CFDP is that we can efficiently compute the maximum of a function over all the individual state combinations in a superstate. For our problem, this amounts to being able to efficiently compute $f_j(I_j, I_{j+1})$. Fortunately, there is an efficient way to compute this.

It is important first to notice we need only consider y_j, y_{j+1} pairs where at least one of the two is the highest value in its interval. We make this precise in the following lemma.

Lemma 3.4.1. *Let $I_j = [a_j, b_j], I_{j+1} = [a_{j+1}, b_{j+1}]$ and $(y_j, y_{j+1}) \in I_j \times I_{j+1}$. There exists $(y'_j, y'_{j+1}) \in I_j \times I_{j+1}$ such that $f(y'_j, y'_{j+1}) \geq f(y_j, y_{j+1})$ and either $y'_j = b_j$ or $y'_{j+1} = b_{j+1}$.*

Proof. Choose $y_j < b_j$ and $y_{j+1} < b_{j+1}$. If $|y_j - y_{j+1}| > 1$ then $f(y_j, y_{j+1}) = -\infty$ and the lemma is trivial, so assume $|y_j - y_{j+1}| \leq 1$. Let $\epsilon = \min\{b_j - y_j, b_{j+1} - y_{j+1}\}$ and let

$y'_j = y_j + \epsilon$ and $y'_{j+1} = y_{j+1} + \epsilon$. We know that:

$$\begin{aligned} f(y'_j, y'_{j+1}) &= \left(\frac{y'_j + y'_{j+1}}{2}\right)\sqrt{1-(y'_j - y'_{j+1})^2} \\ &= \left(\frac{y_j + y_{j+1} + 2\epsilon}{2}\right)\sqrt{1-(y_j - y_{j+1})^2} \\ &= \left(\frac{y_j + y_{j+1}}{2}\right)\sqrt{1-(y_j - y_{j+1})^2} + \epsilon\sqrt{1-(y_j - y_{j+1})^2} \\ &> f(y_j, y_{j+1}) \end{aligned}$$

□

So we can use the following process to compute $f_j(I_j, I_{j+1})$. Letting $I_j = [a_j, b_j], I_{j+1} = [a_{j+1}, b_{j+1}]$, we compute:

$$\sup_{y_{j+1} \in I_{j+1}} f_j(b_j, y_{j+1})$$

and

$$\sup_{y_j \in I_j} f_j(y_j, b_{j+1})$$

and then choose the larger of those two values.

Using simple calculus, we can compute that $f(b_j, y_{j+1})$, viewed as a function of y_{j+1} only, achieves its maximum at $y_{j+1} = \frac{b_j}{2} + \frac{b_j}{2}\sqrt{1+\frac{2}{b_j^2}}$ (if $b_j = 0$, $y_{j+1} = \frac{\sqrt{2}}{2}$). Furthermore, this is the only critical point of that function in the range $y_{j+1} = [b_j - 1, b_j + 1]$. Since f_j is symmetric in its two arguments, this analysis also applies to the case where $y_{j+1} = b_{j+1}$ and f_j is a viewed as a function of y_j only.

So to compute $\sup_{y_{j+1} \in I_{j+1}} f_j(b_j, y_{j+1})$, we just let

$$y^*_{j+1} = \begin{cases} \frac{b_j}{2} + \frac{b_j}{2}\sqrt{1+\frac{2}{b_j^2}} & \text{if } \frac{b_j}{2} + \frac{b_j}{2}\sqrt{1+\frac{2}{b_j^2}} \in I_{j+1}, \\ b_{j+1} & \text{if } \frac{b_j}{2} + \frac{b_j}{2}\sqrt{1+\frac{2}{b_j^2}} > b_{j+1}. \\ a_{j+1} & \text{if } \frac{b_j}{2} + \frac{b_j}{2}\sqrt{1+\frac{2}{b_j^2}} < a_{j+1}. \end{cases} \tag{3.4.3.5}$$

Then:

$$\sup_{y_{j+1} \in I_{j+1}} f_j(b_j, y_{j+1}) = f_j(b_j, y^*_{j+1}) \tag{3.4.3.6}$$

Analogously we let:

$$y^*_j = \begin{cases} \frac{b_{j+1}}{2} + \frac{b_{j+1}}{2}\sqrt{1+\frac{2}{b_{j+1}^2}} & \text{if } \frac{b_{j+1}}{2} + \frac{b_{j+1}}{2}\sqrt{1+\frac{2}{b_{j+1}^2}} \in I_j, \\ b_j & \text{if } \frac{b_{j+1}}{2} + \frac{b_{j+1}}{2}\sqrt{1+\frac{2}{b_{j+1}^2}} > b_j. \\ a_j & \text{if } \frac{b_{j+1}}{2} + \frac{b_{j+1}}{2}\sqrt{1+\frac{2}{b_{j+1}^2}} < a_j. \end{cases} \tag{3.4.3.7}$$

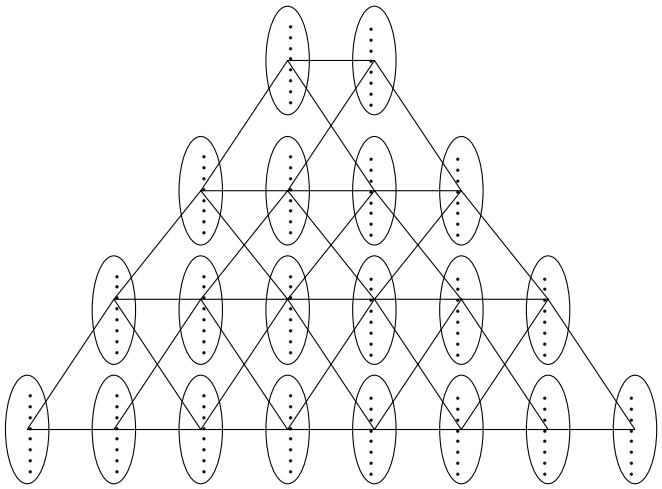

Figure 3.5: The initial trellis for the isoperimetric problem with $n = 8$ and $R = 8$.

so that:

$$\sup_{y_j \in I_j} f_j(b_j, y_{j+1}) = f_j(y_j^*, b_{j+1}) \tag{3.4.3.8}$$

Then by Lemma 3.4.1 we know that:

$$f_j(I_j, I_{j+1}) = \sup_{y_j \in I_j; y_{j+1} \in I_{j+1}} f_j(y_j, y_{j+1}) = \max\{f_j(b_j, y_{j+1}^*), f_j(y_j^*, b_{j+1})\} \tag{3.4.3.9}$$

In this way, we can efficiently compute $f_j(I_j, I_{j+1})$ and thereby implement CFDP.

We have now demonstrated that we have all the tools necessary to use CFDP to solve this problem. We select a value $R = 2^b$ for some integer b to be our resolution. For simplicity, we choose our starting resolution pattern $\mathcal{P}^s$ to be defined by:

$$\mathcal{P}_j^s, \mathcal{P}_{n-j}^s = \{1, 2, \ldots, j\} \text{ for } j = 0, 1, 2, \ldots \left\lfloor \frac{n}{2} \right\rfloor$$

In other words, we start with a resolution pattern where each interval is divided into pieces of length 1. This is shown in Figure 3.5.

3.4.1 Empirical results

We implemented a CFDP solution to this discretized version of the isoperimetric problem using MATLAB. In this way we are able to observe directly how the number of operations increases with R. This allows us to confirm experimentally the predictions of Theorem 3.3.15. The function f for this problem meets the conditions of the theorem except for the discontinuity when the f_j sharply drop to $-\infty$. However, in the proof of Theorem 3.3.15, we see that the continuity of the f, f_j is only necessary within a neighborhood of the maximum. Therefore, we expect to see the behaviour predicted by the theorem.

In Figure 3.6 we see the number of operations taken by CFDP, divided by $R^{\frac{3}{2}}$, plotted against R. If the number of operations grows like $R^{\frac{3}{2}}$, then we would expect to see the normalized operations behave like a constant with respect to R. This appears to be exactly the case.

