UNIVERSITY OF CALIFORNIA,
IRVINE

Dependency Diagrams and Graph-Constrained Correlation Dynamics: New Systems for Probabilistic Graphical Modeling

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Gary Todd Johnson

Dissertation Committee:
Professor Eric Mjolsness, Chair
Professor Pierre Baldi
Professor Peter Kaiser
Professor Alex Ihler

2012

The dissertation of Gary Todd Johnson
is approved and is acceptable in quality and form for
publication on microfilm and in digital formats:

_____

_____

_____

Committee Chair

University of California, Irvine
2012

DEDICATION

To my parents, my grandparents, my wife, my son.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my wife, Dr. Ruth Barrett. She inspires me every day, as a better person, parent, runner, and scientist than I will ever be. She is also an amazing source of support and happiness, and her presence has made these last six years fly by.

I would like to thank my parents, and my grandparents. Susan, Gary, James, and Mary. None of them had a chance to go to college, and this was all they ever wanted for me. I may have given them a little more than they bargained for with graduate school, and I only hope I have made them proud.

I cannot help but thank my advisor, Professor Eric Mjolsness. Eric has been everything a student seeks in an advisor, and more. His patience with and confidence in my work and abilities provided the fuel needed to work through many tough corners, and his guidance and scientific intuition were invaluable.

I would like to thank the department of computer science for the Chair's Award, and the Biomedical Informatics Training program for funding much of my graduate education.

I would like to thank Professors Peter Kaiser, Terry Sejnowski, and Tom Bartol for fruitful collaborations, as well as Dr. Shirley Pepke and Professor Mary Kennedy for data and insights.

The list of professors and peers whose guidance, friendship, and collaboration have seen me through the last five years is almost too long to print. In particular, Professors Alex Ihler, Pierre Baldi, Max Welling, Padhraic Smyth, and Xiaohui Xie have been wonderful teachers and inspirations.

Fellow students Kenny Daily, Lars Otten, Ramzi Nasr, Matt Kayala, Drew Frank, Keith Booher, Chris DuBois and a bunch more I don't have space to name were tremendous collaborators and teammates. In particular, my officemate David Orendorff has been the source of innumerable fruitful conversations. Having such incredible peers can be daunting, but getting to know these gentlemen has been a wonderful privilege.

This work was supported in part by the NIH Biomedical Informatics Training grant (LM-07443-01).

CURRICULUM VITAE

Gary Todd Johnson

EDUCATION

Doctor of Philosophy in Computer Science **2012**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science **2008**
University of California, Irvine *Irvine, California*

Bachelor of Arts in Computer Science **2005**
Bard College *Annandale-on-Hudson, New York*

RESEARCH EXPERIENCE

Graduate Research Assistant **2006–2012**
University of California, Irvine *Irvine, California*

SELECTED HONORS AND AWARDS

Biomedical Informatics Training grant **2007–2010**

ABSTRACT OF THE DISSERTATION

Dependency Diagrams and Graph-Constrained Correlation Dynamics:
New Systems for Probabilistic Graphical Modeling

By

Gary Todd Johnson

Doctor of Philosophy in Computer Science

University of California, Irvine, 2012

Professor Eric Mjolsness, Chair

Dependency Diagrams (DDs) and Graph-Constrained Correlation Dynamics (GCCD) expand the universe of probabilistic graphical models to allow their application in a number of new domains. Dependency Diagrams are a new graphical modeling language which permit graphical representation of model structure and logic beyond the scope of existing formalisms. This allows the generation of graphical models that represent new classes of objects, such as algorithms for sampling and inference, and for the transformation between graphical representations of probabilistic domain models and graphical representations of algorithms acting on those models. This thesis provides semantics for the probability distributions defined by DDs, and describes a process for generating diagrams that specify sampling procedures. Graph-Constrained Correlation Dynamics adds to existing graphical modeling formalisms the ability to represent distributions that evolve continuously in time. In particular, with GCCD a Markov random field is be used to specify continuously evolving parametrized probability distributions in the domain of reaction networks. This capability opens the possibility of representing continuously in time distributions over subsets of complex stochastic systems, permitting model reduction by allowing larger stochastic models the freedom to avoid explicitly representing reactions over these subsets without

sacrificing statistical properties of their simulations. This thesis derives algorithms for optimizing the approximation of a target distribution by a GCCD model, and demonstrates the success of these algorithms. Together, DDs and GCCD represent significant advancements in the available vocabularies of probabilistic graphical modeling, and open the door for exciting new scientific applications.

Chapter 1

Introduction

Probabilistic graphical models provide the foundation for crucial algorithms and techniques in virtually every field open to computational analysis, from the hidden Markov model used widely in genomics and speech processing to the Markov random fields used to formulate graph-cut-based image segmentation techniques. This utilitiy and popularity can be traced to a number of appealing properties of graphical models. Among these are the fact that graphical modeling frameworks provide powerful notations for mathematics, facilitating both the derivation of algorithms and the proof of optimality formulas. From a computational perspective, these same characteristics of graphical models allow them to serve as the fundamental data structures of much of statistical computation. Graphical models also aid in the organization and transmission of ideas, facilitating both the process of creating models and the understanding thereof.

As such, a tremendous amount of research is devoted to the study and application of known graphical model systems, such as Bayesian networks [62], Markov random fields [7], and factor graphs [41]. Techniques exist and are continually refined for

construction, sampling, inference, and optimization of graphical models, and any number of specific models are specified every year. This thesis develops new tools for probabilistic modeling, including a new class of graphical models and a framework to allow existing models to be applied in new domains. Each builds on existing graphical modeling frameworks to provide novel capabilities.

The *Dependency Diagram* system developed in Chapters 2 through 4 expands the existing language of graphical models, so that they approach the expressiveness of first-order probabilistic modeling languages without resorting to extra-graphical notation. To the extent that existing graphical models admit the representation of repeated structure, they do so via "plates" and other annotations outside of the language of graphs. Dependency diagrams move these ideas into the graphical models, and adds the ability to specify algebraic or logical relationships between them. This flexibility allows sampling and inference algorithms for Dependency Diagrams to be expressed as Dependency Diagrams, and in particular allows for algorithmic and programmatic translation of a model diagram into a sampling algorithm diagram. This, coupled with the finite difference and source code generation engines developed along with Dependency Diagrams, produces a powerful tool for general-purpose modeling.

This thesis develops the probabilistic semantics of a variety of node and link types which allow Dependency Diagrams to represent, within a graph, concepts such as repeated structure and conditional evaluation of subgraphs. These semantics are realized in a software package for representing and interpreting Dependency Diagrams. This package also implement techniques for transforming a Dependency Diagram into a directed model of the sampling process, and for generating reusable code for sampling from directed Dependency Diagrams. Dependency Diagrams constitute a novel graphical modeling formalism, with unique representational capabilities, and the implemented software provides valuable probabilistic modeling capabilities within the

Mathematica programming environment.

The *Graph-Constrained Correlation Dynamics* formalism developed in Chapters 5 through 7 builds on Dependency Diagrams to allow probabilistic graphical models to be applied to an entirely new set of problems. In particular, GCCD allows Markov random fields to be used to design goal-directed approximations of probability distributions. In these approximate models, MRF model parameters evolve *continuously and deterministically in time* according to fixed dynamics, as is appropriate in many fundamental applications in science in engineering. This ability to model continuous change in time stands in contrast to existing systems for time-dependent graphical models, such as Hidden Markov Models [4] and Dynamic Bayesian Networks [61], which discretize time. GCCD therefore allows the evaluation across time of distributions over state spaces specified by, for example, chemical reaction networks (a domain which will be explored in detail), where most existing methods must track either fixed functions of the state or perform repeated stochastic simulations from individual starting states in order to approximate distributions. The nearest comparable model is Continuous-Time Bayesian Networks (CTBNs), which provide a related ability to model continuously-evolving probability distributions [60]. However, because GCCD creates a deterministic ordinary differential equation model of the change in the probability distribution over time, unlike CTBNs, the time to evaluate a future time probability does not depend on the time difference or on the rates at which reactions occur.

Because Graph-Constrained Correlation Dynamics leverages the Markov random field formalism for specifying parametrized probability distributions, it permits goal-directed model reduction. Modelers have the flexibilty to specify a set of basis functions of the state of interest, and GCCD will optimize the time-evolution of a probability distribution in terms of these functions. Modelers also have the flexibility to intro-

duce hidden variables, clique potentials, and hence graph structure, likely to be useful based on domain knowledge. This will allow stochastic models which require samples for some subset of species only at particular times or intervals to avoid generating trajectories for those species, and instead quickly solve a system of equations and generate samples from the appropriate distribution on demand.

We also demonstrate a software package which implements GCCD for Markov random field models built using Dependency Diagrams, and demonstrate its ability to fit probabilistic models of chemical reaction networks evolving continuously in time. One application is to a model of calcium binding and transport in the dendritic spine, the portion of a neuron cell which receives a signal from a single axon across a synapse. GCCD represents a new direction in the learning of and model reduction for complex stochastic dynamical systems, which leverages Markov random fields in a novel way.

Together, these systems expand the toolkit available to scientists for modeling natural phenomena. These projects contribute significantly to the graphical modeling and reaction network simulation literatures by providing unique, powerful computational capabilities.

Chapter 2

Dependency Diagrams: Background and Motivation

2.1 Introduction

Diagrammatic notations such as Bayesian networks and factor graphs have been widely adopted to express probabilistic models for machine learning and statistical inference. They have the advantage of being a natural form for human communication and understanding, and when formalized in terms of labeled graphs they also provide representations on which algorithms for analysis, sampling, and inference can operate. Probabilistic models that can be represented by classes of diagrams such as Bayesian networks (BNs) and Markov random fields (MRFs) have received a great deal of attention, algorithm development, and application.

Mathematical extensions of current diagrammatic notations could do even more for machine learning and artificial intelligence. For example, models with a variable structure, such as varying numbers of instances of one variable depending on the val-

ues of other variables, have been awkward to specify entirely within existing graphical models, typically because some key structure-update equations could not be deduced from the diagrams. New classes of models may be opened for exploration if these limitations can be removed. Another substantial increase in utility may be available if not only the *models*, but also the *algorithms* that sample or infer from the models, can be represented and manipulated diagrammatically. In particular, the ability to compute algorithm diagrams from model diagrams would allow automatic generation of inference engines as well as novel graphical analysis and manipulation of representations of algorithms.

The *Dependency Diagrams* proposed below achieve many of these goals. Dependency Diagrams [51] extend chain graphs, which can contain both directed (conditional) and undirected (correlation-inducing) links between random variables [42], by a) adopting the extended factor graph representation [21] with suitably labeled factor nodes and link types, which includes BN and MRF factors; b) replacing the "plates" notation for subscripted and replicated variables with more explicit "index" node and link types; c) singling out some inputs to factor nodes as "gating links" which regulate the presence or absence of the factor in the model; and d) adding constraint links to constrain the values of index nodes.

The resulting diagrams can be translated precisely into algebraic expressions for the probabilistic models they represent. They can also be transformed automatically into other Dependency Diagrams (composed of exactly the same node and link types) that represent Markov Chain Monte Carlo (MCMC) algorithms for sampling and inference on the models. These algorithm diagrams can then be used to automatically generate working code, as demonstrated here by autogenerating sampling or inference algorithms from Dependency Diagrams for several simple models. Problems considered include the canonical ALARM network, clustering, reaction rate inference, and

a variable-structure system for sparse graph priors of current utility in computational biology. Finally, Dependency Diagrams both as a specification language and to drive sample-based approximation in the Graph-Constrained Correlation Dynamics framework, which models the continuous evolution in time of a probability distribution. Our implementation of this framework leverages Dependency Diagrams to sample modeled distributions without the necessity of dedicated research building inference engines for each model.

2.2 Background and Related Work

In this section we discuss related graphical probabilistic modeling work (Section 2.2), before moving on to define some notations used in this paper (Section 3.1), define the semantics of factor graph notation in particular (Section 3.2), define the semantics of a series of novel node and link types for Dependency Diagrams (Sections 3.3, 3.4, 3.5), and specify a general transformation from Dependency Diagrams that define probabilistic models to Dependency Diagrams that define stochastic algorithms for sampling such models (Section 3.6).

The practice of describing interactions between random variables with graph-like diagrams has arisen independently several times in different scientific disciplines. In the study of statistical mechanics, undirected interactions between random variables have traditionally been expressed using diagrams representing the energy functions of Boltzmann distributions. *Markov random fields* [7], which arose from the study of spatial Markov processes [19], represent a general framework for specifying probability distributions in terms of undirected interactions between variables. Isham [33] and Geman and Geman [23] surveyed and described the conditional independence properties of such distributions in terms of graph decompositions. The di-

rected, conditional-probability approach to such interactions and their conditional independence properties was formulated in terms of *Bayesian networks* by Pearl [62]. Lauritzen [42] defined *chain graphs*, which contain both directed (BN-like) and undirected (MRF-like) links, indicated by undirected lines and directed arrows. Bipartite *Tanner graphs* were introduced in the study of coding and information theory, where the nodes represented digits of code and subcodes [73]. Later, Tanner graphs were generalized to represent a wider family of systems – including probability distributions – when Kschischang, Frey and Loeliger introduced *factor graphs* (FGs) [41]. In a factor graph, as in a Tanner graph, factors (subcodes) and random variables (digits) are both represented explicitly, using two distinct types of nodes. Later, Frey extended factor graphs to simultaneously incorporate both directed and undirected interactions [21]. Factor graphs thus allow more detailed expression of the interactions and independencies which could be described using chain graphs, but retain the structural determination of conditional independence via graph separation criteria.

Buntine's introduction of *plates* [11] allowed researchers to decouple the size of a diagram from the number of variables in the probabilistic model. A plate is typically drawn as a rectangle which encloses a set of variable nodes, interaction links, and link ends, and signifies their replication by a shared index or subscript, which in turn runs over a fixed set of index values. Buntine's notation also includes the use of double circles around a node to indicate a "clamped" or "frozen" random variable, which is to say one that has been converted into a fixed numerical parameter or observation. Occasionally, other common notations find their way into diagrams of graphical models, including electrical engineering notations for a multi-way switch (a lever with two or more contacts), or gating arrows pointing to the middle of a link, and so on. These plate, switch, and gate notations do not take the form of labeled graphs, which means that they may not avail of graph-based reasoning and mathematics.

There have also been a number of other systems for graphical models which, though not directly a part of the genealogy of Dependency Diagrams, are related. For instance, Heckerman and colleagues [30] introduced the *dependency network* graphical model formalism as an alternative to Bayesian networks. Dependency networks, which are described by a (potentially cyclic) graph and a conditional probability for each variable given all of its neighbors, were designed to be easily learned from data, and to be more easily interpreted by non-specialists who are confused by phenomena such as "explaining away" [62]. Dependency networks are unique in that the joint probability described by a particular network structure is defined as the stationary distribution of an ordered Gibbs sampler run on the same network, meaning that the resulting joint distribution depends on the ordering used by the sampler [30]. Dependency Diagrams follow in the line of more traditional graphical models, for which consistency and the potential for exact inference are maintained at the cost of more expensive learning procedures, but share with dependency networks the aim of flexibly representing both directed relationships and less easily interpreted notions of dependence between variables. The more recent *relational dependency networks* formalism extends dependency networks to the types of first-order models discussed below [58] and perhaps makes a more apt comparison for Dependency Diagrams, but is still designed with a primary focus on low cost of learning in large data sets. Higher-level abstractions of graphical models have been actively developed for inference of unknown sets of objects and relational structures [e.g. 49, 67, 31, 24, 47, 52, and others]. Some of these, such as BUGS [47], BLOG [49] and Markov Logic Networks [67], are fully-featured software systems amounting to special purpose programming languages for probabilistic models. The framework reported here also supports highly structured and variable-structure models, and therefore allows [51] the expression of a class of models comparable to the classes of "first-order probabilistic models" represented by the these other systems. First-order probabilistic models refer to those

capable of representing unknown numbers of objects, or objects whose specific identities are unknown [48]. The Dependency Diagram indexing syntax, as described in Section 3.4, may be extended to permit unknown or variable bounds [51], and as described may be used in conjunction with gating or existence links, described in Sections 3.3 and 3.5, to describe large collections of variables which may or may not be present. This capability, combined with a sparse correspondence matrix, would allow similar capabilities within the existing Dependency Diagram software system.