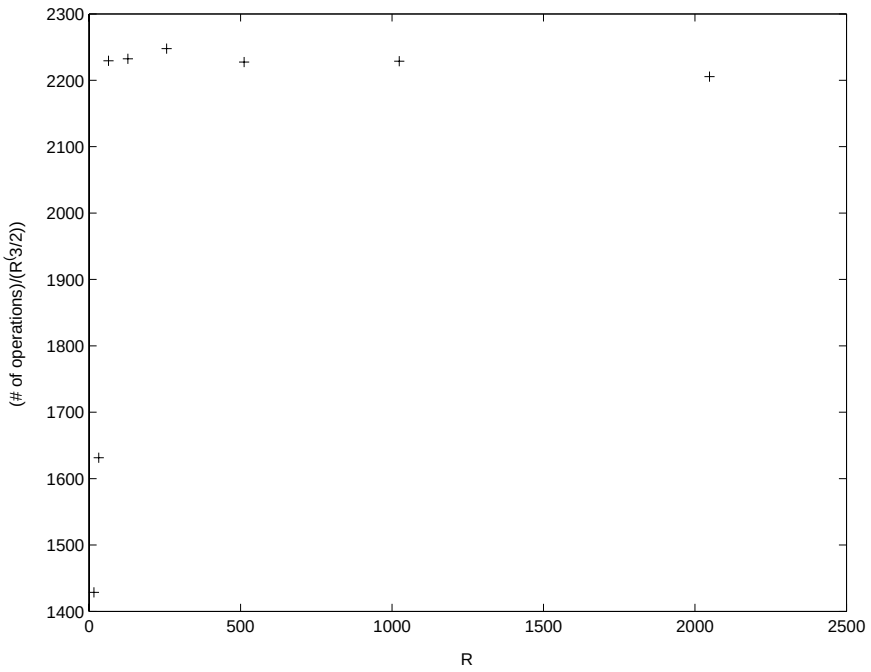

Figure 3.6: Our computational results for the isoperimetric problem.

3.5 General Graph Structures

So far, everything we have discussed has been equivalent to Dynamic Programming on a graph whose underlying structure is a simple chain. However, as we saw in the first chapter, DP can be used for Markov random fields with an arbitrary graph structure. Instead of getting a complexity of $O(R^2)$ as in the chain graph, we get $O(R^{k+1})$ where k is the tree-width of the graph, where R is the size of the state space.

The Coarse-to-Fine Dynamic Programming method, and the continuous framework described earlier also generalizes to the more general graph setting. To make our notation work for the general setting, we must redefine our function f to encompass more complicated functions. We need to introduce cliques as subsets of $\{1, 2, \ldots n\}$. Another difference is that the analogy to finding the maximum path across a trellis is no longer straightforward enough to be useful. Rather we must recall the generalized DP method discussed in Chapter 1 and make our analogies directly to that.

Suppose we have a probability distribution P which respects (i.e. is a Markov Random field with respect to) a graph G with tree-width k. Since G has tree-width k, there exists some k-tree H such that $G \subseteq H$. Since P is clearly respects the graph H, without loss of generality, we can just let G be a k-tree. Furthermore, we can renumber the vertices so that the ordering $\pi = (1, 2, 3, \ldots n)$ has maximum border equal to k. Let $\mathcal{C}$ be the set of maximal cliques in G and let $s = |\mathcal{C}|$. So we can write $\mathcal{C}$ in the form $\mathcal{C} = \{C_1, C_2, \ldots C_s\}$. Since G is a k-tree, our maximal cliques are each of size $k+1$. We represent a clique as a subset of $\{1, 2, \ldots, n\}$. For example, we could let $C_1 = \{1, 3, 4, 5, 7\}$. Given a point $\bar{x}$ we let x_{C_i} represent just the values of x_j for $j \in C_i$. So writing $f_1(x_{C_1})$ would be shorthand for $f_1(x_1, x_3, x_4, x_5, x_7)$. Likewise, given a rectangle $\bar{I}$, we let a *clique interval* I_{C_i} represent the intervals I_j for $j \in C_i$.

Using this notation, we can write:

$$P(x_1, \ldots, x_n) = \prod_{j=1}^{s} g_j(x_{C_j})$$

and if we let:

$$f_j(x_{C_j}) = \log g_j(x_{C_j})$$

we can rewrite P in the form

$$\begin{aligned} \log P(x_1, \ldots, x_n) &= \log \prod_{j=1}^{s} g_j(x_{C_j}) \\ &= \sum_{j=1}^{s} \log g_j(x_{C_j}) \\ &= \sum_{j=1}^{s} f_j(x_{C_j}) \end{aligned}$$

so that maximizing P is equivalent to maximizing the sum of the f_j.

So whereas in the chain graph setting we had:

$$f(\bar{x}) = f_1(x_1, x_2) + f_2(x_2, x_3) + \ldots + f_{n-1}(x_{n-1}, x_n) \tag{3.5.3.1}$$

we have now generalized to

$$f(\bar{x}) = f_1(x_{C_1}) + f_2(x_{C_2}) + \ldots + f_s(x_{C_s}). \tag{3.5.3.2}$$

Note that if our graph is a chain graph then the second formulation reduces to the first.

Continuing the generalization we have:

$$f_j(\bar{I}_{C_j}) = \sup_{x_{C_j} \in I_{C_j}} f(x_{C_j}) \tag{3.5.3.3}$$

$$f(\bar{I}) = \sum_{j=1}^{s} f_j(\bar{I_{C_j}}) \tag{3.5.3.4}$$

$$\tag{3.5.3.5}$$

From the previous chapters, we know that since f respects a graph of tree-width k, we can perform a single DP iteration on a resolution pattern $\mathcal{P}$ in time $O(|\mathcal{P}|^{k+1})$. This enables us to generalize our previous theorems to arbitrary graphs.

Theorem 3.5.1. *Suppose the f_j are continuous and piecewise linear, f has a unique maximum $\bar{x}^*$, and there exists some R_1 such that $R > R_1$ implies $\bar{I}^*$ is unique. Suppose further that f respects a graph with tree-width k. Then coarse-to-fine dynamic programming will find the rectangle $\bar{I}^*(R)$ in time $O((\log R)^{k+2})$.*

Theorem 3.5.2. *Suppose the f_j are C^2 and that f has a unique maximum $\bar{x}^*$. Assume further that the Taylor expansion of f at $\bar{x}^*$ has a negative-definite matrix coefficient in the quadratic term and that f respects a graph of tree-width k. Finally, assume there exists some R_1 such that $R > R_1$ implies $\bar{I}^*$ is unique. Then coarse-to-fine dynamic programming will find the rectangle $\bar{I}^*(R)$ in time $O(R^{\frac{k+2}{2}})$.*

The proofs to these theorems are in the Appendix. Remarks 1-3 from the chain graph versions of the theorems apply to the general version as well. Note that the computational savings over standard DP grow considerably with the tree-width of the graph. In the $k = 1$ case, we demonstrated savings at large $R \approx 10^6$. If the tree-width is larger, the savings become considerable even at smaller R values. Suppose $R = 10^3 \approx 2^{10}$ and $k = 4$. Standard dynamic programming would require 10^{15} computations, out of reach by most standards and certainly unreasonable for any kind of real-time applications. If the problem is piecewise linear, CFDP would require only 10^6 operations. If it is C^2 CFDP would take 10^9 operations. Both of these are tractable by today's standards.

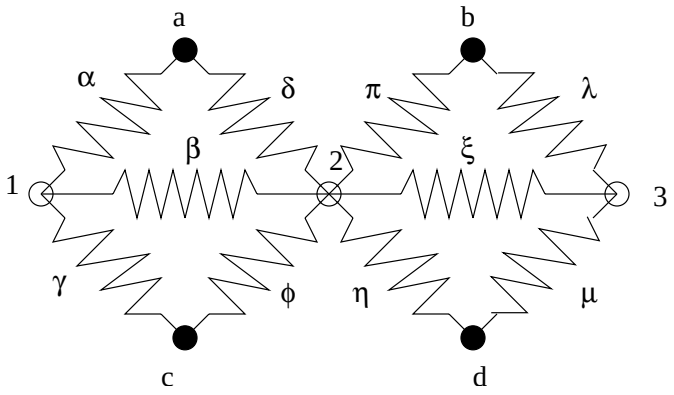

Figure 3.7: A spring network. Black dots are fixed points and white dots are moveable points.