But the Dependency Diagram software package is not yet at the same level of usability as BLOG or Markov Logic Networks for investigating complex domain models. Instead it is intended to demonstrate the potential of the dependency diagram framework for unambiguous specification of complex probabilistic models purely by diagrams, and consequently for the automatic, flexible and semantics-preserving transformation of such diagrammatic models into algorithms for sampling and inference. Dependency Diagrams directly extend existing graphical formalisms to allow the expression of models with repeated elements, including conditional relationships between indices and and variable structure, and this paves the way for new algorithms and simplifications based on the richer graphical structure available in Dependency Diagram models.

Chapter 3

Dependency Diagrams: Theory and Implementation

3.1 Notation

By analogy to the standard set-builder notation $\{x|P(x)\}$ for defining the members of a set using a predicate P, we will define ordered sets using square brackets,

$$[x(i)|P(x(i),i)\|i \in \mathcal{I}].$$

The ordering of the index set $\mathcal{I}$ (e.g. $\mathcal{I} \subseteq \mathbb{N}$) is imposed on any elements $x(i)$ selected for inclusion by the predicate P, denoting a set together with a total ordering. This notation can be read as "$x(i)$ such that $P(x(i), i)$, ordered by $i \in \mathcal{I}$." Either or both of the "$|P(x(i), i)$" and "$\|i \in \mathcal{I}$" clauses may be omitted if the condition P or the ordering $\mathcal{I}$ is obvious from the context. For example, $[x_i]$, which we write as shorthand for $[x(i)]$, denotes a vector with components x_i, where i runs over some fixed ordered range. The indicator function $\mathbf{1}(P)$ is 1 if predicate P is true and 0

otherwise. Written elsewhere as $[P]$, this is known as "Iverson's convention" [41] and satisfies $\mathbf{1}(\wedge_a P_a) = \prod_a \mathbf{1}(P_a)$.

Our strategy for defining the semantics for Dependency Diagrams is to begin with those node and link types necessary to represent factor graphs. We will then describe further classes of Dependency Diagrams constructed by the addition of new node and link types. For each new piece of graphical syntax will be defined either by a semantic function Ψ from diagrams containing those node and link types to probability density functions or by a reduction $\mathcal{E}$ to a previously defined class of Dependency Diagrams.

Different classes of Dependency Diagrams are parametrized by their link types and function spaces. For example, $\mathcal{D}(f, d; \mathcal{F})$ is the class of factor-graph-encoding Dependency Diagrams defined in Section 3.2 below, with f-links for undirected factors, d-links for directed factors, and all factors in function space $\mathcal{F}$.

3.2 Factor Graph Semantics

Following Frey [21], recall that undirected (MRF-like) and directed (BN-like) probabilistic graphical models may be incorporated into the factor graph framework by introducing nodes labeled by probability density factors. In the notation of parametrized classes of Dependency Diagrams, factor graphs are precisely the class $\mathcal{D}(f, d; \mathcal{F})$, with factors ϕ_κ connected to variables by undirected f-links and directed d-links. Each factor node ϕ_κ is labeled with a member of some function space $\mathcal{F}$ of functions whose values are nonnegative. Examples of common spaces used in this context include conditional probabilities and exponentials of real-valued "energy functions."

In order to express succinctly the probability distribution associated with a diagram

$\mathbb{D} \in \mathcal{D}(f, d; \mathcal{F})$, we introduce a bit more notation. Let $\kappa \in \mathcal{K}_1$ index the factor nodes with undirected f links, and $\kappa \in \mathcal{K}_2$ index the factor nodes with directed d links, and $i \in \mathcal{I}$ index the random variable nodes. That is, a diagram with variables *grade*, *greScore*, and *admit* would be represented in the algebra to follow in terms of variables x_i for $i \in \mathcal{I} = \{1, 2, 3\}$, where x_1 stands in for *grade*, and so forth.

Further, define two 0/1-valued adjacency matrices F and D, where $F_{\kappa i} = 1$ ($D_{i\kappa} = 1$ or $D_{\kappa i} = 1$) if the variable node indexed by i is connected via an f-link (d-link) to the factor node indexed by κ. In general, each Dependency Diagram link type will define an adjacency matrix representing the links of that type in a given graph. The semantics function Ψ maps graphs with nodes and edges labeled with these symbols to probability density functions and can be written as follows:

$$\Psi(\mathbb{D}) = \Pr(\mathbf{x}) = \frac{1}{Z} \prod_{\kappa \in \mathcal{K}_1} \phi_\kappa([x_i | F_{\kappa i} = 1]) \prod_{\kappa \in \mathcal{K}_2} \phi_\kappa([x_i | D_{\kappa i} = 1] | [x_j | D_{j\kappa} = 1]). \quad (3.1)$$

An f-link between the ith variable node and the κth factor node, or $F_{i\kappa} = 1$, represents participation in a factor ϕ_κ of a Boltzmann distribution or Markov random field. Although the graph F is itself directed, its square $U = FF^T$ symmetrically connects random variables that are related by one or more potential functions. By contrast the directed d-links in D form a directed acyclic graph (DAG) and represent a generalized form of conditional probability density with factors ϕ_κ in which each probability factor participates in the normalization relationship for directed factor graphs [12, 42, 21], and specializes to a conditional distribution if a variable x_i is connected by d-link to a factor ϕ_κ, has no other parents connected by d-links, and has no neighbors connected by f-links. The list of conditioned variables $[x_j | D_{\kappa j} = 1]$ in a factor may be empty. In addition, some of the variable nodes x_i may actually be labeled as fixed parameters. These may be interpreted as conditionalized random variables.

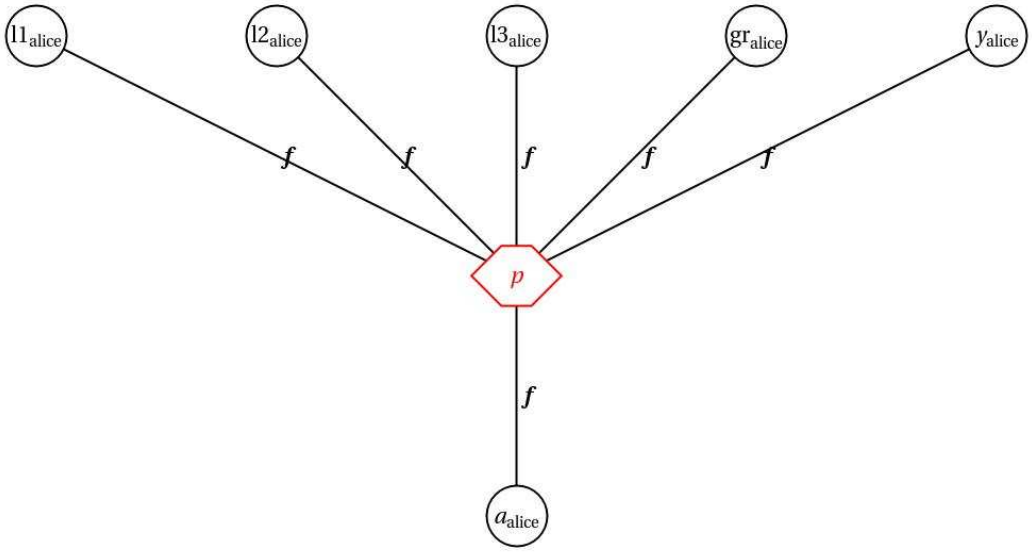


Figure 3.1: Dependency Diagram for graduate admissions.

Section 4 describes several experiments using Dependency Diagram models. However, it will be useful to see simpler illustrative examples as the various link and node types are defined. Therefore, beginning with Figure 3.1, we will build up along the way a Dependency Diagram describing the probability that a student is admitted to graduate school. The first diagram uses only the node and link types defined by factor graphs, and has the following interpretation. The binary variable a represents the reviewer's decision to admit or not, and is influenced by a single factor. This factor takes into account three letters of recommendation, $l1$, $l2$, $l3$ and an undergraduate grade point average gr. Since this GPA may not be relevant if the student is returning to school after some time in the workforce, the model also includes a binary variable y, which indicates whether or not the student is a recent graduate.

3.3 Gating Links

One of the most important ways in which Dependency Diagrams improve upon existing probabilistic graphical model frameworks is in the ability to more easily represent variable-structure systems. The *gating link* is of particular interest for such models. A gating link may be drawn between a variable node x_i and a factor node ϕ_κ, and has the effect that, when $x_i \leq 0$, ϕ_κ does not participate in the product of factors.

In order to define the gating link, the semantic function Ψ is extended so that it assigns to each diagram $\mathbb{D} \in \mathcal{D}(f, d, \gamma; \mathcal{F})$ the following probability density function:

$$\begin{aligned} \Pr(\mathbf{x}) = \quad & \frac{1}{Z} \prod_{\kappa \in \mathcal{K}_1} \phi_\kappa(\{x_i | F_{i\kappa} = 1\})^{(\prod_{k|\Gamma(\kappa,k)=1} \mathbf{1}(x_k > 0))} \\ & \times \prod_{\kappa \in \mathcal{K}_2} \phi_\kappa(\{x_i | D_{i\kappa} = 1\} | \{x_j | D_{\kappa j} = 1\})^{(\prod_{k|\Gamma(\kappa,k)=1} \mathbf{1}(x_k > 0))} \end{aligned} \tag{3.2}$$

Here, the adjacency matrix $\Gamma(\kappa, k) = \Gamma_{k,\kappa}$ specifies the gating links, so that $\Gamma_{k\kappa}$ means variable x_k gates factor ϕ_κ. The Heaviside step function $\mathbf{1}(x_k > 0)$ is applied to each such integer- or real-valued random variable x_k, and is placed in the exponent to the relevant ϕ_κ. Only if all of its gating constraints (if any) are met is a probability factor ϕ_κ multiplied into the joint probability distribution $\Pr(\mathbf{x})$. Equation 3.2 defines the semantics for the DD class $\mathcal{D}(f, d, \gamma; \mathcal{F})$.

Returning to the example of graduate school admissions, one possible model is that undergraduate GPA is only meaningful if the student is a recent graduate. This can be expressed in a Dependency Diagram by first separating the factor p into two pieces (one of which is concerned with the letters and another with the GPA) and by then allowing the 'recent graduate' variable y to gate the GPA factor. Figure 3.2 shows

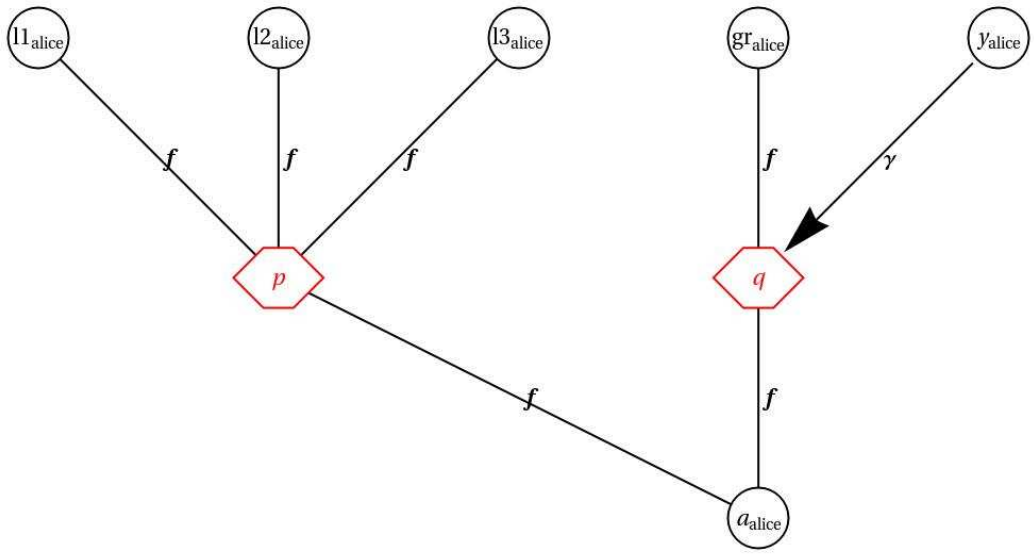


Figure 3.2: Dependency Diagram for graduate admissions, with gating.

the resulting diagram. Obviously, this model is not as general as the original factor graph model. Indeed, this is precisely the point: this diagram is more clear than that in Figure 3.1, because it expresses the nature of the relationship between y and a. The set of distributions corresponding to the model in Figure 3.2 is a subset of those described by Figure 3.1, and we expect tools for analysis and inference to exploit this specificity.

Note that both d- and f-type interactions can be gated. Since $\phi_\kappa^0 = 1$, in the absence of all gating links each product $\prod_k \mathbf{1}(\cdot)$ is empty, and hence $= 1$, and the above expression reduces to the previous one for FG's, Boltzmann distributions, MRF's, and BN's.

Identifying individual factors between Equations 3.2 and 3.1, we can find the "elimination map" $\mathcal{E} : \mathcal{D}(f, d, \gamma; \mathcal{F}) \to \mathcal{D}(f, d; \mathcal{F})$ such that $\Psi \circ \mathcal{E} = \Psi$ on $\mathcal{D}(f, d, \gamma; \mathcal{F})$. This map eliminates all gating links in favor of f and d links by replacing every instance of a gating link from variable x_i to a factor ϕ_κ with a dependency (f or d) link between the same two nodes. If possible, this link could be a conditional dependence (d) link,

though this may require local re-normalization of ϕ_κ. In general, however, f links could be used; this only requires changing the probability factor ϕ_κ by raising it to the power of the product all incident 0/1 gating variable values, assuming only that this is possible within the function class $\mathcal{F}$. The ungated graphical model resulting from $\mathcal{E}$ has the same "meaning" (maps under Ψ to the same joint distribution on the same random variables) as the gated one, so $\Psi \circ \mathcal{E} = \Psi$ on $\mathcal{D}(f, d, \gamma; \mathcal{F})$, but is devoid of gating links. This elimination was previously described (in much the same detail) in Mjolsness [51].

The two notations represent equivalent sets of functions, but the Dependency Diagram with gating makes the relationship more explicit, and in that sense provides a more precise representation for domain models.

3.4 Index Nodes and Links

In structured applications involving time, space, or other architectural regularities, conventional algebraic notation makes use of subscripts or indices to simplify representations. The semantics of such indices is formalized within Dependency Diagrams by the introduction of *index nodes*, which allow a fixed amount of network description to specify an arbitrary or variable number of random variables and dependencies. Index nodes, first described in Mjolsness [50], are a reformulation and extension of the plates notation of Buntine [11]. Whenever variables or functions would algebraically appear with subscripts indicating multiplicity, a Dependency Diagram will have an index node with an index link to the corresponding variable or function node.

Once again we introduce a new construct into the semantics by first defining an appropriate adjacency matrix. Let $\iota(j, \alpha) = 1$ if there is an index link from index

node α to another node j. Since factor and variable nodes are indexed by two different sets, it is necessary to specify both $\iota(i,\alpha) \in \{0,1\}$ and $\iota(\kappa,\alpha) \in \{0,1\}$, where i indexes the variable nodes x_i, α indexes the index nodes a_α, and κ indexes the probability factor nodes ϕ_κ.

An additional sparse matrix $\hat{\iota}(\kappa,\alpha)$ records relationships between an index node a_α and a factor ϕ_κ which is not replicated by the index a_α, but rather receives a vector of arguments indexed by a_α. We refer to this as an *argument indexing* link. If an index node is connected to a factor node via an argument indexing link, then all of the copies of any variables replicated by that indexing link are passed to that factor as vectors. This is the difference between indexing, which produces a product such as $\prod_i f(x_i)$, and indexing combined with argument indexing, which produces a vector of arguments such as $f([x_i])$.