3.6 Multi-dimensional example

To illustrate an application of CFDP on a more complicated graph, we consider a "spring network" problem. This is shown in Figure 3.7. The setup is as follows. There are four fixed points in the plane, labeled a, b, c, d, and represented by black dots in the diagram; and three moveable points, labeled $1, 2, 3$, and represented by white dots in the diagram. A total of 10 springs connect the various points. These springs are labeled by the Greek letters $\alpha, \beta, \gamma, \delta, \phi, \pi, \eta, \xi, \lambda, \mu$. Each spring has a resting length l_s and a spring constant k. The problem is to calculate where the moveable points will come to rest in this system. One way to do this is to consider the equation for the total energy of all the springs, and find the locations of the moveable points which minimize this energy. As we will soon see, this can be fit into a generalized Dynamic Programming, and a CFDP framework.

While heretofore we have considered only maximization problems, our theorems and methods apply analogously to minimization. The energy stored in a spring is $\frac{1}{2}k(l_s - l_{act})^2$, where k is the spring constant, l_s is the length of the spring at rest, and l_{act} is the actual length of the spring. Therefore, if we let, x_i, y_i be the x and y-coordinates of point i (for $i = 1, 2, 3, a, b, c, d$), and redefine our constants k to include the $\frac{1}{2}$ term, then we can express the total energy of our system as:

$$\begin{aligned} f(x_1, y_1, x_2, y_2, x_3, y_3) = k_\alpha(l_\alpha - \sqrt{(x_1 - x_a)^2 + (y_1 - y_a)^2})^2 \\ +k_\beta(l_\beta - \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2})^2 + k_\gamma(l_\gamma - \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2})^2 \\ +k_\delta(l_\delta - \sqrt{(x_a - x_2)^2 + (y_a - y_2)^2})^2 + k_\phi(l_\phi - \sqrt{(x_c - x_2)^2 + (y_c - y_2)^2})^2 \\ +k_\pi(l_\pi - \sqrt{(x_2 - x_b)^2 + (y_2 - y_b)^2})^2 + k_\xi(l_\xi - \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2})^2 \\ +k_\eta(l_\eta - \sqrt{(x_2 - x_d)^2 + (y_2 - y_d)^2})^2 + k_\lambda(l_\lambda - \sqrt{(x_b - x_3)^2 + (y_b - y_3)^2})^2 \\ +k_\mu(l_\mu - \sqrt{(x_d - x_3)^2 + (y_d - y_3)^2})^2 \end{aligned}$$

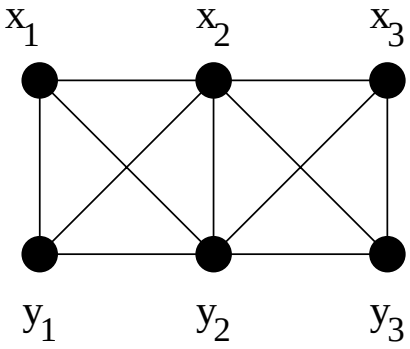

Figure 3.8: The MRF dependendy graph for the spring problem.

or more simply:

$$\begin{aligned} f(x_1, y_1, x_2, y_2, x_3, y_3) = g_\alpha(x_1, y_1) + g_\beta(x_1, y_1, x_2, y_2) + g_\gamma(x_1, y_1) \\ + g_\delta(x_2, y_2) + g_\phi(x_2, y_2) + g_\pi(x_2, y_2) + g_\xi(x_2, y_2, x_3, y_3) \\ + g_{eta}(x_2, y_2) + g_\lambda(x_3, y_3) + g_\mu(x_3, y_3) \end{aligned}$$

So if we let:

$$f_1(x_1, y_1, x_2, y_2) = g_\alpha(x_1, y_1) + g_\beta(x_1, y_1, x_2, y_2) + g_\gamma(x_1, y_1) + g_\delta(x_2, y_2) + g_\phi(x_2, y_2)$$

and

$$f_2(x_2, y_2, x_3, y_3) = g_\pi(x_2, y_2) + g_\xi(x_2, y_2, x_3, y_3) + g_\eta(x_2, y_2) + g_\lambda(x_3, y_3) + g_\mu(x_3, y_3)$$

Then we can write:

$$f(x_1, y_1, x_2, y_2) = f_1(x_1, y_1, x_2, y_2) + f_2(x_2, y_2, x_3, y_3) \tag{3.6.3.1}$$

In other words, f respects the graph G in Figure 3.8.

To make this a generalized DP problem, we need to discretize the state space. We do this, as usual, by choosing a maximum resolution R. However, in this problem there is an additional issue to contend with. In theory, the x_i and y_i could be anywhere on the real line. In practice however, it is not difficult to find a range in which the minimum must lie. For the purposes of our application, we consider only the interval $[0, 1]$ for each variable. We choose the constants of our problem in a way so that this is realistic.

Once again, in order implement CFDP for this problem, we need to be able to efficiently compute:

$$f_1(I_{x_1}, I_{y_1}, I_{x_2}, I_{y_2}) = \inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}; x_2 \in I_{x_2}; y_2 \in I_{y_2}} f_1(x_1, y_1, x_2, y_2)$$

and

$$f_2(I_{x_2}, I_{y_2}, I_{x_3}, I_{y_3}) = \inf_{x_2 \in I_{x_2}; y_2 \in I_{y_2}; x_3 \in I_{x_3}; y_3 \in I_{y_3}} f_3(x_2, y_2, x_3, y_3)$$

In this case, however, finding that inf is difficult, so we will settle instead on using a heuristic:

$$\begin{aligned} h_1(I_{x_1}, I_{y_1}, I_{x_2}, I_{y_2}) &= \inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}} g_\alpha(x_1, y_1) + \inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}; x_2 \in I_{x_2}; y_2 \in I_{y_2}} \\ & \quad g_\beta(x_1, y_1, x_2, y_2) + \inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}} g_\gamma(x_1, y_1) + \inf_{x_2 \in I_{x_2}; y_2 \in I_{y_2}} g_\delta(x_2, y_2) \\ & \quad + \inf_{x_2 \in I_{x_2}; y_2 \in I_{y_2}} g_\phi(x_2, y_2) \\ &\leq \inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}; x_2 \in I_{x_2}; y_2 \in I_{y_2}} f_1(x_1, y_1, x_2, y_2) \end{aligned}$$

with h_2 defined accordingly and $h = h_1 + h_2$.

We discussed earlier that our results still apply to heuristics as long as they are within a linear factor (relative to $|\bar{I}|$) of the true underlying function f, and are equal to f on minimum length intervals. Since f has a continuous first derivative and we are working on a compact space, the first condition is satisfied. Unfortunately, we are unable to compute f directly even on minimum length intervals. If we could compute f directly on minimum length intervals, we could just define h to be equal to f on minimum length intervals and truly claim to be minimizing f. However, in this case, we are truly minimizing h and not f. However, since the difference between the two functions is linear in $\bar{I}$, as R gets big this should make little difference.

To compute our heuristic h, we need to efficiently find:

$$\inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}; x_2 \in I_{x_2}; y_2 \in I_{y_2}} K(l - \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2})^2 \tag{3.6.3.2}$$

In the terms which involve only one moveable point, we just keep the fixed point constant and take the infimum over the two coordinates of the moveable point.

The procedure for finding this infimum is not difficult. First we compute:

$$w_{min} = \inf_{x_1 \in I_{x_1}; x_2 \in I_{x_2}} (x_1 - x_2)^2$$

and

$$w_{max} = \sup_{x_1 \in I_{x_1}; x_2 \in I_{x_2}} (x_1 - x_2)^2$$

This amounts to just finding the inf and sup of $d(x_1, x_2)$ for $x_1 \in I_{x_1}; x_2 \in I_{x_2}$.

Likewise we let:

$$z_{min} = \inf_{y_1 \in I_{y_1}; y_2 \in I_{y_2}} (y_1 - y_2)^2$$

and

$$z_{max} = \sup_{y_1 \in I_{y_1}; y_2 \in I_{y_2}} (y_1 - y_2)^2$$

If we let $I_q = [\sqrt{w_{min} + z_{min}}, \sqrt{w_{max} + z_{max}}]$, then

$$\begin{aligned} \inf_{x_1 \in I_{x_1}; y_1 \in I_{y_1}; x_2 \in I_{x_2}; y_2 \in I_{y_2}} K(l - \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2})^2 &= \inf_{q \in I_q} K(l - q)^2 \\ &= K d(l, I_q)^2 \end{aligned}$$

In this way we can quickly find the required minimum and thereby implement a CFDP solution for this problem.

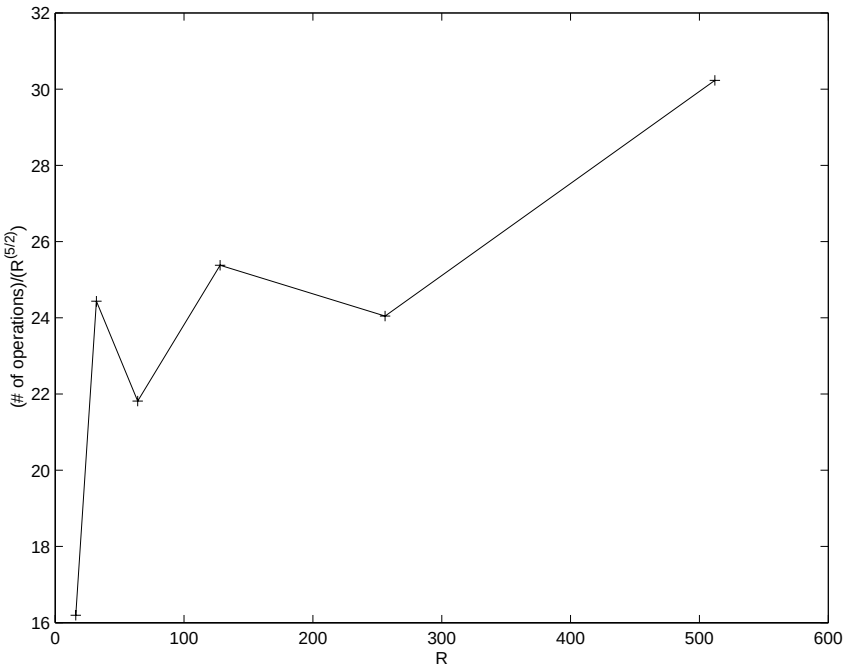

Figure 3.9: The computational results for the spring problem.

3.6.1 Results

We implemented a CFDP solution to the spring network using MATLAB. We confirmed the correctness of our answers by calculating the net force on each point in our solution and confirming that it was 0 (within a reasonable error tolerance given by our interval size). We then analyzed how the number of operations grew with R. According to our theorem, we would expect asymptotically that the number of operations would grow like $R^{\frac{5}{2}}$. So if we scale the number of operations by dividing by $R^{\frac{5}{2}}$ we would expect that number to behave like a constant for large enough R. In Figure 3.9 we plot the scaled number of operations on the y-axis against R on the x-axis. While we do not see the clear constant pattern we saw in the isoperimetric example, this is not totally unexpected. Since this problem requires more operations as a function of R, we were unable to increase R to the extent we did in the previous problem. In this case, however, the improvement over standard DP is remarkable. At $R = 512$, standard DP would take $2 \times (512^2) \approx 1.37 \times 10^{11}$ operations. CFDP took about 1.8×10^8 operations, roughly 766 times faster.

APPENDIX

In this Appendix we prove the main results of Chapter 3. These proofs are fairly technical in nature, but this is necessary for mathematical precision. We will give some necessary background lemmas and propositions, before restating and proving our theorems. Our notation follows from the general framework presented in Section 3.5.

Background Definitions, Lemmas, Propositions

We will use $d(.,.)$ to represent the distance function in several ways depending on the types of arguments. Generally, the definitions are what one would expect, but we will make them explicit here for clarity. If $x_j, y_j \in I\!R$, then:

$$d(x_j, y_j) = |x_j - y_j| \tag{A-1}$$

We define the distance between a real number x_j and an interval of reals I_j by

$$d(x_j, I_j) = \inf_{y_j \in I_j} d(x_j, y_j) \tag{A-2}$$

The distance between two points $\bar{x}, \bar{y} \in I\!R^n$ (or $[0,1]^n$) will be the simple Euclidean distance:

$$d(\bar{x}, \bar{y}) = \left(\sum_{j=1}^{n} (x_j - y_j)^2 \right)^{\frac{1}{2}} \tag{A-3}$$

Finally, the distance between a point $\bar{x} \in I\!R^n$ (or $[0,1]^n$) and a rectangle $\bar{I} \subseteq I\!R^n$ (or $[0,1]^n$) is given by:

$$d(\bar{x}, \bar{I}) = \inf_{\bar{y} \in I} d(\bar{x}, \bar{y}) \tag{A-4}$$

Let the *CFDP overestimate for* $\bar{I}$ be given by

$$OE(\bar{I}) = f(\bar{I}) - \sup_{\bar{x} \in \bar{I}} f(\bar{x}) \tag{A-5}$$

Thus it is easy to see that if

$$f(\bar{x}^*) > f(\bar{x}) + OE(\bar{I}) \tag{A-6}$$

for all $\bar{x} \in \bar{I}$ then $f(\bar{x}^*) > f(\bar{I})$.

Lemma .0.1. *A resolution pattern $\mathcal{P}$ is sufficient (for a resolution R) if for every rectangle $\bar{I} \in \mathcal{A}(\mathcal{P})$ at least one of the following is true:*

1. $\bar{I} \in \mathcal{M}(R)$
2. $f(\bar{x}^*) > f(\bar{I})$

Proof. Let $\bar{I}' = \arg\max_{\bar{I} \in \mathcal{A}(\mathcal{P})} f(\bar{I})$. For any $\bar{I} \notin \mathcal{M}(R)$, $f(\bar{I}) < f(\bar{x}^*) \leq f(\bar{I}^*)$. So $\bar{I}' \in \mathcal{M}(R)$ and $\mathcal{P}$ is sufficient. □

Proposition .0.2. *Fix a set of f_{C_j} which respects a graph of tree-width k and suppose that for some sequence of resolutions $R \to \infty$ we have a legal, sufficient resolution pattern $\mathcal{P}(R) = \{P_1(R), P_2(R), \ldots, P_n(R)\}$. If*

$$|\mathcal{P}(R)| = O(g(R)) \tag{A-7}$$

then CFDP will run to completion in time $O((g(R))^{k+2})$.

Proof. The proof of the proposition rests on the following lemma:

Lemma .0.3. *Suppose we have a sufficient resolution pattern $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ and another resolution pattern $\mathcal{P}^1 = \{P_1^1, P_2^1, \ldots, P_n^1\}$ which is not sufficient. Let $\mathcal{P}^2$ be the resolution pattern yielded by CFDP after one iteration on the resolution pattern $\mathcal{P}^1$. Then there exists some number $y \in [0,1]$ and some coordinate j such that $y \in P_j^2$ and $y \in P_j - P_j^1$.*

Proof. Let $\bar{I}$ be the rectangle chosen on the first iteration of CFDP on the resolution pattern $\mathcal{P}^1$. Since $\mathcal{P}^1$ is not sufficient, there exists at least one j such that $m(I_j) > l$. Let $J = \{j : m(I_j) > l\}$ and let $y_j =$the midpoint of I_j for $j \in J$. Clearly, for $j \in J$, $y_j \in P_j^2$ and $y_j \notin P_j^1$. So we need to show that for at least one $j \in J$, $y_j \in P_j$. Suppose, by contradiction, that for all $j \in J$ we have $y_j \notin P_j$. We will construct a rectangle $I_j^0 \in \mathcal{A}(P)$ in the following way. For $j \in J$, let I_j^0 be the interval in P_j', the interval partition corresponding to P_j, which contains y_j, so that $y_j \in I_j^0$. Since our resolution patterns are legal, it is not possible that $I_j^0 \subset I_j$, so we know that $I_j \subseteq I_j^0$.

For $j \notin J$, $m(I_j) = l$, so let I_j^0 be the interval in P_j' which contains I_j. By our construction, $\bar{I}^0 \in \mathcal{A}(P)$. We know that there exists a rectangle $\bar{I}^* \in \mathcal{A}(P)$ such that $\bar{I}^* \in \mathcal{M}(R)$ and $f(\bar{I}^*) > f(\bar{I})$ for $\bar{I} \in \mathcal{A}(P), \bar{I} \neq \bar{I}^*$. We know $\bar{I}^0 \notin \mathcal{M}(P)$ so therefore $\bar{I}^* \neq \bar{I}^0$. Therefore $f(\bar{I}^*) > f(\bar{I}^0)$. Since $\bar{I} \subseteq \bar{I}^0$ we know that $f(\bar{I}^0) \geq f(\bar{I})$. Thus we have that $f(\bar{I}) < f(\bar{I}^*)$ which contradicts Lemma 3.3.13 .