Of course, the symbols α, i, and κ above are not themselves index nodes but rather are part of the mathematical language we are using to describe Dependency Diagrams. Recall that a node *grade* might be represented in the semantics as x_1. Now suppose that the diagram contains grades for many students and many courses, represented by index nodes *student* and *course*, each connected to the *grade* node by an index link. Then the model would refer to an indexed variable $grade_{student,course}$. Then *student* and *course* would be represented by a_α, with $\alpha \in \{1,2\}$ and $\alpha = 1$ for *student* and $\alpha = 2$ for *grade*, and $grade_{student,course}$ would be represented in the semantics by $x_{1_{a_1,a_2}}$. density formula. For every node, define $J(\cdot)$ as the set of indices attached to that node. Following the conventions developed earlier, restrict κ to run over factors and i and j to run over variables, so that $J(\kappa) = [a_\mu|\iota(\kappa,\mu) = 1]$, $J(i) = [a_\alpha|\iota(i,\alpha) = 1]$, and $J(j) = [a_\beta|\iota(j,\beta) = 1]$. Next define unordered index sets

specific to each factor, over which products will be taken:

$$L_1(\kappa) = \left(\cup_{i|F_{i\kappa}=1}\{a_\alpha|\iota(i,\alpha)=1\}\right) \cup \{a_\mu|\iota(\kappa,\mu)=1\}\backslash\{a_\nu|\hat{\iota}(\kappa,\nu)=1\},$$
$$L_2(\kappa) = \left(\cup_{i|D_{i\kappa}=1}\{a_\alpha|\iota(i,\alpha)=1\}\right) \cup \{a_\mu|\iota(\kappa,\mu)=1\}\backslash\{a_\nu|\hat{\iota}(\kappa,\nu)=1\}. \quad (3.3)$$

The first term of the definition of L_1 says that for factor node κ, $L_1(\kappa)$ is the set of indices a_α which are connected to variables x_i such that variable x_i is also connected to factor ϕ_κ via an undirected dependency. This set is then unioned with the set of indices μ which are connected directly to κ. Finally, the set of indices ν connected to κ via an argument indexing link ($\hat{\iota}$) are removed from the set. L_2 is defined similarly for variables connected via directed, rather than undirected, links.

We must also define argument lists for the factors. Let $R_\mu = \{a_\mu^{\min}, \ldots a_\mu^{\max}\}$ be the range of integer-valued index a_μ. Then define:

$$A(\kappa) = [x_{i,J(i)}|F_{i\kappa}=1\|i\in\mathcal{I}, \{a_\mu\in R_\mu|\hat{\iota}(\kappa,\mu)=1\}]$$
$$B(\kappa) = [x_{i,J(i)}|D_{i\kappa}=1\|i\in\mathcal{I}, \{a_\mu\in R_\mu|\hat{\iota}(\kappa,\mu)=1\wedge\iota(i,\mu)=1\}]$$
$$C(\kappa) = [x_{j,J(j)}|D_{\kappa j}=1\|j\in\mathcal{I}, \{a_\mu\in R_\mu|\hat{\iota}(\kappa,\mu)=1\wedge\iota(j,\mu)=1\}].$$

Then the semantics for $\mathbb{D}\in\mathcal{D}(f,d,\gamma,\iota,\hat{\iota};\mathcal{F})$ is given by

$$\Psi(\mathbb{D}) = \Pr(\mathbf{x}) = \tfrac{1}{Z}\prod_{\kappa\in\mathcal{K}_1}\left\{\prod_{\{a_\alpha|\alpha\in L_1(\kappa)\}}\phi_{\kappa,J(\kappa)}(A(\kappa))\right\}$$
$$\times\prod_{\kappa\in\mathcal{K}_2}\left\{\prod_{\{a_\alpha|\alpha\in L_2(\kappa)\}}\phi_{\kappa,J(\kappa)}(B(\kappa)|C(\kappa))\right\}. \quad (3.4)$$

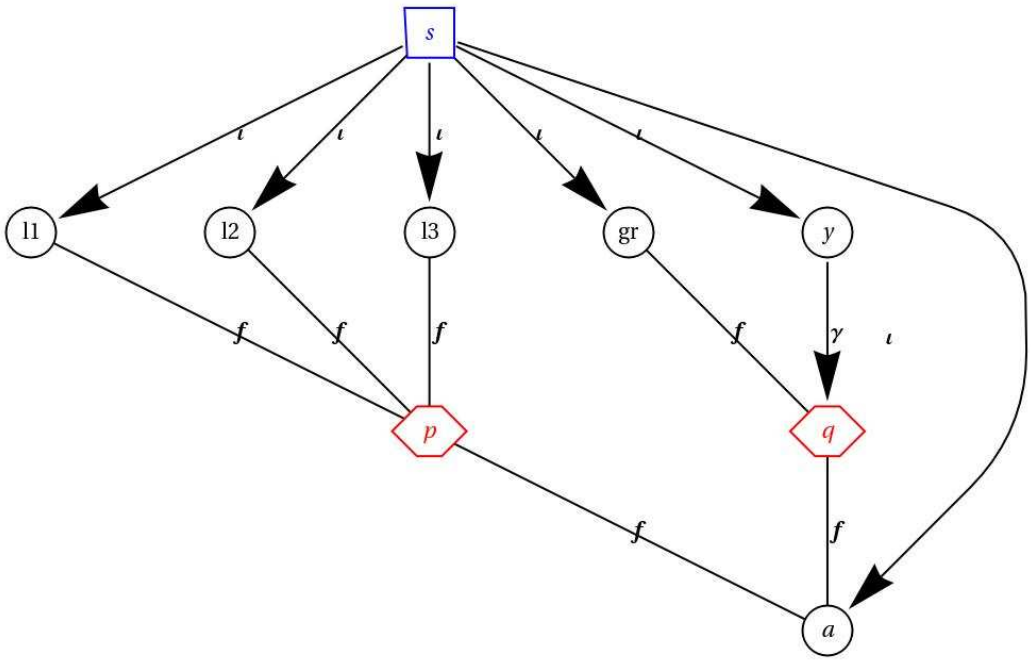


Figure 3.3: Dependency Diagram for graduate admissions, with indexing.

To illustrate this concept, consider once again the graduate admissions model. So far this model has only described the chances for admission associated with a single student named Alice. Index nodes and links provide a way to model every student at once. Figure 3.3 includes a new node, index s, which creates a copy of each variable for every student.

Each product over indices acts within a local scope associated with one factor node, and the resulting factors multiply. This differs from the single global scope proposed in Mjolsness [51], and simplifies the examples of Section 4.

The elimination map $\mathcal{E} : \mathcal{D}(f, d, \iota, \hat{\iota}; \mathcal{F}) \to \mathcal{D}(f, d; \mathcal{F})$ can be defined by observing that, once nested products are flattened out, Equation 3.4 and Equation 3.1 have the same form. Thus, $\Psi \circ \mathcal{E} = \Psi$ on $\mathcal{E} : \mathcal{D}(f, d, \iota, \hat{\iota}; \mathcal{F})$. This allows gating and index elimination to be combined easily.

Index and argument-index links in Dependency Diagrams generalize plates in several

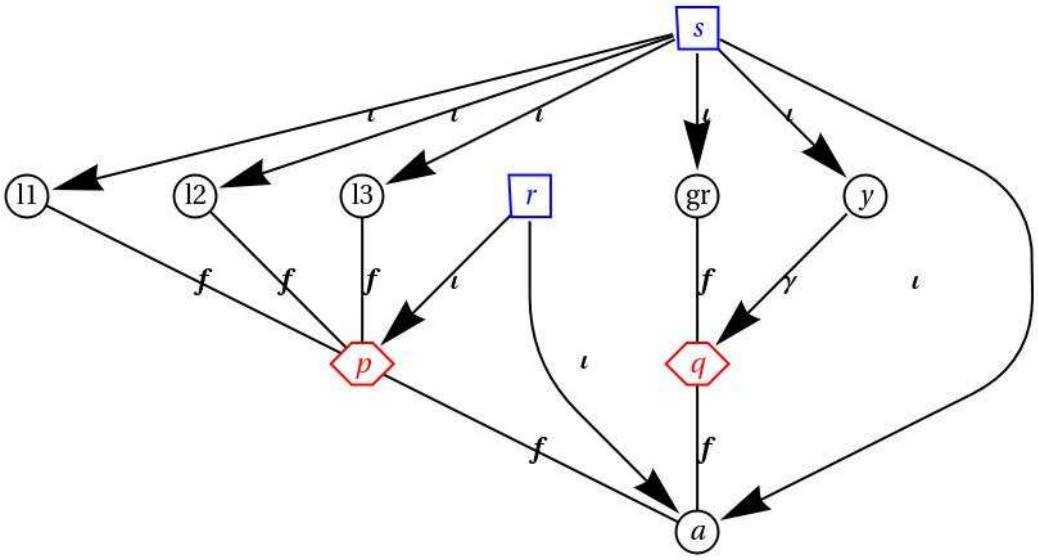


Figure 3.4: Dependency Diagram for graduate admissions, with multiple reviewers.

ways. Notably, flexible sharing of factors is possible: there is no longer a constraint that the interaction factors associated with two variables that are connected by a dependency link and both are indexed by the same index a_α are always (or are always not) similarly indexed. The degree of factor sharing is specified flexibly by $\iota(i, \alpha) \in \{0, 1\}$ and $\iota(\kappa, \alpha) \in \{0, 1\}$. The utility of this added power can be seen by modifying the admissions example to account for multiple reviewers who may each consider each student's application. Assuming that every reviewer will interpret a GPA in the same way, there is still a need to account for each reviewer's interpretation of letters of recommendation. This can be achieved by adding a new index, r, which replicates the decision variable and the factor associated with letters, but nothing else. This is shown in Figure 3.4.

In addition, there is no constraint that all nodes indexed by a common index node must be arranged "inside" a compact grouping in a two dimensional layout. When these groupings have partial overlap, the problem of layout becomes that of drawing a Venn diagram for a lattice of many subsets.

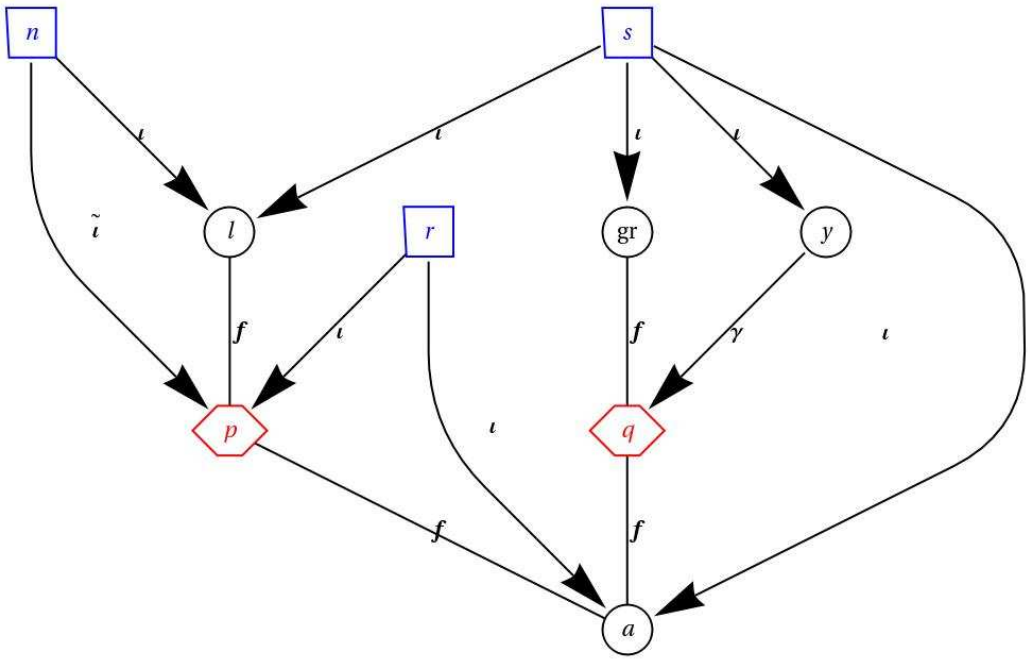


Figure 3.5: Dependency Diagram for graduate admissions, with argument indexing.

A final refinement of the admissions model underlines the utility of the argument-indexing link described above. Referring to Figure 3.4, it is apparent that the model still contains unnecessarily repeated elements. Since each student has three letters of recommendation, the model could be simplified further if one node l represented all of the letters. This is possible if l is replicated by a new index node n. However, to do this, it is necessary to have a way to specify that all of a student's letters are arguments to the same factor p, rather than each being an argument to only its own factor. Argument indexing allows this, as in Equations 3.3 and 3.4. Applying the argument index link from n to p, as shown in Figure 3.5, achieves the intended effect. Now all variables indexed by n are grouped on that index before being passed to the factor p.

3.5 Secondary Link Types

Another elimination map $\mathcal{E} : \mathcal{D}(f, d, \delta, \iota, \hat{\iota}; \mathcal{F}) \rightarrow \mathcal{D}(f, d, \iota, \hat{\iota}; \mathcal{F})$ allows one to add constraint nodes that can be reduced to factor nodes. For simplicity we just consider $\mathcal{E} : \mathcal{D}(f, d, \delta; \mathcal{F}) \rightarrow \mathcal{D}(f, d; \mathcal{F})$. If some factor nodes connected to variables according to sparse matrix $F_{i\kappa}$ are replaced by constraint nodes connected via sparse matrix $\tilde{F}_{i,\kappa}$, and if $\delta\left(g_{\kappa,J(\kappa)}(\cdot)\right)$ is a suitable Kronecker or Dirac delta distribution composed with a function $g_{\kappa,J(\kappa)}(\cdot)$ of the random variables, then we can augment the semantics of Equation 3.1 with the extra factor $\prod_{\kappa \in \mathcal{K}(\delta)} \delta(g_{\kappa,J(\kappa)}([x_{i,J(\kappa)}|\tilde{F}_{i,\kappa} = 1]))$. This notation goes beyond function nodes [11] to encompass relation nodes, due to the presence of undirected δ links specified by matrix $\tilde{F}$, and the functions g_κ. The elimination map is specified by relating the augmented equation to Equation 3.1 and identifying $\phi_{\kappa,J(\kappa)}(\cdot)$ with $\delta\left(g_{\kappa,J(\kappa)}(\cdot)\right)$. A function node described by $\delta(x_{i(\kappa)} = f_{\kappa,J(\kappa)}(\cdot))$ can also be normalized and moved to the directed node portion of Equation 3.1. The following generalizations of the indexing mechanism can be added to DDs, go further beyond factor graphs and plates, and are straightforward to express in terms of more general probability formulas: (a) numerical index constraint links δ_ι; (b) multiple levels of indexing (subscripts on subscripts); (c) random variable existence links ϵ; (d) reordering of indices to express transpose operations; (e) value type links for variables and indices, which specify in what measure space each random variable takes its values; (f) constraints relating random variable values and index values, such as variable upper index limits $a_\alpha^{\max}$, which can be variable (rather than constant as assumed so far), to obtain variable-structure systems of unbounded size.

Of these extensions, the essential one is (a) numerical index constraint links δ_ι, as the power to describe Markov Chain Monte Carlo algorithms graphically is of particular interest. Index constraints are accommodated within the semantics by a factor omission formula similar to Equation 3.2. In the context of automatically generated

sampling code, index constraints may be used to avoid unnecessarily looping over invalid combinations of indices. Many of the other extensions are discussed in Mjolsness [51], but not required by the examples of Section 4, so they are omitted.

3.6 Inference and Sampling Via DD Transformation

Because Dependency Diagrams have been consistently defined by reduction to a form equivalent to factor graphs, all of the algorithms for inference developed for other graphical models would work just as well for Dependency Diagrams. However, the Dependency Diagram framework also opens up the possibility of new inference schemes. Here we begin to consider one such method: the transformation of a model diagram into a diagram that represents an inference algorithm. In particular, we describe a process which maps a Dependency Diagram $\mathbb{D}$ representing a model into another diagram $\mathbb{D}_{\mathrm{MC}}$ representing a Markov chain that samples the distribution described by $\mathbb{D}$. This diagram $\mathbb{D}_{\mathrm{MC}}$ has the attractive properties that it is entirely directed and can easily be converted into runnable computer code that generates the Markov chain. Taken together, this results in a way of automatically sampling (and therefore performing inference on) any distribution that can be described by a Dependency Diagram.