□

Starting from the basic resolution pattern and applying the lemma repeatedly we see that after $nO(g(R))$ DP iterations, if we have not already solved the problem, we will have a resolution pattern $\mathcal{P}^*$ which is at least as refined as $\mathcal{P}$. In other words, for all j, $P_j \subseteq P_j^*$. Since any refinement of a sufficient resolution pattern is itself sufficient (Lemma 3.3.12), the problem will be solved on the next iteration. So we have used $nO(g(R))$ DP iterations, the most expensive of which takes time $O(g(R)^{k+1})$. Thus we get a complexity of $O(g(R)^{k+2})$.

□

Theorem for Piecewise Linear functions

Theorem .0.4. *Suppose the f_j are continuous and piecewise linear, f has a unique maximum $\bar{x}^*$, and there exists some R_1 such that $R > R_1$ implies $\bar{I}^*$ is unique. Suppose further that f respects a graph with tree-width k. Then coarse-to-fine dynamic programming will find the rectangle $\bar{I}^*(R)$ in time $O((\log R)^{k+2})$.*

Proof. All the proofs of our main results will follow this basic outline. First, we carefully define a neighborhood of $\bar{x}^*$, called G. This region will have the property that CFDP never

chooses more that a constant number of rectangles outside of G, independent of R. In other words, to determine the number of iterations required by CFDP asymptotically, we need only consider how f behaves in the region G. We use this set G to define a procedure such that given any "big enough" resolution R, we can define a sufficient resolution pattern $\mathcal{P}$. We then use Proposition .0.2 to relate the size of the resolution pattern (as a function of R) to the complexity of the problem with respect to CFDP.

We begin in the piecewise linear setting. Since f is piecewise linear, and $[0,1]^n$ is compact, we can find a constant $K_1 > 0$ such that:

$$K_1 > \sup\{\left|\frac{\partial f_i}{\partial x_j}(\bar{x})\right|, \left|\frac{\partial f}{\partial x_j}(\bar{x})\right|\} \tag{A-8}$$

where the max is taken over all $i \in 1, 2, \ldots s$, all $j \in 1, 2, \ldots, n$ and all $\bar{x} \in [0,1]^n$ where the partial derivatives exist.

Lemma .0.5. *Recall that we defined $OE(\bar{I}) = f(\bar{I}) - \sup_{\bar{x}\in\bar{I}} f(\bar{x})$. Under the assumptions of our theorem we have:*

$$OE(\bar{I}) \leq s(k+1)K_1|\bar{I}| \tag{A-9}$$

Proof. Let $\bar{x}' = \arg\max_{\bar{x}\in\bar{I}} f(\bar{x})$. Using this definition and expanding f we get:

$$\begin{aligned}
OE(\bar{I}) &= \left(\sum_{j=1}^{s} f_j(\bar{I}_{C_j})) - f_j(\bar{x}'\right) \\
&= \sum_{j=1}^{s} \left(f_j(\bar{I}_{C_j}) - f_j(\bar{x}'_{C_j})\right) \\
&= \sum_{j=1}^{s} \left(f_j(I_{j_1}, I_{j_2}, \ldots, I_{j_{k+1}}) - f_j(x'_{j_1}, x'_{j_2}, \ldots, x'_{j_{k+1}})\right) \\
&= \sum_{j=1}^{s} \left(\sup_{x_{j_1}\in I_{j_1}; \ldots; x_{j_{k+1}}\in I_{j_{k+1}}} f_j(x_{j_1}, x_{j_2}, \ldots, x_{j_{k+1}}) - f_j(x'_{j_1}, x'_{j_2}, \ldots, x'_{j_{k+1}})\right) \\
&< \sum_{j=1}^{s} \left(K_1|\bar{I}|(k+1)\right) \\
&= s(k+1)K_1|I|
\end{aligned}$$

□

Let $D_f(\bar{x}, \theta)$ be the directional derivative of the function f at $\bar{x}$ in the direction θ , for $\theta \in S_{n-1}$ (the unit sphere). Let $q(\theta) = D_f(\bar{x}^*, \theta)$. Since $\bar{x}^*$ is a unique maximum and f is piecewise linear we know that $q(\theta)$ is bounded away from 0. Therefore, we can find a constant $K_2 > 0$ such that:

$$K_2 < \min_{\theta\in S_{n-1}} |D_f(\bar{x}^*, \theta)|. \tag{A-10}$$

We are now ready to begin constructing our set G. Our strategy is to let $G = G(\Delta)$ be given by:

$$\bar{x} \in G \iff x_j \in [x_j^* - \Delta, x_j^* + \Delta] \text{ for all } j = 1, 2, \ldots, n \tag{A-11}$$

Then our goal is to find a Δ and ϵ' small enough so that G satisfies the following properties:

1. $\bar{x} \notin G \Rightarrow f(\bar{x}) < f(\bar{x}^*) - \epsilon'$
2. $\bar{x} \in G - \{\bar{x}^*\} \Rightarrow f(\bar{x}) < f(\bar{x}^*) - K_2 d(\bar{x}, \bar{x}^*)$

That such a set G is possible follows more or less directly from the fact that f is continuous, piecewise linear, and has a unique maximum. For those who wish to see the details of construction, they are included below. However, the section is quite technical without being particularly enlightening. If you wish, you may proceed directly to Lemma .0.6.

Let $H(\epsilon)$ be the subset of $[0,1]^n$ such that $\bar{x} \in H(\epsilon)$ if and only if $f(\bar{x}) > f(\bar{x}^*) - \epsilon$. We define

$$\epsilon_1 = \sup\{\mu > 0 : H(\epsilon) \text{ is a connected set for all } 0 < \epsilon < \mu\} \tag{A-12}$$

So one property of our Δ should be that $G(\Delta) \subseteq H(\epsilon)$ for some $\epsilon < \epsilon_1$.

We then choose a number δ_1 in the following way:

$$\delta_1 = \sup\{\delta : D_f(\bar{x}, \theta) = D_f(\bar{x}^*, \theta) \text{ (for all } \theta \in S_{n-1}) \text{ for all } \bar{x} \in B_\delta(\bar{x}^*)\} \tag{A-13}$$

where $B_\delta(\bar{x})$ is a δ-sized ball around the point $\bar{x}$.

So, as long as we are inside of $B_{\delta_1}(\bar{x}^*)$, we can ensure that the second property listed above holds for G.

Our procedure for choosing Δ then goes in this way. We choose Δ such that

1. $G(\Delta) \subseteq H(\epsilon_1)$
2. $\Delta < \frac{\delta}{\sqrt{n}}$ (i.e. $G(\Delta) \subseteq B_{\delta_1}(\bar{x}^*)$)

We then choose ϵ' such that $H(\epsilon') \subseteq G(\Delta)$. By our construction, both required properties for G are satisfied.

Lemma .0.6. *If $\bar{I} \cap G = \emptyset$ and $|I| \leq \frac{\epsilon'}{s(k+1)K_1}$ then $f(\bar{x}^*) > f(\bar{I})$.*

Proof. Since $\bar{I} \cap G = \emptyset$, we know that for any $\bar{x} \in \bar{I}$ that

$$f(\bar{x}) < f(\bar{x}^*) - \epsilon_1 \tag{A-14}$$

So we have

$$\begin{aligned} f(\bar{I}) &= \max_{\bar{x} \in I} f(\bar{x}) + OE(\bar{I}) \\ &\leq f(\bar{x}^*) - \epsilon' + OE(\bar{I}) \\ &\leq f(\bar{x}^*) - \epsilon' + s(k+1)K_1|\bar{I}| \\ &\leq f(\bar{x}^*) - \epsilon' + (s(k+1)K_1)\frac{\epsilon'}{s(k+1)K_1} \\ &\leq f(\bar{x}^*) \end{aligned}$$

□

Lemma .0.7. *If $\bar{I} \cap G \neq \emptyset$ and $|I| \leq \min\{\frac{K_2}{s(k+1)K_1} d(\bar{x}^*, \bar{I}), \frac{\epsilon'}{s(k+1)K_1}\}$ then $f(\bar{x}^*) > f(\bar{I})$.*

Proof. Since $\bar{I} \cap G \neq \emptyset$, we know that

$$\bar{x} \in \bar{I} \Rightarrow f(\bar{x}) < f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I}), \epsilon'\}$$

Letting $K_3 = s(k+1)K_1$ we have:

$$\begin{aligned} f(\bar{I}) &= \max_{\bar{x} \in \bar{I}} f(\bar{x}) + OE(\bar{I}) \\ &\leq f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I}), \epsilon'\} + OE(\bar{I}) \\ &\leq f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I}), \epsilon'\} \ + s(k+1)K_1|\bar{I}| \\ &\leq f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I}), \epsilon'\} + K_3 \min\{\frac{K_2}{K_3} d(\bar{x}^*, \bar{I}), \frac{\epsilon'}{K_3}\} \\ &\leq f(\bar{x}^*) \end{aligned}$$

□

Using this set G, we will now construct a resolution pattern (as a function of R) which satisfies the properties of Lemma .0.1. Then, by counting the size of the resolution pattern as a function of R, we can use Proposition .0.2 to calculate the complexity of this class of problems with respect to CFDP.