This process begins by applying a map

$$\mathcal{T} : \mathcal{D}(f, d, \gamma, \iota, \hat{\iota}, \delta_\iota; \mathcal{F}) \to \mathcal{D}(d, \gamma, \iota, \hat{\iota}, \delta_\iota; \mathcal{F})$$

to transform a Dependency Diagram $\mathbb{D} \simeq (F, D, \gamma, \iota, \hat{\iota}, \delta_\iota; \phi)$ into another Dependency Diagram $\mathbb{D}_{\mathrm{MC}}$ representing a Markov chain that samples the distribution represented

by $\mathbb{D}$ via the Metropolis-Hastings algorithm [28].

We will illustrate this process by reference to the model shown in Figure 3.6, which describes the most basic sort of Bayesian inference task. The data, represented by variable x, is observed. This data is drawn from a distribution llh, which specifies the likelihood of a model parameter μ. This likelihood has an additional fixed parameter σ, and the free parameter is assumed to be drawn from a prior distribution prr. Given some fixed values of x and σ, the standard task is to represent the posterior distribution of μ. In cases when the prior and likelihood distributions are not conjugate, one approach is to draw a set of samples which approximates this distribution.

The Metropolis-Hastings diagram $\mathbb{D}_{\text{MC}}$ begins with two indices not in the original diagram, with an index constraint between them which requires that the first is always one less than the second. These indices represent a numbering of the successive samples in the Markov chain. Because we will refer to them often, it will be useful to call these the *time indices*. This is depicted in Figure 3.7.

The diagram $\mathbb{D}_{\text{MC}}$ then gets two copies of each variable. The first time index is connected to one copy of each variable in the diagram and the second time index to the second copy of each variable. Further indices are added to give each variable the correct dimension. Figure 3.8 depicts this, though the variable does not have any indices.

$\mathbb{D}_{\text{MC}}$ describes a Markov chain Monte Carlo algorithm that makes successive sweeps over the variables of $\mathbb{D}$ and updates each in turn. Singleton variables (those not connected to indices) are updated at each iteration. For each indexed set of variables the algorithm must pick a random index to update. So, for each dimension of each variable $\mathbb{D}_{\text{MC}}$ gets a new "index selection" variable, which has as its domain the integer range of the relevant index. Each such variable is connected to a factor specifying

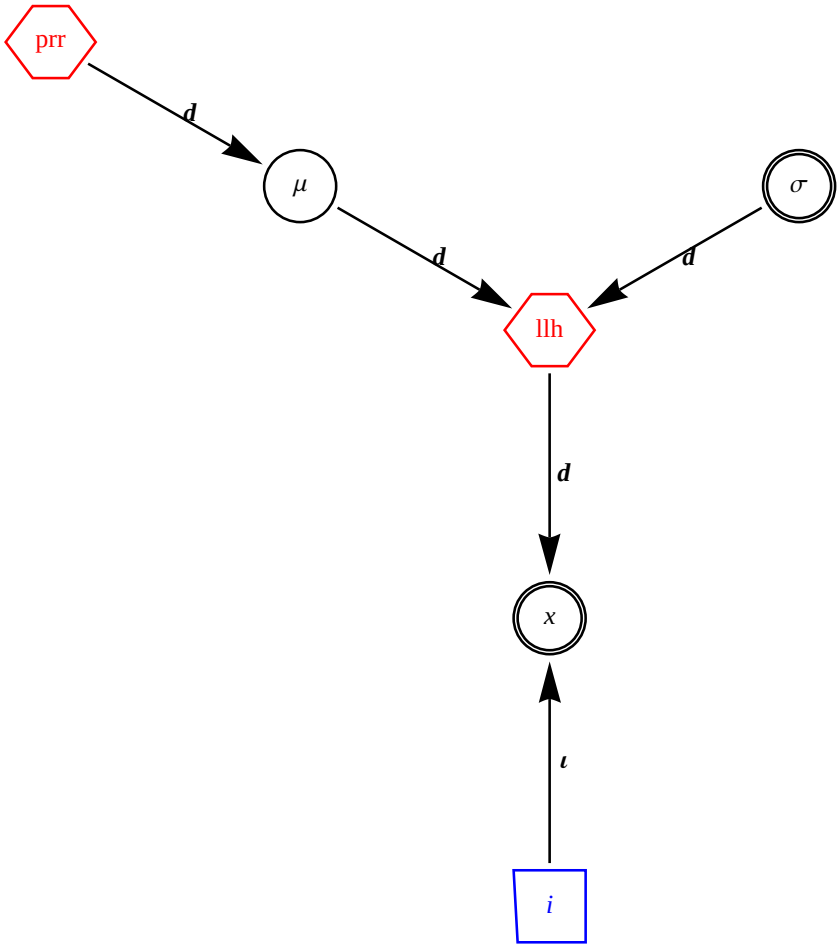


Figure 3.6: Simple Bayesian Model

Figure 3.7: Index nodes for sampling time.

Figure 3.8: Index nodes for time, attached to model variables.

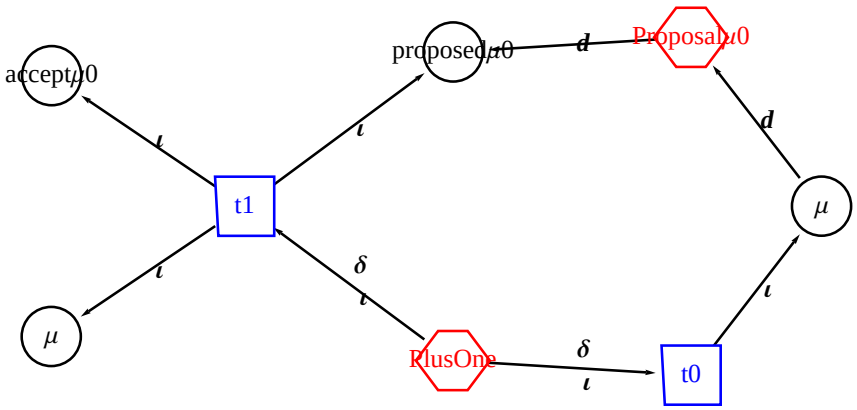


Figure 3.9: Adding proposed updates and acceptance indicators.

a uniform distribution over the appropriate range. Because the model in Figure 3.6 does not have indexed variables, this step is not depicted.

Next, for each variable x_i, $\mathbb{D}_{\text{MC}}$ gets a new variable $x_{i_{prop}}$ that has the same domain as x_i and represents a proposed move for the variable. In the example model, the variable μ is associated with $proposed\mu 0$, which is drawn in Figure 3.9. Many proposal distributions are possible for any given domain, though in order to simplify computation of the probability of accepting the proposal (described below) we make the assumption that the proposal distribution is symmetric [14]. Our implementation provides symmetric default proposal distributions for all of the variable types that it understands (discrete sets, integer values, and real values), but these can be overridden. Each proposal variable $\mathbb{D}_{\text{MC}}$ also gets a factor node representing the proposal distribution connected with a d-link in from the copy of the variable x_i which is linked to the first time index and add a d-link out to the proposal variable $x_{i_{prop}}$. In the example, this is represented by the node labeled $Proposal\mu 0$ in Figure 3.10.

The diagram then gets another new variable, $x_{i_{accept}} \in \{0, 1\}$, as well as a factor to govern this variable, which represents the probability of accepting the proposed

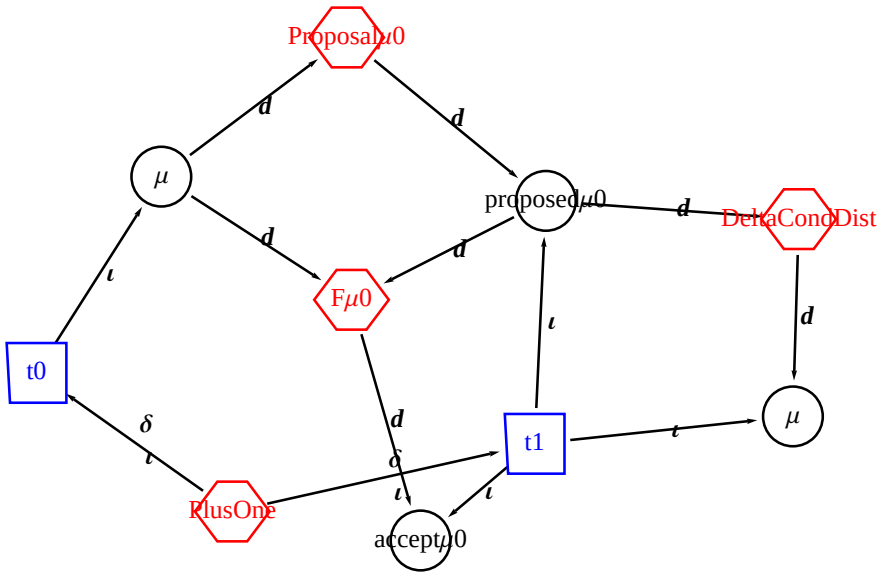


Figure 3.10: Adding proposal distributions.

moved. The variable is first drawn in Figure 3.9, named *accept*μ0, and the factor is drawn in Figure 3.10, named $F\mu 0$. The value of this factor is the smaller of one and the ratio of the full joint probability density (JPD) with the proposed variable in place of the variable being considered for an update and the JPD at the current time step, as this determines the probability of accepting a proposed move in the Metropolis-Hastings sampling algorithm. Each such probability factor receives a directed conditional dependence d-link from x_i and x_i's Markov blanket at the current time step, and has an outgoing d-link to $x_{i_{accept}}$ in the next time step. This is shown in Figure 3.10, where the parents of $F\mu 0$ are the variable μ at time $t0$ and the proposed update *proposed*μ0 at time $t1$, while its child is the acceptance indicator *accept*μ0. This ratio is automatically simplified by a finite difference package we have implemented in the *Mathematica* computer algebra system, within which we have also implemented Dependency Diagrams.

As an example, suppose that the likelihood llh is taken to be a Gaussian distribution

with $\sigma = 25$, and the prior prr is uniform on the interval $(10, 35)$ – this is a model with some domain knowledge represented in the prior, but significant uncertainty. The posterior distribution of μ is also confined to the region $(10, 35)$, and if $\delta\mu = \mu_{\text{t1}} - \mu_{\text{t0}}$ then the acceptance probability is given by

$$
\begin{aligned}
&F\mu 0(accept\mu 0|\mu_{\text{t0}}, \text{proposed}\mu 0, x, \sigma) \\
&= \frac{e^{-N\delta\mu^2 + 2N\delta\mu\mu_{\text{t0}} - 2\delta\mu\sum_{i=1}^{N} x[[i]]\big/2\sigma^2} \left(\begin{cases} \frac{1}{25} & 10 \leq \text{proposed}\mu 0 \leq 35 \\ 0 & \text{True} \end{cases} \right)}{\begin{cases} \frac{1}{25} & 10 \leq \mu_{\text{t0}} \leq 35 \\ 0 & \text{True} \end{cases}}.
\end{aligned}
$$

The implementation creates a *Mathematica* function which computes this probability. A future improvement might note that this update could be simplified from linear in N to constant time by pre-computing and storing the sum over the values of observations x_i.

Next, for the copy of each variable linked to the second time index, the diagram gets parent factors which relate the past-time and proposed values to the future-time value for the variable. These factors are gated by the indicator variables, so that the future-time variable is deterministically set to the past-time value if the proposal was rejected, or is deterministically set to the proposed value if the proposal was accepted. This is depicted in Figure 3.11, where *DeltaCondDist* is a delta distribution which assigns the child the value of the parent with probability 1.

Finally, the constants are added, and attached as parents of acceptance probabilities which depend on them, as shown in Figure 3.12. This results in a diagram which represents the process of drawing samples from the posterior distribution over μ. The variables μ in this diagram form a list which in aggregate has the same distribution as

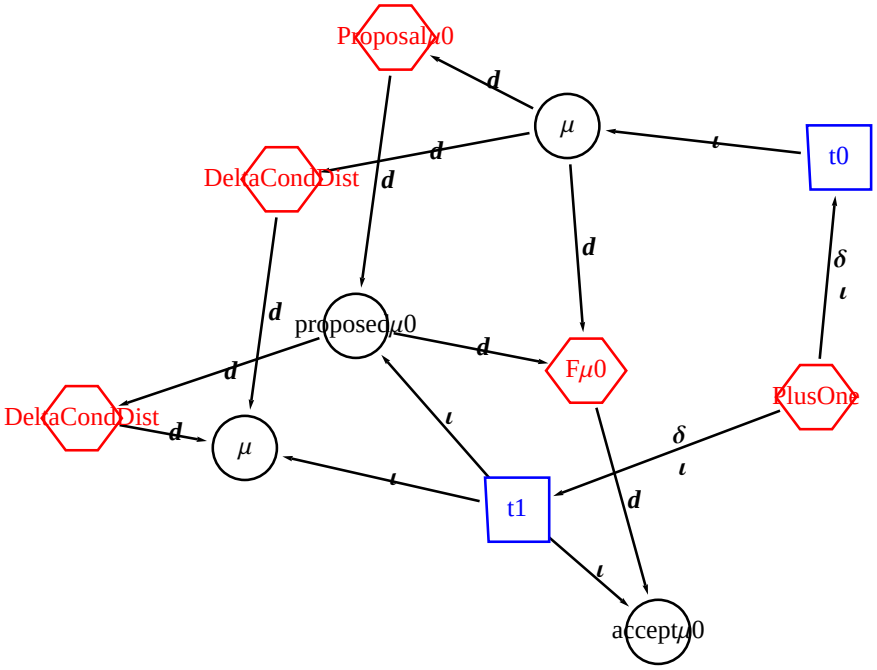


Figure 3.11: Adding deterministic value-setting for μ_{t1}.

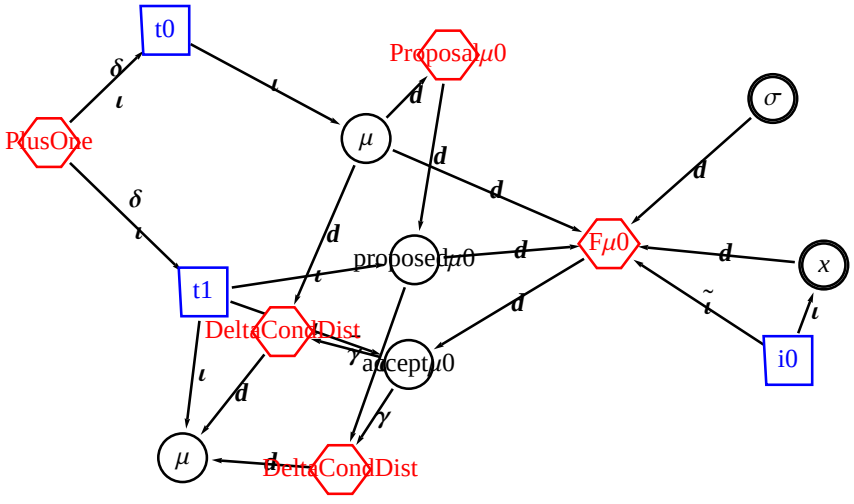


Figure 3.12: Adding constants.

that in Figure 3.6, but this diagram can be sampled-from in a trivial way, via forward sampling [40]. Because every function node represents a directed conditional probability, and evidence is present only at the root of the directed diagram, the variables can be sampled by sorting nodes topologically and drawing each child conditioned on its parents.

Therefore, this graph-transformed diagram $\mathbb{D}_{\text{MC}}$ can then be translated into an algorithm for creating an MCMC sample of the original distribution. Specifically, a trivial algorithm to perform forward sampling of a directed graphical model, combined with indexing and gating, when applied to $\mathbb{D}_{\text{MC}}$, has the effect of performing Metropolis-Hastings MCMC on the distribution specified by the original diagram.

The implementation of Dependency Diagrams will also produce runnable, reusable code to sample any diagram which is ammenable to forward sampling. Thus, runnable code for sampling any diagram can be generated by first creating the corresponding MCMC diagram, and then creating the forward sampling code. For the diagram in Figure 3.12 the generated code is showin in Figure 3.13.

The transformed diagram could also be used as a starting point for a user to craft a custom MCMC algorithm for a particular model. This could be achieved in a number of ways, but in particular by defining custom proposal distributions and by editing the links and nodes in the diagram $\mathbb{D}_{\text{MC}}$. In this way, the Dependency Diagram software may be able to speed development of MCMC algorithms by allowing a user to begin with a default template and only customize sections of interest or in which model-specific speed-ups could be achieved.

All of these steps are programmed and performed within the commercial *Mathematica* computer algebra system. Equivalent implementations could be created in other commercial or noncommercial computer algebra systems.

```
Function[{constants0},
 Module[{initializevariables = True, acceptMu1, i1, proposedMu1,
         t2, t3, x, Mu, Sigma},
    t3min1 = 1; t3max1 = ITERATIONS;
    t2min1 = 1; t2max1 = ITERATIONS;
    i1min1 = 1; i1max1 = numData;
    x = Symbol["x"] /. constants0;
    Sigma = Symbol["Sigma"] /. constants0;
  (* initialize an empty list of the appropriate size for each
     variable *)
  If[initializevariables,
   Mu = IndListToTable[{{t2min1, t2max1}}];
   proposedMu1 = IndListToTable[{{t3min1, t3max1}}];
   acceptMu1 = IndListToTable[{{t3min1, t3max1}}];
   For[t3 = Min[t3min1, 1 + t2min1],
       t3 <= Max[t3max1, 1 + t2max1], t3++,
    If[(t3 < Max[t3min1, 1 + t2min1] ||
        t3 > Min[t3max1, 1 + t2max1]) && t3 >= t3min1 && t3 <= t3max1,
      proposedMu1[[t3]] =
       RandomReal[UniformDistribution[{10, 35}]];];
    If[(t3 < Max[t3min1, 1 + t2min1] ||
        t3 > Min[t3max1, 1 + t2max1]) && t3 >= t3min1 && t3 <= t3max1,
      acceptMu1[[t3]] =
       RandomInteger[DiscreteUniformDistribution[{0, 1}]];];
    If[(t3 < Max[t3min1, 1 + t2min1] ||
        t3 > Min[t3max1, 1 + t2max1]) && t3 >= t3min1 &&
      t3 <= t3max1, Mu[[t3]] =
       RandomReal[UniformDistribution[{10, 35}]];];];
  For[t3 = Max[t3min1, 1 + t2min1], t3 <= Min[t3max1, 1 + t2max1], t3++,
      t2 = -1 + t3;
      proposedMu1[[t3]] = RandomReal[ProposalMu1[Mu[[t2]]]];
      acceptMu1[[t3]] =
        RandomInteger[FMu1[proposedMu1[[t3]],
                           DDCollect[x, {{i1min1, i1max1}}],
                           Mu[[t2]], Sigma]];
      If[acceptMu1[[t3]] > 0, 1, 0] == 1,
         Mu[[t3]] = RandomReal[DeltaCondDist[proposedMu1[[t3]]]];
      If[1 - If[acceptMu1[[t3]] > 0, 1, 0] == 1,
         Mu[[t3]] = RandomReal[DeltaCondDist[Mu[[t2]]]];];
  ];
  {ToExpression["acceptMu1"] -> acceptMu1,
   ToExpression["proposedMu1"] -> proposedMu1,
   ToExpression["Mu"] -> Mu}
]]
```

Figure 3.13: Runnable code.

For the graph transformation $\mathcal{T}$ to MCMC algorithms, the essential DD link types are indexing and index-index constraints: ι, δ_ι. These link types enable the unambiguous symbolic expression of iterative Markov chain algorithms, without unrolling the time dimension, and thereby enable automatic generation of runnable code.

Chapter 4

Dependency Diagrams: Implementation and Experiments

This section describes experiments on several probabilistic models, such as the real-world ALARM Bayesian network, a mixture of Gaussians clustering model, and the inference of reaction rate parameters. Finally, a model of prior distributions on graph connectivity is explored in more detail. We have also applied the method described in Section 3.6 to several simple models not discussed at length, including a one-dimensional Ising model and a cigar-shaped multivariate Gaussian [14].

Table 4.1 outlines the rendering of labeled graphs for DD's into pictorial representations in the figures that follow. The link types f, d, γ, δ, ι, $\hat{\iota}$, and δ_ι correspond to adjacency matrices F, D, γ, $\tilde{F}$, ι, $\hat{\iota}$, and δ_ι, respectively, in the semantics. The node types are rendered as circle, double circle, hexagon, square for random variable, model parameter, factor, index respectively. Many other renderings are possible. Rendering factors as elongated hexagons is nonstandard (small filled rectangles are standard) but allows room for labels and formulas to be enclosed within the nodes

Table 4.1: One possible rendering of Dependency Diagrams

Algebraic expressions	Full name	Rendering
F	factor link	f link
D	conditional dependency	d link
ι	index link	ι link
$\hat{\iota}$	argument index link	$\hat{\iota}$ link
γ	gating link	γ link
δ_ι	index constraint link	δ link
x	random variable	circle node, x
θ = various symbols	model parameter	double circle node, θ
ϕ	factor	hexagon node, ϕ
a	index	square node, a

and distinguishes them from index nodes, which are rectangular like the plates they replace.

4.1 Implementation

Our implementation covers the Dependency Diagram class $\mathcal{D}(f, d, \gamma, \epsilon, \iota, \hat{\iota}, \delta_\iota; \mathcal{F})$, which is to say those diagrams built upon any subset of the following link types: directed and undirected dependency, gating, existence (see Section 3.5, item (c)), indexing and argument indexing. The source code, in the form of a *Mathematica* package and a set of notebooks defining example models, is available from `http://computableplant.ics.uci.edu/sw/dd/`.

The software package allows Dependency Diagrams to be specified using the `Graph` data structure built in to *Mathematica*. The package includes the ability to parse DDs, evaluating them according to the semantics outlined in Chapter 3 and returning an algebraic expression for the defined probability distribution. The package also renders Dependency Diagrams graphically, using shapes, colors, and labels to indicate the various node and link types.

The Dependency Diagram software also includes a parser for the BIF (Bayesian Interchange Format) Bayesian network specification language, which reads a BIF file, defines *Mathematica* functions for each of the conditional probability distributions described therein, and returns a Dependency Diagram which implements the described Bayesian network.

Because Dependency Diagrams subsume Markov random fields, the package implements an algorithm, due to Bron and Kerbosch, for efficiently enumerating cliques of a graph [9]. This allows a user to specify an undirected diagram without explicitly representing function nodes. The DD software will transform such a diagram, adding a factor for each clique, so that the user can subsequently provide *Mathematica* function definitions for the generated factors.

The Dependency Diagram software package includes an implementation of the graph transformation from a model diagram into a diagram representing a sampling procedure, as described in Section 3.6. In conjunction with this, the package implements a sophisticated finite-difference engine for automatically reducing the algebraic form of each MCMC proposal acceptance probability as much as possible. The software also includes the ability to generate runnable code for generating samples from the distribution specified by any diagram that consists entirely of directed arrows without observed variable nodes ("evidence") as their children.

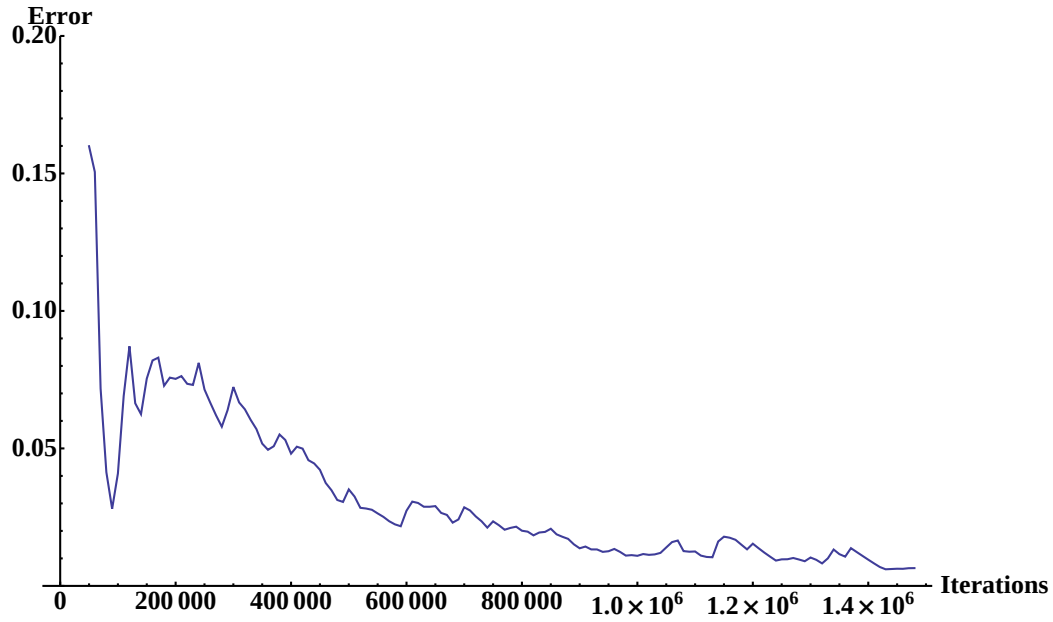
4.2 The ALARM Network

The ALARM Bayesian network [5] is a real-world model used widely in the literature on graphical models to test everything from model selection algorithms [29] to approximate inference techniques [e.g., 57, 44, 56]. The model was originally designed

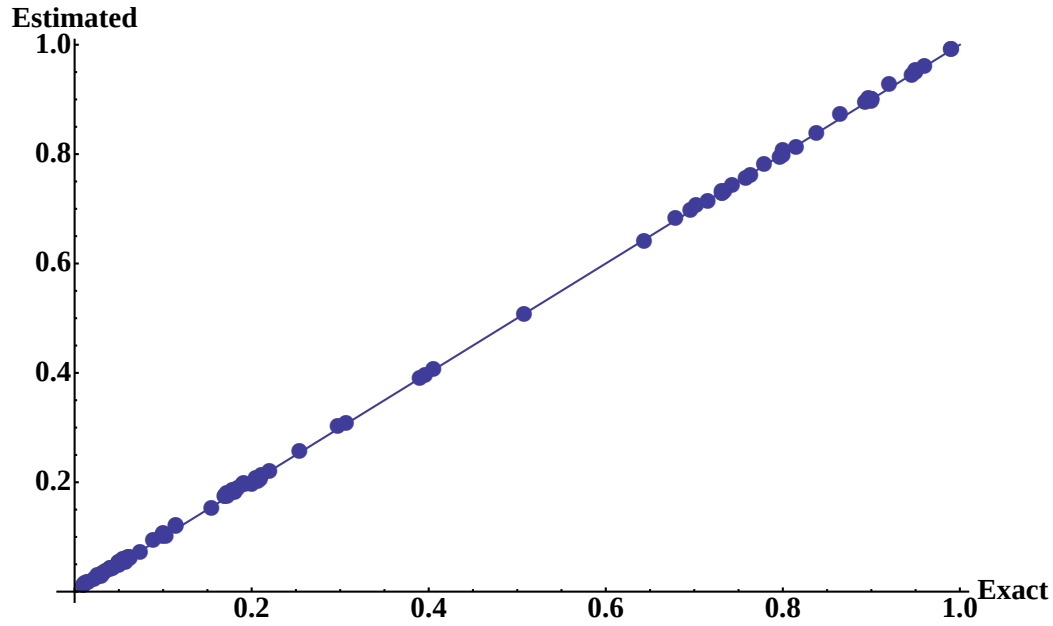
to provide relevant text messages to medical personnel based on patient history and information available to an anesthesia monitor, and so it represents the probabilities of eight possible diagnoses given sixteen findings and thirteen intermediate variables. In total, the network consists of thirty-seven discrete variables with domain sizes of two, three, and four. When the model is drawn as a factor graph, the largest factor has five neighbors.

In order to demonstrate that the autogenerated inference engines described in Section 3.6 are implemented correctly, we used such an auto-generated sampler to estimate the single-variable marginal probabilities for all of thirty-seven nodes in the ALARM network. Figure 4.1a shows the l_∞ error of this estimate plotted against number of MCMC iterations, where the l_∞ error measure is defined as largest difference in the exact and estimated probabilities for one value of a single-variable marginal. As expected, the error drops toward zero as the number of samples increases, demonstrating that the generated sampler is choosing from the target distribution.

Figure 4.1b plots the exact single-variable marginals for all 37 nodes in the original network versus estimates computed by running the automatically generated Metropolis-Hastings to generate 1,485,000 samples. The first 5,000 were discarded as burn-in, and the rest were simply counted. This plot was created at the end of the same run used to generate Figure 4.1a. As expected, all of the points are near the line $x = y$, indicating that the approximations are all nearly identical to the exact values.



(a) ℓ_∞ error across iterations of MCMC. As expected, the error diminishes toward zero with added samples.



(b) Estimated versus exact marginals, lying along the line $x = y$.

Figure 4.1: Estimation of ALARM network marginals

4.3 Mixture-of-Gaussians Clustering

A Dependency Diagram for an elementary Mixture-of-Gaussians clustering model is shown in Figure 4.2. Relevant node types are reviewed in the caption, but see Table 4.1 for a complete list of link and node types used. The model has a single probability factor node representing the Gaussian $N(x|\mu,\sigma) = G((x-\mu)/\sigma)$, gated by cluster membership indicator variables M_{ia}, which in turn have a unique cluster membership constraint $\sum_{a=1}^{a_{\max}} M_{ia} = 1$. This formulation of probabilistic clustering is related to "1-of-K encoding" or "neuron multiplexing" in the literature of optimization via neural networks [64]. Data vectors x_i are indexed by i; cluster centers μ_a are indexed by a, as are fixed standard deviations σ_a.

The semantics $\Psi(D)$ for this diagram approximates

$$\Pr(M,\mu|x,\sigma) = \frac{1}{Z}\left(\prod_{a=1}^{a_{\max}}\prod_{i=1}^{i_{\max}} N(x_i|\mu_a,\sigma_a)^{M_{ia}}\right)\left(\prod_{i=1}^{i_{\max}}\delta(\sum_{a=1}^{a_{\max}} M_{ia}-1)\right). \qquad (4.1)$$

The main factor in this diagram is $f(x|\mu,\sigma) = \prod_i N(x_i|\mu_a,\sigma_a)$, where N is the normal distribution with mean μ and standard deviation σ. The preference that each variable belong to a single cluster is enforced by a Gaussian of the form $N(\sum_a M_{ia}|\mu_M = 1, \sigma_M = 0.3)$, despite its integral arguments. This Dependency Diagram was automatically converted to another one for the MCMC algorithm (Figure 4.3). Here F_1, F_2, and F are symbolically autogenerated factors. Given a well-separated data set, the autogenerated MCMC algorithm for clustering finds the correct membership matrix M and draws a set of samples whose average approximates the true means μ_a.

Under the semantics defined in the previous chapter, the MCMC sampling diagram has this meaning:

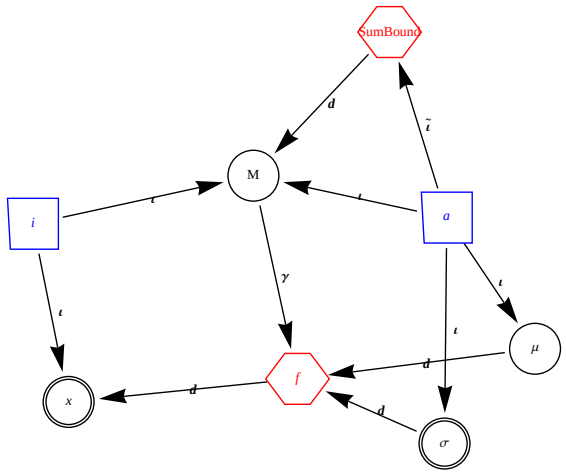


Figure 4.2: Dependency Diagram for the probabilistic clustering model. Random variables are drawn as circles. Double circles represent observations. Functions are drawn as hexagons, and index nodes are drawn as squares. Edges are labeled as in Table 4.1. SumBound implements the δ function on M, as in Equation 4.1.