Recall that a resolution pattern $\mathcal{P}$ consists of n interval partitions P_j. We will give a process to create an interval partition for arbitary j which will yield our resolution pattern $\mathcal{P}$. Specifically, we will create a sequence of interval partitions $P_j^0, P_j^1, P_j^2, \ldots, P_j^m$. We then create one more set, called Q_j to ensure the legality of our eventual partition and then let $P_j = \left(\bigcup_{i=1}^m P_j^i\right) \cup Q_j$. It is useful to keep in mind that we are creating a partition such that every rectangle $\bar{I} \in \mathcal{A}(\mathcal{P})$ will either have $f(\bar{I}) < f(\bar{x}^*)$ or $|\bar{I}| = l$.

With this in mind, our first goal is to ensure that any rectangle $\bar{I} \in \mathcal{A}(\mathcal{P})$ such that $\bar{I} \cap G = \emptyset$ has the property that $|\bar{I}| \leq \frac{\epsilon'}{s(k+1)K_1}$. In fact, we would like all $\bar{I} \in \mathcal{A}(\mathcal{P})$ to have this property. This is accomplished quite simply. Choose the smallest integer a such that $\frac{1}{2^a} < \frac{\epsilon'}{s(k+1)K_1}$. Now we let:

$$P_j^0 = \{\frac{1}{2^a}, \frac{2}{2^a}, \frac{3}{2^a}, \frac{4}{2^a}, \ldots, \frac{2^a}{2^a}\} \tag{A-15}$$

This step alone ensures that when we create our resolution pattern $\mathcal{P}$, that every $\bar{I} \in \mathcal{A}(\mathcal{P})$ has $|\bar{I}| < \frac{\epsilon'}{s(k+1)K_1}$. Therefore, every $\bar{I}$ such that $\bar{I} \cap G = \emptyset$ will have the property that $f(\bar{I}) < f(\bar{x}^*)$.

Now our goal is to make our resolution pattern refined enough so that every $\bar{I}$ such that $\bar{I} \cap G \neq \emptyset$ either has $|\bar{I}| = l$ (i.e. so $\bar{I} \in \mathcal{M}(R)$) or has the property that $|\bar{I}| < \frac{K_2}{s(k+1)K_1} d(\bar{x}^*, \bar{I})$. If the latter property holds, since we are already assured that $\bar{I}| < \frac{\epsilon'}{s(k+1)K_1}$, we can use Lemma .0.7 to conclude that $f(\bar{I}) < f(\bar{x}^*)$.

To simplify our notation we will let $C_1 = \frac{K_2}{s(k+1)K_1}$. So our goal is to ensure that $|\bar{I}| \leq \max\{C_1 d(\bar{x}^*, \bar{I}), l\}$. To do this, we construct our interval partitions in such a way so that if an interval $I_j \in P_j'$ (the corresponding interval decomposition) and $|I_j| = 2^y l$, then

we must have $d(x_j^*, I_j) \geq \frac{2^y}{C_1} l$. Informally, a region of size $\frac{2}{C_1} l$ on either side of x_j^* must be "broken up" into pieces of size l, a region of size $\frac{2^2}{C_1} l$ on either side of x_j^* must be broken up into pieces of size no bigger than $2l$, and in general a region of size $\frac{2^i}{C_1} l$ on either side on x_j^* must be broken up into pieces of size no bigger than $2^{i-1} l$. This must be done to include all of the region G, that is for $i = 1, 2, \ldots, m$ where m is the smallest integer such that $\frac{2^m}{C_1} l > \Delta$.

Lets try to make this precise. This section is very technical because we must take special care to ensure that we have a *legal* resolution pattern at the end of the process. We let:

$$a_1 = \text{ largest integer } w \text{ divisible by 2 such that } \frac{w}{R} < x_j^* - \frac{2}{C_1} l$$
$$b_1 = \text{ smallest integer } w \text{ divisible by 2 such that } \frac{w}{R} > x_j^* + \frac{2}{C_1} l$$

then

$$P_j^1 = \{a_1, a_1 + l, a_1 + 2l, a_1 + 3l, \ldots, b_1 - 3l, b_1 - 2l, b_1 - l, b_1\}$$

This step defines the region around $\bar{x}^*$ such that all rectangles $\bar{I} \in \mathcal{A}(\mathcal{P})$ in that area must have $|\bar{I}| = l$. Precisely, with these points in our resolution pattern, any rectangle $\bar{I}$ which has $d(\bar{x}^*, \bar{I}) \leq \frac{2}{C_1} l$ will have $|\bar{I}| = l$. Choosing a_1 and b_1 in that manner ensures the legality of our final resolution pattern.

In the next step, we define the limits where any rectangle $\bar{I} \in \mathcal{A}(\mathcal{P})$ must have $|\bar{I}| \leq 2l$.

$$a_2 = \text{ largest integer } w \text{ divisible by 4 such that } \frac{w}{R} < x_j^* - \frac{4}{C_1} l$$
$$b_2 = \text{ smallest integer } w \text{ divisible by 4 such that } \frac{w}{R} > x_j^* + \frac{4}{C_1} l$$

then

$$P_j^2 = \{a_2, a_2 + 2l, a_2 + 4l, a_2 + 6l, \ldots, a_1 - 6l, a_1 - 4l, a_1 - 2l, a_1\} \cup \{b_1, b_1 + 2l, b_1 + 4l, b_1 + 6l, \ldots, b_2 - 6l, b_2 - 4l, b_2 - 2l, b_2\}$$

We continue this process in general, creating P_j^i to define the region that we "break up" into segments of size $2^{i-1} l$. We continue until P_j^m, so that every rectangle $\bar{I} \in \mathcal{A}(\mathcal{P})$ such that $\bar{I} \cap G \neq \emptyset$ has the property that $|\bar{I}| \leq \min\{C_1 d(\bar{x}^*, \bar{I}), l\}$. So precisely:

$$a_i = \text{ largest integer } w \text{ divisible by } 2^i \text{ such that } \frac{w}{R} < x_j^* - \frac{2^i}{C_1} l$$
$$b_i = \text{ smallest integer } w \text{ divisible by } 2^i \text{ such that } \frac{w}{R} > x_j^* + \frac{2^i}{C_1} l$$

then

$$P_j^i = \{a_i, a_i + 2^{i-1}l, a_i + 2(2^{i-1})l, a_i + 3(2^{i-1})l, \ldots, a_{i-1} - 2(2^{i-1})l, a_{i-1} - 2^{i-1}l, a_{i-1}\} \cup$$
$$\{b_{i-1}, b_{i-1} + 2^{i-1}l, b_{i-1} + 2(2^{i-1})l, b_{i-1} + 3(2^{i-1})l, \ldots, b_i - 2(2^{i-1})l, b_i - 2^{i-1}l, b_i\}$$

we do this for all $i = 1, 2, \ldots, m$ where again $m = \lceil \log_2 CR\Delta \rceil$. Note that $m = O(\log R)$.

By our construction, we have taken care to make sure that our pattern is legal within G. The only region where our pattern might not be legal is around a_m and b_m. To ensure the overall legality, we must ensure that all the "ancestors" of these points are in our interval partition P_j. So we let:

$$Q_j = \{ \text{ the ancestors of } a_m, b_m \text{ in the legality tree } \} \tag{A-16}$$

Since the depth of the tree is $\log_2(R)$, clearly $|Q_j| \leq 2\log_2(R)$.