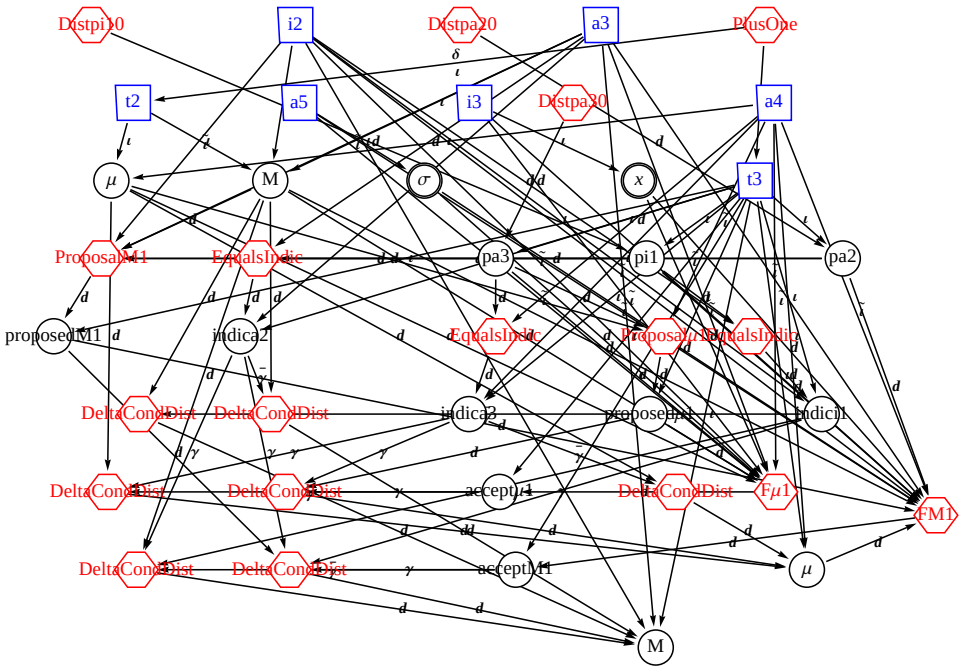
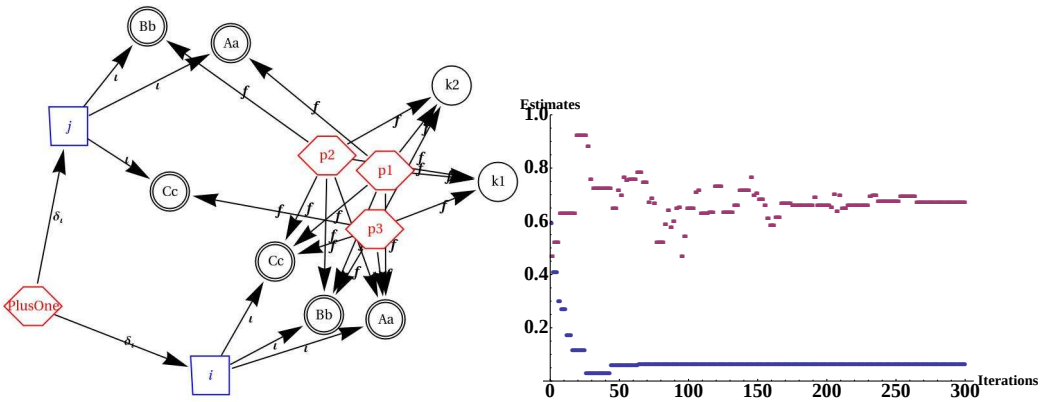


Figure 4.3: Automatically generated MCMC diagram for the clustering experiment. Several automatically generated functions and variables are present. DeltaCondDist implements a δ conditional distribution which assigns probability one to a particular value of the child. Proposal distributions for variables x are named Proposal$x$1. Random variables pXI select which member of an indexed set of variables, indexed by X, will be updated at the current iteration of MCMC. Because several variables may be indexed by X, a unique id I is generated for each such variable. EqualsIndic has as a parent a numerical index, and as a child a set of binary indicators, and places probability one on the appropriate bit being set to one. IndicOther has as a parent an indicator, and as a child another indicator, and forces them to disagree. Acceptance distributions Fx have as children indicator variables, and as parents the variables necessary to compute the acceptance probability of a proposed update for variable x. They assign this probability to their children.

$$\left(\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{Distpa20}[],\text{pa2}_{\text{t3}}\right]\right)$$
$$\times\left(\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{Distpa30}[],\text{pa3}_{\text{t3}}\right]\right)$$
$$\times\left(\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{Distpi10}[],\text{pi1}_{\text{t3}}\right]\right)$$
$$\times\left(\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{FM1}\left[M,\text{pa2}_{\text{t3}},\text{pi1}_{\text{t3}},\text{proposedM1}_{\text{t3}},x,\mu,\sigma\right],\text{acceptM1}_{\text{t3}}\right]^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{F}\mu\text{1}\left[M,\text{pa3}_{\text{t3}},\text{proposed}\mu\text{1}_{\text{t3}},x,\mu,\sigma\right],\text{accept}\mu\text{1}_{\text{t3}}\right]^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{ProposalM1}\left[M,\text{pa2}_{\text{t3}},\text{pi1}_{\text{t3}}\right],\text{proposedM1}_{\text{t3}}\right]^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}} \text{PDF}\left[\text{Proposal}\mu\text{1}\left[\text{pa3}_{\text{t3}},\mu\right],\text{proposed}\mu\text{1}_{\text{t3}}\right]^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{a4=a4min0}}^{\text{a4max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\begin{cases}1 & \text{proposed}\mu\text{1}_{\text{t3}} == \mu_{\text{a4,t3}}\\ 0 & \text{True}\end{cases}\right)^{\text{If[accept}\mu\text{1}_{\text{t3}}>0,1,0]\text{If}\left[\text{indica3}_{\text{a4,t3}}>0,1,0\right]}\right)$$
$$\times\left(\prod_{\text{a3=a3min0}}^{\text{a3max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\begin{cases}1 & (\text{a3} == \text{pa2}_{\text{t3}}\&\&\text{indica2}_{\text{a3,t3}} == 1)\parallel(\text{a3}\neq\text{pa2}_{\text{t3}}\&\&\text{indica2}_{\text{a3,t3}} == 0)\\ 0 & \text{True}\end{cases}\right)$$
$$\times\left(\prod_{\text{a4=a4min0}}^{\text{a4max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\begin{cases}1 & (\text{a4} == \text{pa3}_{\text{t3}}\&\&\text{indica3}_{\text{a4,t3}} == 1)\parallel(\text{a4}\neq\text{pa3}_{\text{t3}}\&\&\text{indica3}_{\text{a4,t3}} == 0)\\ 0 & \text{True}\end{cases}\right)$$
$$\times\left(\prod_{\text{i2=i2min0}}^{\text{i2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\begin{cases}1 & (\text{i2} == \text{pi1}_{\text{t3}}\&\&\text{indici1}_{\text{i2,t3}} == 1)\parallel(\text{i2}\neq\text{pi1}_{\text{t3}}\&\&\text{indici1}_{\text{i2,t3}} == 0)\\ 0 & \text{True}\end{cases}\right)$$
$$\times\left(\prod_{\text{a3=a3min0}}^{\text{a3max0}}\prod_{\text{i2=i2min0}}^{\text{i2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\begin{cases}1 & \text{proposedM1}_{\text{t3}} == M_{\text{a3,i2,t3}}\\ 0 & \text{True}\end{cases}\right)^{\text{If[acceptM1}_{\text{t3}}>0,1,0]\text{If}\left[\text{indica2}_{\text{a3,t3}}>0,1,0\right]\text{If}\left[\text{indici1}_{\text{i2,t3}}>0,1,0\right]}\right)$$
$$\times\left(\prod_{\text{a4=a4min0}}^{\text{a4max0}}\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\left(\begin{cases}1 & \mu_{\text{a4,t2}} == \mu_{\text{a4,t3}}\\ 0 & \text{True}\end{cases}\right)^{1-\text{If}\left[\text{indica3}_{\text{a4,t3}}>0,1,0\right]}\right)^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{a4=a4min0}}^{\text{a4max0}}\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\left(\begin{cases}1 & \mu_{\text{a4,t2}} == \mu_{\text{a4,t3}}\\ 0 & \text{True}\end{cases}\right)^{(1-\text{If[accept}\mu\text{1}_{\text{t3}}>0,1,0])\text{If}\left[\text{indica3}_{\text{a4,t3}}>0,1,0\right]}\right)^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{a3=a3min0}}^{\text{a3max0}}\prod_{\text{i2=i2min0}}^{\text{i2max0}}\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\left(\begin{cases}1 & M_{\text{a3,i2,t2}} == M_{\text{a3,i2,t3}}\\ 0 & \text{True}\end{cases}\right)^{1-\text{If}\left[\text{indica2}_{\text{a3,t3}}>0,1,0\right]}\right)^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\left(\prod_{\text{a3=a3min0}}^{\text{a3max0}}\prod_{\text{i2=i2min0}}^{\text{i2max0}}\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\left(\begin{cases}1 & M_{\text{a3,i2,t2}} == M_{\text{a3,i2,t3}}\\ 0 & \text{True}\end{cases}\right)^{1-\text{If}\left[\text{indici1}_{\text{i2,t3}}>0,1,0\right]}\right)^{\text{KroneckerDelta[1+t2,t3]}}\right)$$
$$\times\prod_{\text{a3=a3min0}}^{\text{a3max0}}\prod_{\text{i2=i2min0}}^{\text{i2max0}}\prod_{\text{t2=t2min0}}^{\text{t2max0}}\prod_{\text{t3=t3min0}}^{\text{t3max0}}\left(\left(\begin{cases}1 & M_{\text{a3,i2,t2}} == M_{\text{a3,i2,t3}}\\ 0 & \text{True}\end{cases}\right)^{(1-\text{If[acceptM1}_{\text{t3}}>0,1,0])\text{If}\left[\text{indica2}_{\text{a3,t3}}>0,1,0\right]\text{If}\left[\text{indici1}_{\text{i2,t3}}>0,1,0\right]}\right)$$



(a) Model diagram.

(b) Sampled parameter values as a function of iteration number. Annealing causes samples to converge to point estimates near the true values.

Figure 4.4: The rate parameter inference experiment.

4.4 Reaction Rate Inference

Consider the chemical reaction $A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C$, which behaves in a stochastic way for small numbers of molecules. We would like to infer forward and reverse rates k_1 and k_2 from a time series of observations of A_i, B_i, C_i measured at successive time intervals of length τ. A simplified model for this situation is shown in Figure 4.4a. It contains the factors relating earlier and later time steps in the manner of a stochastic differential equation:

$$p_1(A_{i+1}|A_i, B_i, C_i, k_1, k_2) = \mathcal{N}(A_{i+1}|A_i - A_i B_i k_1 \tau + C_i k_2 \tau, \sigma)$$
$$p_2(B_{i+1}|A_i, B_i, C_i, k_1, k_2) = \mathcal{N}(B_{i+1}|B_i - A_i B_i k_1 \tau + C_i k_2 \tau, \sigma)$$
$$p_3(C_{i+1}|A_i, B_i, C_i, k_1, k_2) = \mathcal{N}(C_{i+1}|C_i + A_i B_i k_1 \tau - C_i k_2 \tau, \sigma)$$

Short runs of the autogenerated MCMC algorithm (DD in Figure 4.5) without annealing gave parameter reconstruction with 15% error. This was accomplished by

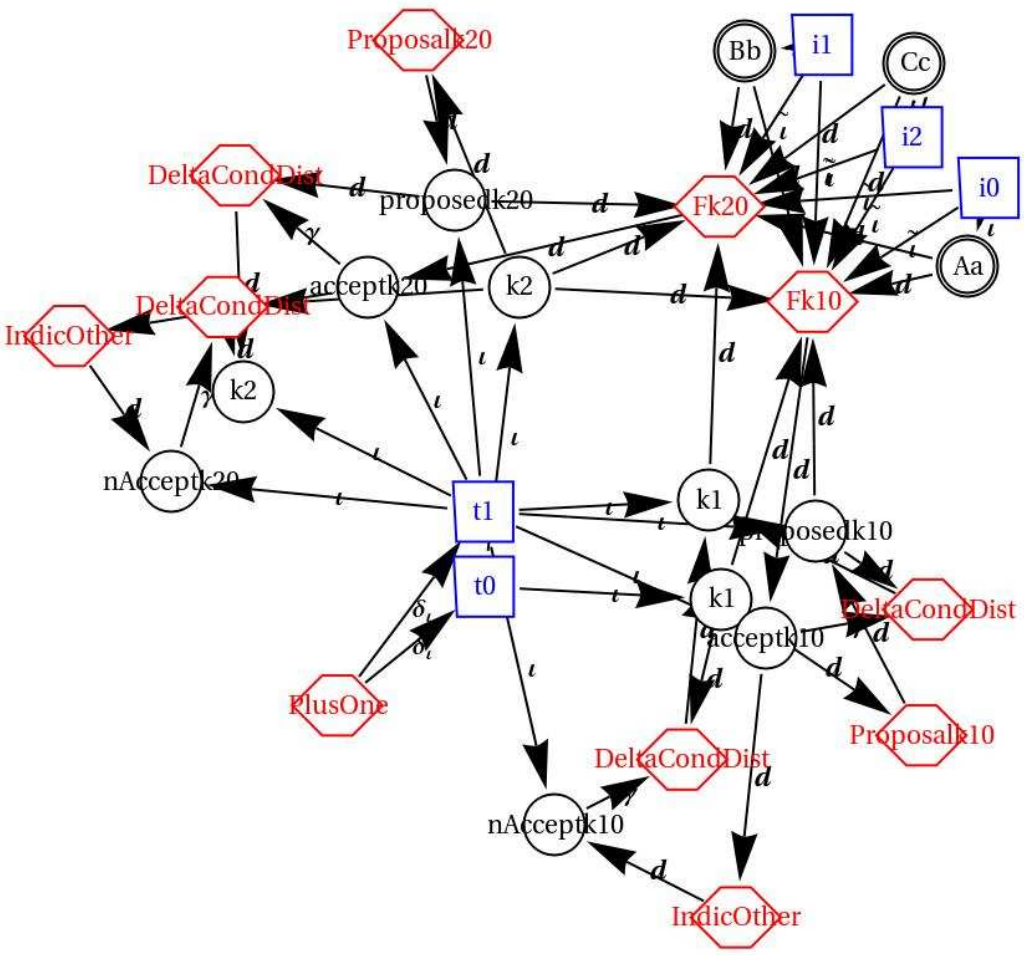


Figure 4.5: Dependency Diagram representation of the inference algorithm for parameter inference problem.

averaging 400 sampled rates after a burn-in period of 200 samples. Annealing at a fast pace of 33% drop in temperature every 20 iterations, the sample converged within only 300 iterations to point estimates with errors of 5% and 20% for k1 and k2. Converting the autogenerated sampling code into an annealing process required about half a dozen lines of code, and the process could easily be automated. Figure 4.4b shows convergence towards the true values 0.7 for k1 and .05 for k2.

4.5 Geometric Graph Prior for Variable-Structure Systems

Many problems of unknown or variable graphical structure arise in bioinformatics, computational biology, vision, and machine learning. A common component of such models is the expectation of a sparse and spatially embeddable graph. Therefore a model for unknown graph structure with a bias towards graphs than can be embedded naturally in a low-dimensional space is a potentially significant example for any modeling framework.

Protein interaction network statistics, including graphlet counts, can be fit well by such models [66]. Also a class of developmental biology models such as [35] use weak-spring variable-structure mechanical models embedded in real space. Bayesian inference of detailed regulatory networks has employed graph priors [75]. For machine learning, the "equilibrium graphical automata" [8] generalization of the following model could be adapted to provide priors on the structure of graphical models including Dependency Diagrams.

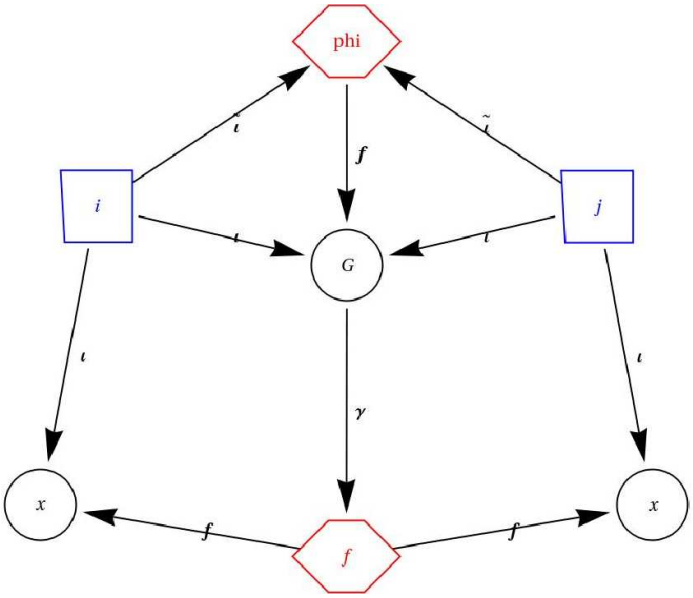
A prior probability distribution favoring such graphs is given by the Boltzmann distribution on adjacency matrices $G_{ij} \in \{0, 1\}$ with energy function

$$E_{\mathrm{GP}}(G, x) = C_1 \sum_{ij} G_{ij}(\|\mathbf{x}_i - \mathbf{x}_j\| - d)^2 + \psi_{\mathrm{CMP}}(\sum_{ij} G_{ij}|\lambda, v). \tag{4.2}$$

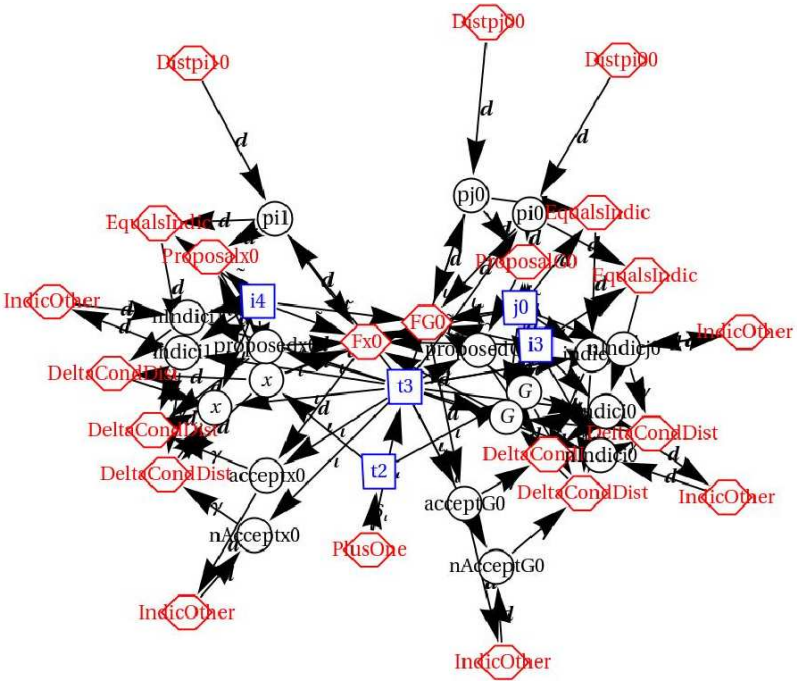
The first term of Equation 4.2 provides a spatial embedding prior in which the preferred distance between neighbors is d. The potential e is gated by G. A wide variety of other parameter-agreement potentials e could be used, leading toward the more gen-

eral "equilibrium graph automaton" class of models $E_{\mathrm{EGA}}(G, x)$ [8]. For example, if the parameter x_i included one discrete-valued component representing node type and if e rewarded type compatibility, then E_{GP} could provide a sparsity prior on Dependency Diagrams. The second term of Equation 4.2 is $\psi_{\mathrm{CMP}}(n|\lambda, v) = -n\lambda + v\log(n!)$ and corresponds to the Conway-Maxwell-Poisson (CMP) distribution [70, 36]. This distribution is defined as $P_{\mathrm{CMP}}(n|\lambda, v) = \lambda^n / \left[(n!)^\nu Z_{\mathrm{CMP}}(\lambda, \nu)\right]$, and here is applied to the total number of connections in G. CMP is in the exponential family or log-linear class of distributions, and generalizes the Poisson distribution to allow for under- or over-dispersion relative to the Poisson distribution. Here the standard deviation $\sigma = \sqrt{\lambda}$, and the Poisson is a special case of the CMP where $\nu = 1$. First described in 1962, the CMP distribution has recently been revived as a way to model count data in situations where the Poisson and Poisson-gamma distributions are not sufficiently flexible [27, 46]. When the distance term is omitted (by setting $C_1 = 0$), the CMP distribution on connection number generalizes the large-size limit of the standard Erdos-Renyi distribution on graphs, which in that limit is Poisson in the number of graph links. CMP can also be applied node-wise to in-degrees and out-degrees, specializing the exponential random graphs of [59]. The interaction between connection and interaction is also reminiscent of self-organizing maps. Additional entropy terms omitted here can roughly account for the degeneracy in mapping adjacency matrices to graphs.

The Dependency Diagram for Equation 4.2 is shown in Figure 4.6a, and that for the automatically transformed diagram for an MCMC sampling algorithm is shown in Figure 4.6b. We ran this model to sample the statistics of the graphs that result. We controlled the parameters λ and ν, varying λ in the range $[5, 50]$, in increments of 5 and ν in the range $[1, 2]$ in steps of size one-tenth. The number of nodes was 10. The quantities measured were the average number of edges observed (over 5500 samples each from 10 runs of 6000 samples, the first 500 of which were discarded for



(a) Model



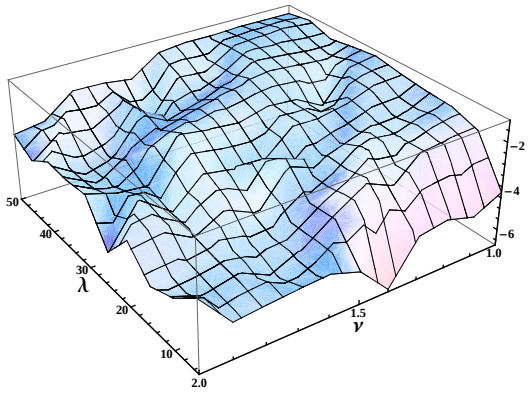
(b) Sampling algorithm

Figure 4.6: Dependency Diagrams for the graph prior experiment.

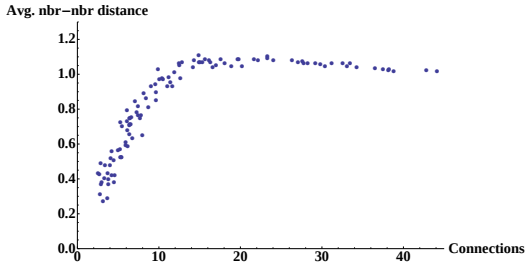
burn in), and the average distance between two connected nodes. Results are shown in Figure 4.7. The average number of edges ($\langle N_{\text{edges}} \rangle$) increases with λ and decreases with ν as expected. Figure 4.7c shows the relationship between the expected value of the CMP distribution and its parameters over the range considered. Figure 4.7d shows the observed average number of edges plotted against the parameters of the CMP distribution. This posterior average swells in the middle because of the combinatorial effect of the number of different ways of achieving a given total number of 1's in a 10x10 matrix, and is pulled down everywhere by the effect of the energy penalty incurred by each pair of neighbors.

The average distance ($\langle |\text{nbr} - \text{nbr}| \rangle$) between connected nodes is closest to the target of 1.0 in regions with low ν and high λ. This suggests the possibility of a relationship between the observed average distance between neighbors and the observed average number of edges present. Figure 4.7b shows this relationship quite clearly. This figure raises interesting questions: why does the system prefer closer neighbors for sparser graphs, in spite of the energy penalty associated with neighbors closer than one unit apart; why does the system never average greater than one unit separation for connected nodes, since the penalty is symmetric for distances above and below the target? We have not explored these questions in detail, but we believe they demonstrate the power of a Dependency Diagram as simple as Figure 4.6a to describe a complex and scientifically interesting system.

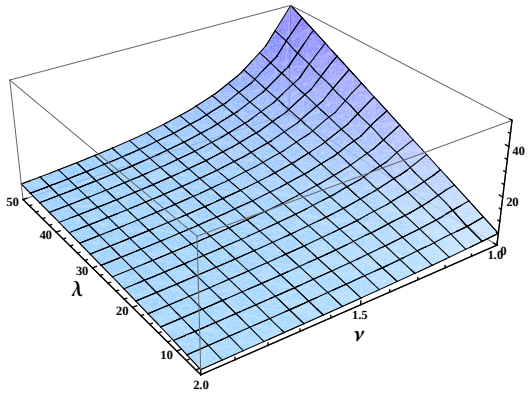
For the geometric graph prior example, the essential new DD link types were both types of indexing: ι and $\hat{\iota}$. These enabled the creation of a truly variable-structure system. Gating links are used where possible, but are always optional, since they can be eliminated in favor of f links. They are preferred over f links because they literally represent important, as-yet-unexploited opportunities for automated application of improved algorithms (e.g. softmax MFT updates, or use of sparse data structures).



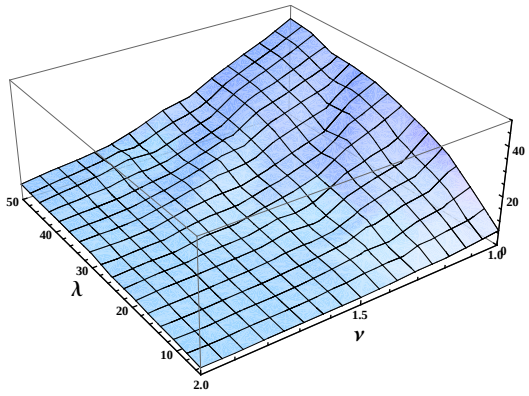
(a) Z-scores for observed graph connectivity distributions



(b) $\langle N_{\text{edges}} \rangle$ vs. $\langle |\text{nbr} - \text{nbr}| \rangle$



(c) Expected value of CMP distribution for pairs λ and ν



(d) Observed average connectivity

Figure 4.7: Results of graph prior experiment

The primary limitation of this model is that the system is bounded in size. This limitation could be removed by implementing inequality constraints between index nodes and random variable nodes, so that an index could have a variable upper bound, but we have not done this yet. The semantics involved require that such heterogeneous constraint links in the DD be excluded from certain graph cycles.

4.6 General-purpose sampling for statistical models in biochemistry

The Graph-Constrained Correlation Dynamics (GCCD) system described in Chapters 5 through 7 is built on top of Dependency Diagrams. As a tool designed with domain modeling in mind, GCCD development required a graphical modeling package that would allow simple specification of and automatic sampling and inference on arbitrary Markov random fields. To support GCCD, several features were added to Dependency Diagrams, including junction tree finding [34] and expectation calculation via belief propagation [62] for the subset of DDs which represent Markov random fields over variables of finite dimension.

4.7 Conclusion

Dependency Diagrams are a new type of graphical model, one which expands on the capabilities of existing models, allowing graphical models to better express variable-structure systems, to express iterative algorithms for sampling and inference, and to support graph-level transformations from models to such algorithms. We have defined the sematics of node and link types for indexing, gating, and constraints including those required for iteration. We defined a graph transformation that maps models to MCMC algorithms.

We have also implemented these data types, the transformation of model diagrams to algorithm diagrams, and the translation of certain diagrams to autogenerated code using a computer algebra system, and have demonstrated the implementation on a variety of small examples. The following chapters will build on the same DD

toolkit and software to demonstrate the applicability of Dependency Diagrams as the inference engine on which further scientific research can be built.

These examples exhibit not only the power and accuracy of the automated inference framework built into the Dependency Diagram system, but also the flexibility of Dependency Diagrams to encode numerous types of probabilistic models. Because the semantics extend those of chain and factor graphs, Dependency Diagrams are not restricted to indexed and variable-structured generalizations of Bayesian networks or Markov random fields; the examples presented here were encoded using a combination of directed and undirected models, and were handled by the same automated inference system.

Chapter 5

Graph-Constrained Correlation Dynamics: Motivation and Background

5.1 Introduction

Graph-Constrained Correlation Dynamics (GCCD) provides a framework with which to describe the time-evolution of a probability distribution by combining a Markov random field (MRF) and a set of ordinary differential equations (ODEs), given some understanding of this evolution in terms of either time series data or a reaction network model. The MRF is a parametrized model capable of describing the desired probability distribution at an instant in time. The ODEs describe a deterministic time-evolution of the parameters of the MRF. Together, these elements produce the desired time-evolution of a distribution. By learning a set of optimal functions for evolving the parameters of a Markov random field in time, the distribution described

by the MRF will match the target distribution as closely as possible at each time point. The possibility of fitting a GCCD model to a description of the same system in terms a reaction network arises via minimization of the Kullback-Leibler divergence between the true distribution and a parametrized distribution, and hinges on appeal to the chemical master equation for the evolution of probability under a reaction network [74]. This process is explored in detail in Chapter 6. Fitting a GCCD model to data, as described in Chapter 7, is a matter of approximating derivatives numerically and function fitting.

In either case, GCCD gives a modeler the ability to exploit domain knowledge, both when specifying the structure of the MRF and when choosing basis functions. The choice of structure for the MRF determines which variables and relationships are modeled over time, and the choice of basis functions determines which statistics will drive changes in the distribution.

5.2 Continuous-time Markov Processes and Reaction Networks

Many domains, particularly in biology and chemistry, are concerned with model systems in a particular subclass of continuous-time Markov processes defined on discrete state spaces that describe a number of objects of each of a set of types or species. Any continuous time, probabilistic system for which it is possible to describe the tendency of the system to move between any two or more states with a fixed set of transition rates is such a system.

One formalism for thinking about such systems is in terms of a *reaction network* [32]. A reaction network consists of a finite set S of species and a finite set R of reactions.

We will write a state of the system $\vec{n} = \{n_i\}$, which represents the number n_i of each species $s_i \in S$. In the event that we refer to distinct states i and j, they will be differentiated with superscripts, as $\{n_l^i\}$ and $\{n_l^j\}$. A reaction $r_i \in R$ consists of a pair of sets $S_{\text{in},i}$ and $S_{\text{out},i}$, where each member of S is labeled with a number of molecules or other discrete objects of that species which go into or come out of the reaction, and a *reaction rate*. We will write input stoichiometries n_{ij} for reaction j and species i, and output stoichiometries $\hat{n}_{ij}$. Rates we write k_j, so that we can specify a reaction with

$$\{n_{ij}s_i\} \xrightarrow{k_j} \{\hat{n}_{ij}s_i\}.$$

A reaction destroys, and thus requires, the relevant number n_{ij} corresponding to each species in $S_{\text{in},i}$, and creates the relevant number $\hat{n}_{ij}$ for each species in $S_{\text{out},i}$.The reaction rate is a nonnegative number which encodes the speed of the reaction.

There are two principal methods of modeling the time-evolution of such systems: deterministic evolution of the expected concentrations of each species, and stochastic simulation of one or more possible sequences of reaction events in accordance with the network [25]. By repeating stochastic simulations with the same initial conditions and different random seeds, it is possible to calculate statistics of interest beyond expected values of species members at a given time. Graph-Constrained Correlation Dynamics bridges these approaches by deterministically simulating an approximate probability distribution over states. This is accomplished by using a reduced state space (the space of parametrizations of an MRF) to approximate the probability density function over $\{n_i\}$.