Finally we let

$$P_j = \left(\bigcup_{i=0}^{m} P_j^i \right) \cup Q_j \tag{A-17}$$

and let

$$\mathcal{P} = (P_1, P_2, \ldots, P_n) \tag{A-18}$$

It is straightforward from our construction that $\mathcal{P}$ fulfills the criteria of Lemma .0.1 and is therefore sufficient. Now we need only verify how $|\mathcal{P}|$ grows with R.

Lemma .0.8. *Under our construction,* $|\mathcal{P}| = O(\log R)$.

Proof. We know that $|\mathcal{P}| \leq \sum_{i=0}^{m} |P_j^i|$. Furthermore, $|P_j^0|$ is a constant independent of R, call it C_0. It is also not difficult to see that:

$$|P_j^1| \leq \frac{\frac{4}{C_1}l}{l} + 4 = \frac{4}{C_1} + 4$$

which is also independent of R. Moreover, for $i = 2, 3, \ldots, m$.

$$\begin{aligned} |P_j^i| &= \frac{\frac{2^i}{C_1}l}{2^{i-1}l} + 4 \\ &= \frac{2}{C_1} + 4 \end{aligned}$$

So we get:

$$\begin{aligned} |\mathcal{P}| &\leq C_0 + \frac{4}{C_1} + 4 + (m-1)(\frac{2}{C_1} + 4) + 2\log_2(R) \\ &\leq C_2 + mC_3 + 2\log_2(R) \text{ for some constants } C_2, C_3 \text{ independent of } R \\ &\leq O(\log R) \text{ since } m = O(\log R) \end{aligned}$$

□

So we have a sequence of sufficient resolution patterns $\mathcal{P}(R)$ such that $|\mathcal{P}(R)| = O(\log R)$. By Proposition .0.2 we can conclude that our CFDP run time is $O((\log R)^{k+2})$. □

Theorem for C^2 functions

Theorem .0.9. *Suppose the f_j are C^2 and that f has a unique maximum $\bar{x}^*$. Assume further that the Taylor expansion of f at $\bar{x}^*$ has a negative-definite matrix coefficient in the quadratic term and that f respects a graph of tree-width k. Finally, assume there exists some R_1 such that $R > R_1$ implies $\bar{I}^*$ is unique. Then coarse-to-fine dynamic programming will find the rectangle $\bar{I}^*(R)$ in time $O(R^{\frac{k+2}{2}})$.*

Proof. The proof for this theorem is nearly identical to the previous proof. Once again, the overestimate factor can be bounded since f is continuous on a compact set. Again, we define a set G which is a neighborhood of $\bar{x}^*$ and use it to create a sufficient resolution pattern $\mathcal{P}$ for any big enough resolution R. However, in this case, the relation between the distance of point $\bar{x} \in G$ from $\bar{x}^*$ and the difference $f(\bar{x}^*) - f(\bar{x})$ will be quadratic, rather than linear. Thus when we use the properties of G to form $|\mathcal{P}|$, we will find that $|\mathcal{P}| = O(\sqrt{R})$. As a result, our final CFDP complexity will be $O((\sqrt{R})^{k+2})$ rather than $O((\log R)^{k+2})$.

Since f is C^2, and $[0,1]^n$ is compact, we can find a constant $K_1 > 0$ such that:

$$K_1 > \max\{|\frac{\partial f_i}{\partial x_j}(\bar{x})|, |\frac{\partial f}{\partial x_j}(\bar{x})|\} \tag{A-19}$$

where the max is taken over all $i \in 1, 2, \ldots s$, all $j \in 1, 2, \ldots, n$ and all $\bar{x} \in [0,1]^n$ where the partial derivatives exist.

Lemma .0.10. *Recall that we defined $OE(\bar{I}) = f(\bar{I}) - \sup_{\bar{x} \in \bar{I}} f(\bar{x})$. Under the assumptions of our theorem we have:*

$$OE(\bar{I}) \leq s(k+1)K_1|\bar{I}| \tag{A-20}$$

Proof. The proof is identical to that of Lemma .0.5 in the previous proof. □

We will now create the set G for this problem. Specifically, we will find a $\Delta > 0$, $\epsilon' > 0$ and $K_2 > 0$ such that if we define G by:

$$\bar{x} \in G \iff x_j \in [x_j^* - \Delta, x_j^* + \Delta] \tag{A-21}$$

then we have the following properties

1. $\bar{x} \notin G \Rightarrow f(\bar{x}) < f(\bar{x}^*) - \epsilon'$
2. $\bar{x} \in G \Rightarrow f(\bar{x}) < f(\bar{x}^*) - K_2 d(\bar{x}, \bar{x}^*)^2$

By expanding f in a Taylor Series, and using the fact that $\bar{x}^*$ is a maximum, we get:

$$f(\bar{x}^*) - f(\bar{x}^* + \bar{h}) = -q(\bar{h}) - R_2(\bar{h}) \tag{A-22}$$

where $q(\bar{h})$ is a negative-definite quadratic and $R_2(\bar{h}) = o(|\bar{h}|^2)$. That is,

$$\lim_{|h|\to 0} \frac{R_2(\bar{h})}{|\bar{h}|^2} = 0 \tag{A-23}$$

Note that we can rewrite $q(\bar{h})$ as

$$q(\bar{h}) = |\bar{h}|^2 q\left(\frac{\bar{h}}{|\bar{h}|}\right) \tag{A-24}$$

Now let:

$$M = \min_{|h|=1}\{-q\left(\frac{\bar{h}}{|\bar{h}|}\right)\}$$

and let $K_2 = \frac{M}{2}$. Since $R_2(\bar{h}) = o(|\bar{h}|^2)$, we can find an h_0 such that:

$$|\bar{h}| < h_0 \Rightarrow |R_2(\bar{h})| < K_2|\bar{h}|^2$$

So for $|h| < h_0$ we have:

$$\begin{aligned} f(\bar{x}^*) - f(\bar{x}) &= -q(h) - R_2(\bar{h}) \\ &\geq M|\bar{h}|^2 - K_2|\bar{h}|^2 \\ &\geq K_2|\bar{h}|^2 \end{aligned}$$

So if $G \subseteq B_{h_0}(\bar{x}^*)$ then we are assured that the second of our two required properties is fulfilled. Now we need only ensure the first property. We can do this by choosing $\epsilon' > 0$ such that:

$$\{\bar{x} : f(\bar{x}) \geq f(\bar{x}^*) - \epsilon'\} \subseteq B_{h_0}(\bar{x}^*)$$

Since f is continuous and $\bar{x}^*$ is a unique maximum we are guaranteed to find such an ϵ'. Now we just choose $\Delta < \frac{h_0}{n}$. By our construction, our set G satisfies both of the required properties.

Lemma .0.11. *If $\bar{I} \cap G = \emptyset$ and $|I| \leq \frac{\epsilon'}{skK_1}$ then $f(\bar{x}^*) > f(\bar{I})$.*

Proof. The proof is identical to Lemma 3.6.1 in the linear case. □

Lemma .0.12. *If $\bar{I} \cap G \neq \emptyset$ and $|I| \leq \min\{\frac{K_2}{s(k+1)K_1} d(\bar{x}^*, \bar{I})^2, \frac{\epsilon'}{s(k+1)K_1}\}$ then $f(\bar{x}^*) > f(\bar{I})$.*

Proof. Since $\bar{I} \cap G \neq \emptyset$, we know that:

$$\bar{x} \in \bar{I} \Rightarrow f(\bar{x}) < f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I})^2, \epsilon'\}$$

Letting $K_3 = s(k+1)K_1$, we have:

$$\begin{aligned} f(\bar{I}) &= \max_{\bar{x}\in I} f(\bar{x}) + OE(\bar{I}) \\ &\leq f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I})^2, \epsilon'\} + OE(\bar{I}) \\ &\leq f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I})^2, \epsilon'\} \ + s(k+1)K_1|\bar{I}| \\ &\leq f(\bar{x}^*) - \min\{K_2 d(\bar{x}^*, \bar{I})^2, \epsilon'\} + K_3 \min\{\frac{K_2}{K_3} d(\bar{x}^*, \bar{I})^2, \frac{\epsilon'}{K_3}\} \\ &\leq f(\bar{x}^*) \end{aligned}$$

□

In a similar way to the previous proof, we construct our resolution pattern $\mathcal{P}$. Let $C_1 = \frac{K_2}{s(k+1)K_1}$.

$$P_j^0 = \{\frac{1}{2^a}, \frac{2}{2^a}, \frac{3}{2^a}, \frac{4}{2^a}, \dots, \frac{2^a}{2^a}\} \tag{A-25}$$

$$\begin{aligned} a_1 = &\text{ largest integer } w \text{ divisible by 2 such that } \frac{w}{R} < x_j^* - \sqrt{\frac{2}{C_1}}\sqrt{l} \\ b_1 = &\text{ smallest integer } w \text{ divisible by 2 such that } \frac{w}{R} > x_j^* + \sqrt{\frac{2}{C_1}}\sqrt{l} \end{aligned}$$

then

$$P_j^1 = \{a_1, a_1 + l, a_1 + 2l, a_1 + 3l, \dots, b_1 - 3l, b_1 - 2l, b_1 - l, b_1\}$$

$$\begin{aligned} a_2 = &\text{ largest integer } w \text{ divisible by 4 such that } \frac{w}{R} < x_j^* - \sqrt{\frac{4}{C_1}}\sqrt{l} \\ b_2 = &\text{ smallest integer } w \text{ divisible by 4 such that } \frac{w}{R} > x_j^* + \sqrt{\frac{4}{C_1}}\sqrt{l} \end{aligned}$$

then

$$\begin{aligned} P_j^2 = \{a_2, a_2 + 2l, a_2 + 4l, a_2 + 6l, \dots, a_1 - 6l, a_1 - 4l, a_1 - 2l, a_1\} \cup \\ \{b_1, b_1 + 2l, b_1 + 4l, b_1 + 6l, \dots, b_2 - 6l, b_2 - 4l, b_2 - 2l, b_2\} \end{aligned}$$

$$\begin{aligned} a_i = &\text{ largest integer } w \text{ divisible by } 2^i \text{ such that } \frac{w}{R} < x_j^* - \sqrt{\frac{2^i}{C_1}}\sqrt{l} \\ b_i = &\text{ smallest integer } w \text{ divisible by } 2^i \text{ such that } \frac{w}{R} > x_j^* + \sqrt{\frac{2^i}{C_1}}\sqrt{l} \end{aligned}$$

then

$$\begin{aligned} P_j^i = \{a_i, a_i + 2^{i-1}l, a_i + 2(2^{i-1})l, a_i + 3(2^{i-1})l, \dots, a_{i-1} - 2(2^{i-1})l, a_{i-1} - 2^{i-1}l, a_{i-1}\} \cup \\ \{b_{i-1}, b_{i-1} + 2^{i-1}l, b_{i-1} + 2(2^{i-1})l, b_{i-1} + 3(2^{i-1})l, \dots, b_i - 2(2^{i-1})l, b_i - 2^{i-1}l, b_i\} \end{aligned}$$

we do this for all $i = 1, 2, \dots, m$ where again $m = \lceil \log_2 CR\Delta^2 \rceil$. Note that $m = O(\log R)$.

Finally, we let:

$$Q_j = \{ \text{ the ancestors of } a_m, b_m \text{ in the legality tree } \} \tag{A-26}$$

Since the depth of the tree is $\log_2(R)$, clearly $|Q_j| \leq 2\log_2(R)$.

Again we define:

$$P_j = \left(\bigcup_{i=0}^{m} P_j^i \right) \cup Q_j \tag{A-27}$$

and let

$$\mathcal{P} = (P_1, P_2, \ldots, P_n) \tag{A-28}$$

From our construction, we have forced any rectangle $\bar{I}$ to either have minimum length l or to have the property that $f(\bar{I}) < f(\bar{x}^*)$. So by Lemma .0.1, we can conclude that $\mathcal{P}$ is sufficient.

Lemma .0.13. *Under our construction,* $|\mathcal{P}| = O(\sqrt{R})$.

Proof. We know that $|\mathcal{P}| \leq \sum_{i=0}^{m} |P_j^i|$. Furthermore, $|P_j^0|$ is a constant independent of R, call it C_0. It is also not difficult to see that:

$$|P_j^1| \leq \frac{2\sqrt{\frac{2}{C_1}}\sqrt{l}}{l} + 4 = C_2\sqrt{R} + 4$$

which is also independent of R. Moreover, for $i = 2, 3, \ldots, m$.

$$\begin{aligned} |P_j^i| &= \frac{\frac{2}{C_1}(\sqrt{2^i} - \sqrt{2^{i-1}})\sqrt{l}}{2^{i-1}l} + 4 \\ &= \frac{\frac{2}{C_1}(\sqrt{2} - 1)(\sqrt{2^{i-1}})\sqrt{R}}{2^{i-1}} + 4 \\ &= \frac{C_3\sqrt{R}}{\sqrt{2^{i-1}}} + 4 \end{aligned}$$

Recall that $m = \lceil \log_2 CR\Delta^2 \rceil$ so we can say $m \leq \log_2 2CR\Delta^2$

So we get:

$$\begin{aligned} |\mathcal{P}| &\leq C_0 + C_2\sqrt{R} + 4 + \sum_{i=2}^{m} \left(\frac{C_3\sqrt{R}}{\sqrt{2^{i-1}}} + 4\right) + 2\log_2(R) \\ &\leq C_0 + C_2\sqrt{R} + 4m + C_3(\sqrt{R})\left(\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{4}} + \frac{1}{\sqrt{8}} + \ldots + \frac{1}{\sqrt{2^m}}\right) + 2\log_2(R) \\ &\leq C_0 + C_2\sqrt{R} + 4m + C_3(\sqrt{R})(1 + \sqrt{2}) + 2\log_2(R) \\ &\leq C_0 + C_2\sqrt{R} + 4(\log(2CR\Delta^2) + C_3(1 + \sqrt{2})\sqrt{R} + 2\log_2(R) \\ &= O(\sqrt{R}) \end{aligned}$$

□

So we have constructed a sequence of legal, sufficient resolution patterns $\mathcal{P}(R)$ for $R \to \infty$, with the property that $|\mathcal{P}(R)| = O(\sqrt{R})$. By Proposition .0.2 the CFDP complexity for these problems is $O(R^{\frac{k+2}{2}})$. □

Bibliography

[1] E. Amir. Efficient approximation for triangulation of minimum tree-width. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2001.

[2] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey. *BIT*, 25:2–23, 1985.

[3] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[4] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 81–89. Morgan Kaufmann, 1996.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[6] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[7] C. Cannings, T. E.A., and H. Skolnick. Probability functions on complex pedigrees. *Advances in Applied Probability*, 10:26–61, 1978.

[8] P. A. Cohen P., Chaudhri V. and S. R. Does prior knowledge facilitate the development of knowledge-based systems. In *AAAI Proceedings '99*, pages 221–226, 1999.

[9] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In M. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1999.

[10] B. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

[11] R. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, Massachusetts Institute of Technology, 1963.

[12] S. Geman and K. Kochanek. Dynamic programming and the representation of soft-decodable codes. *IEEE Transactions on Information Theory*, 47(2):549–568, 2001.

[13] J. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. Technical report, University of California, Berkeley, 1968.

[14] F. V. Jensen and F. Jensen. Optimal junction trees. In R. Mantaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 360–366. Morgan Kaufmann, 1994.

[15] U. Kjaerulff. *Aspects of Efficiency Improvement in Bayesian Networks.* PhD thesis, Aalborg University, Department of Mathematics and Computer Science, 1993.

[16] K. Kochanek. *Dynamic Programming Algorithms for Maximum Likelihood Decoding.* PhD thesis, Division of Applied Mathematics, Brown University, 1998.

[17] S. Lauritzen and D. Speigelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological)*, 50(2):157–224, 1988.

[18] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Comm. ACM*, 38(11):33–38, 1995.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[20] M. B. Pradhan M., Provan G. and H. M. Knowledge engineering for large belief networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 484–490, 1994.

[21] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM Journal of Discrete Mathematics*, 10(1):146–157, 1997.

[22] C. Raphael. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1379–1390, 2001.

[23] C. Raphael and S. Geman. A grammatical approach to mine detection. In *Detection and Remediation Technologies for Mines and Minelike Targets II, Proceedings of SPIE*, pages 316–332, 1997.

[24] B. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[25] N. Robertson and P. D. Seymour. Graph minors ii: algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.

[26] N. Robertson and P. D. Seymour. Graph minors xiii: the disjoint paths problem. *Journal of Comb. Theory, Series B*, 63:65–110, 1995.

[27] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[28] D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7:317–322, 1974.

[29] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13(3):566–579, 1984.