The dynamics induced on the probability distribution over states by a reaction network take a well-known form, written succinctly in terms of a dot product. For a

state space of discrete-valued variables with probability distribution $p(\vec{n})$ and time-independent dynamics described by a fixed transition rate matrix W, the master equation [74] describes the time-evolution of p with the differential equation

$$\frac{dp}{dt} = W \cdot p. \tag{5.1}$$

Following [54], the matrix W can be calculated by summing over contributions from each reaction, with reaction r_i contributing a matrix with entries for the rate to transition between states j and k,

$$W^i_{jk} = \prod_{s_l \in S} \frac{n^k_l!}{n^k_l - n_{li}} \delta\left(n^j_l - n^k_l - n_{li} + \hat{n}_{li}\right),$$

where $\delta(x)$ is a Dirac delta function and so 1 if x is 0, and 0 otherwise. Given these per-reaction matrices W^i, the system is summarized by the reaction rate matrix $W = \sum_{r_i} W^i - \text{diag}(\vec{1} \cdot W^i)$, where $\text{diag}(v)$ creates a matrix with v along the diagonal and zero elsewhere. In general, a complete solution to the master equation for non-trivial networks is considered intractable [25].

Each of the two standard approaches to reaction network modeling have potential pitfalls. Stochastic simulation may require prohibitively many simulations, each with many simulation steps, in order to guarantee statistics of interest are computed with a desired degree of precision. This is particularly so if reaction rates differ by orders of magnitude and the "slow" processes are of interest [2]. For complex networks, deterministic simulation may require numerical solution of prohibitively large systems of equations, each potentially stiff [15]. Additionally, the deterministic simulation of functions other than mean concentration is itself an area of research [16]. Much work in recent years has focused on methods of speeding up stochastic simulation [53, 2, 13, 26], or simplifying deterministic differential equations [15, 16], in order to

address these problems. GCCD contributes to this literature by addressing the largely overlooked concern of continuously approximating the distribution $p(\vec{n})$.

For a system with a nontrivial number of states, explicit representation of the matrix W, and even the vector representation of $p(\vec{n})$, will be prohibitively large. Graph-Constrained Correlation Dynamics instead represents $p(\vec{n})$ as a parametrized distribution, which facilitates moving the representation of dynamics from this dot product to a more compact dynamics on the relevant parameters.

In particular, a GCCD model consists of two parts: a Markov random field to describe the instantaneous distribution of probability, and a set of differential equations to describe the flow of this probability. When creating a GCCD model, the first task is to specify the MRF. The organizing principle behind this task is the idea that the model should contain terms for meaningful interactions among variables in $\{\vec{n}\}$, subject to the constraint that sparsity and fewer numbers of terms each make learning and inference easier. In particular, any variables without an interaction term will be conditionally independent of one another in the resulting instantaneous probability distributions; in any case when this is not a reasonable assumption, it is preferable to add an interaction term. Section 5.3 describes these structures and concerns.

The second task in creating a GCCD model is to specify the set of differential equations for the parameters of the MRF. In general, we view this to be a learning problem, and seek methods to find ODEs which "best fit" the model under some set of criteria. Chapters 6 and 7 describe two such methods of fitting. The substrate for this fitting will be a set of *basis functions*, from which the ODEs will be built, as described in Section 5.4.

5.3 Markov random fields

The Markov random field (MRF) is a convenient formalism for the specification of a probability distribution. As described in Section 2.2, an MRF is used to describe a distribution in terms of undirected interactions between variables. Given a set X of random variables, an MRF consists of a graph G, such that each node of the graph is labeled with a variable $x \in X$, along with a set of non-negative *potential functions* ϕ. Nodes and the variables with-which they are labeled may be referred to interchangeably. Each clique c of the graph, which includes a subset X_c of the nodes, is associated with a potential function ϕ_c, such that the MRF describes the unnormalized weight distribution:

$$q(X) = \prod_c \phi_c(X_c).$$

This distribution of weights is normalized by the *partition function* $Z = \sum_X q(X)$, so that the probability $p(X) = q(X)/Z$. The most important conditional independence properties of such distributions are that any node is conditionally independent of the rest of graph, given the set of nodes to which it is connected directly. This set is called its *Markov blanket*[62]. The decision to base the GCCD approximation of p on the Markov random field allows the modeler the flexibility to represent precisely the quantities and functions of interest.

We will focus particularly on log-linear MRFs, which restrict their functions to be of the form $\phi_c(X_c) = e^{-\mu_c V_c(X_c)}$. Note taht V_c need not be a linear function of the modeled variables X_c, but such models are called log-linear because the log probability is linear in the tunable parameters μ. The log probability, which up to an arbitrary additive constant is $\sum_C \mu_c V_c(X_c)$, is sometimes called the negative log probability of the system. Each μ_c is a parameter that controls the weight that function V_c contributes

to the distribution. Because the functions V_c are fixed, it is these parameters which determine the structure of the probability distribution for a given MRF. Restricting our focus to this subset of MRF models allows us to make certain claims about the parameters μ_c, such as $\partial \log Z / \partial \mu_c = -\langle V_c \rangle$, where the notation $\langle \cdot \rangle$ denotes an expected value. As this particular relationship will be used repeatedly throughout the derivation to follow, we demonstrate it here. The calculation is a straight-forward application of the chain rule.

$$\begin{aligned}
\frac{\partial \log Z}{\partial \mu_c} &= \frac{\partial}{\partial \mu_c} \log \int \prod_\beta e^{-\mu_\beta V_\beta(x)} \mathrm{d}x \\
&= \frac{1}{\int \prod_\beta e^{-\mu_\beta V_\beta(x)} \mathrm{d}x} \frac{\partial}{\partial \mu_c} \int \prod_\beta e^{-\mu_\beta V_\beta(x)} \mathrm{d}x \\
&= \frac{1}{Z} \int \prod_\beta e^{-\mu_\beta V_\beta(x)} \left(-V_c(x)\right) \mathrm{d}x \\
&= -\int \tilde{p}(x) V_c(x) \mathrm{d}x \\
&= -\langle V_c \rangle. \qquad (5.2)
\end{aligned}$$

The functions V_c are sometimes called *feature functions*, as they are meant to compute meaningful features of the variables in the related cliques. Here they will be most often called *statistics* of the state, as they represent the *natural statistics* of these distributions viewed as exponential family distributions.

Additionally, we will draw MRFs as Dependency Diagrams (or, equivalently on the subset of DD notion we will use, factor graphs), as we feel that the visual representation is slightly more clear with the functions represented explicitly, rather than implicitly defined over cliques of variable nodes. This means that we will draw an MRF as a bipartite graph, where the clique potentials are explicitly represented as nodes in the diagram; see Section 3.2 for more on the relationship between these

notations. However, since we will not use any of the additional properties of DDs or factor graphs, and because our claims will not be general to all models of those types, we will still refer to the models as Markov random fields.

5.4 MRFs with Continuous-Time Dynamics

Graph-Constrained Correlation Dynamics leverages the MRF formalism to provide a way of describing probability distributions which evolve continuously in time.

As noted above, the probability distribution described by a given Markov random field is a function solely of the parameters μ. Therefore, while the MRF cannot represent continuous-time dynamics, a continuously evolving probability distribution can be specified with a traditional MRF and a continuously evolving set of parameters μ. GCCD accomplishes this by specifying differential equations for the parameters μ of the MRF. This model will be limited for tractability, in that all conditional independence relationships will be fixed across time by the MRF's factor graph. This is the "graph-constrained" part of Graph-Constrained Correlation Dynamics. This restriction comes in exchange for the flexibility to track the time evolution of any statistics of interest, which improves upon known techniques for tracking the time-evolution of the expected values of concentrations of each species in a reaction network.

In order to allow a GCCD model to be learned, the GCCD differential equations are defined to be linear combinations of sets of basis functions. That is, given a set $\mathcal{B}_\alpha$ of basis functions $f_\alpha(\mu)$, the dynamics for MRF parameter μ_α is assumed a linear combination of the form

$$\frac{d}{dt}\mu_\alpha = f_\alpha(\mu|\theta) \equiv \sum_A \theta_A f_{\alpha A}(\mu) \tag{5.3}$$

The remaining work of modeling required for GCCD is the selection of these basis functions. Sections 6.3 and 6.4 derive optimal bases for two small problems, and consider general forms that can be extrapolated from these.

Chapter 6

GCCD: Derivation

This first approach to Graph-Constrained Correlation Dynamics derives an analytically optimal solution in terms of reaction rates, using the dot product formulation for the dynamics represented by the master equation.

Let $\{x\}$ be a set of discrete variables, over which exists a target or "true" distribution be $p(\vec{x})$. Let the approximating distribution, specified by the MRF, be $\tilde{p}(\vec{x})$. We seek to minimize the time-averaged Kullback-Leibler (KL) divergence between these two distributions. To do this, we first take the derivative of the divergence first with respect to the MRF parameters μ, and then with respect to t. By setting the result equal to zero and solving for the free parameters in the dynamics, we derive an optimal approximation, given the MRF and basis functions at hand.

6.1 Derivation of Optimal Approximation

We begin with the definition of KL divergence, $\mathcal{D}_{\mathcal{KL}}(\tilde{p}\|p) = -\tilde{p}\log(p/\tilde{p})$ (or equivalently, $\tilde{p}\log(\tilde{p}/p)$). We could equally well begin with the divergence in the other

direction, as $-p\log(\tilde{p}/p)$; in fact, we have done this derivation, and tried the resulting algorithm. We focus on the stated direction because it seems to provide slightly more reliable results, but see Appendx A for the other derivation, and Section 6.5 for a comparison of the two methods.

Our approach will be to compute the derivative $\partial\mathcal{D}_{\mathcal{KL}}/\partial\mu_\alpha$, then take the time-derivative of this term, and minimize that. Minimization of $\partial\mathcal{D}_{\mathcal{KL}}/\partial\mu_\alpha$ corresponds to matching the distribution $\tilde{p}$ to p at an initial time point, and we will need to take for granted the ability to do this well once, as an initialization. Then, if we have done a good job of that, keeping the change in this term 0 – setting the derivative of this term equal to zero and solving for the parameters of a GCCD model – will keep the solution optimal as the two distributions change in time.

Before beginning the derivation, note that though we have so far defined X to be a discrete space, we use the integral notation throughout this derivation, as integration specializes to summation on a discrete domain, but the converse is not true.

The first few steps here are just pulling apart terms, including splitting the log of a ratio, and manipulating minus signs, starting from the definitions of the MRF, as

follows:

$$\begin{aligned}
\mathcal{D}_{\mathcal{KL}}(\mu(t)) &= -\int \tilde{p}(x;\mu(t)) \log\left(\frac{p(x;t)}{\tilde{p}(x;\mu(t))}\right) \mathrm{d}x \\
&= -\left(\int \tilde{p}(x;\mu(t)) \log(p(x;t))\mathrm{d}x - \int \tilde{p}(x) \log\left(\tilde{p}(x;\mu(t))\right) \mathrm{d}x\right) \\
&= -\int \tilde{p}(x;\mu(t)) \log\left(p(x;t)\right) \mathrm{d}x + \int \tilde{p}(x) \log\left(\tilde{p}(x;\mu(t))\right) \mathrm{d}x \\
&= -\int \tilde{p}(x;\mu(t)) \log\left(p(x;t)\right) \mathrm{d}x \\
&\quad + \int \tilde{p}(x;\mu(t)) \log\left(\frac{e^{\left(-\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x)\right)}}{Z(\mu(t))}\right) \mathrm{d}x \\
&= -\int \tilde{p}(x;\mu(t)) \log\left(p(x;t)\right) \mathrm{d}x \\
&\quad + \int \tilde{p}(x;\mu(t)) \log\left(e^{\left(-\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x)\right)}\right) \mathrm{d}x \\
&\quad - \int \tilde{p}(x;\mu(t)) \log\left(Z(\mu(t))\right) \mathrm{d}x \\
&= -\int \tilde{p}(x;\mu(t)) \log\left(p(x;t)\right) \mathrm{d}x \\
&\quad - \int \tilde{p}(x;\mu(t)) \sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) \mathrm{d}x \\
&\quad - \int \tilde{p}(x;\mu(t)) \log\left(Z(\mu(t))\right) \mathrm{d}x.
\end{aligned}$$

Finally, gathering terms

$$\mathcal{D}_{\mathcal{KL}}(\mu(t)) = -\int \tilde{p}(x;\mu(t)) \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right)\right) \mathrm{d}x.$$

We are now prepared to evaluate the derivative $\partial\left[\mathcal{D}_{\mathcal{KL}}(\mu(t))\right]/\partial\mu_\alpha$. Beginning with

the product rule, we have

$$\frac{\partial \mathcal{D}_{\mathcal{KL}}(\mu(t))}{\partial \mu_\alpha} = -\int \left(\frac{\partial}{\partial \mu_\alpha}\tilde{p}(x;\mu(t))\right) \tag{6.1}$$

$$\times \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right)\right) \mathrm{d}x$$

$$-\int \tilde{p}(x;\mu(t)) \tag{6.2}$$

$$\times \frac{\partial}{\partial \mu_\alpha}\left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right)\right) \mathrm{d}x. \tag{6.3}$$

Breaking this expression down, we begin by evaluating $\frac{\partial}{\partial \mu_\alpha}\tilde{p}(x;\mu(t))$.

$$\begin{aligned}
\frac{\partial}{\partial \mu_\alpha}\tilde{p}(x;\mu(t)) &= \frac{\partial}{\partial \mu_\alpha}\frac{\prod_\beta e^{-\mu_\beta V_\beta(x)}}{Z(\mu(t))} \\
&= \left(\frac{\partial}{\partial \mu_\alpha}\frac{1}{Z(\mu(t))}\right)\prod_\beta e^{-\mu_\beta V_\beta(x)} + \left(\frac{\partial}{\partial \mu_\alpha}\prod_\beta e^{-\mu_\beta V_\beta(x)}\right)\frac{1}{Z(\mu(t))} \\
&= \frac{-\frac{\partial}{\partial \mu_\alpha}Z(\mu(t))}{Z(\mu(t))^2}\prod_\beta e^{-\mu_\beta V_\beta(x)} - \left(V_\alpha(x)\prod_\beta e^{-\mu_\beta V_\beta(x)}\right)\frac{1}{Z(\mu(t))} \\
&- \frac{\partial \log(Z(\mu(t)))}{\partial \mu_\alpha}\tilde{p}(x) - V_\alpha(x)\tilde{p}(x) \\
&= \tilde{p}(x)\left(\langle V_\alpha\rangle - V_\alpha(x)\right).
\end{aligned} \tag{6.4}$$

Unless otherwise noted, all expectations are with respect to the distribution $\tilde{p}$. The last step follows from Equation (5.2).

Plugging this result back into Equation (6.3), we have

$$\begin{aligned}\frac{\partial \mathcal{D}_{\mathcal{KL}}(\mu(t))}{\partial \mu_\alpha} = &- \int \tilde{p}(x;\mu(t)) \left(\langle V_\alpha \rangle - V_\alpha(x)\right) \\ &\times \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right) \right) \mathrm{d}x \\ &- \int \tilde{p}(x;\mu(t)) \\ &\times \frac{\partial}{\partial \mu_\alpha} \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right) \right) \mathrm{d}x.\end{aligned}$$

Because μ doesn't appear in the target distribution p, $\frac{\partial \log(p(x;t))}{\partial \mu_\alpha} = 0$, and we have

$$\begin{aligned}\frac{\partial \mathcal{D}_{\mathcal{KL}}(\mu(t))}{\partial \mu_\alpha} = &- \int \tilde{p}(x;\mu(t)) \left(\langle V_\alpha \rangle - V_\alpha(x)\right) \\ &\times \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right) \right) \mathrm{d}x \\ &- \int \tilde{p}(x;\mu(t)) \\ &\times \frac{\partial}{\partial \mu_\alpha} \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) \right) \mathrm{d}x. \qquad (6.5)\end{aligned}$$

From Equation (5.2), $\partial \log\left(Z(\mu(t))\right)/\partial \mu_\alpha$ is $-\langle V_\alpha \rangle$. Fitting this into Equation (6.5) leaves

$$\begin{aligned}\frac{\partial \mathcal{D}_{\mathcal{KL}}(\mu(t))}{\partial \mu_\alpha} = &- \int \tilde{p}(x;\mu(t)) \left(\langle V_\alpha \rangle - V_\alpha(x)\right) \\ &\times \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) + \log\left(Z(\mu(t))\right) + \log\left(p(x;t)\right) \right) \mathrm{d}x \\ &- \int \tilde{p}(x;\mu(t)) \\ &\times \frac{\partial}{\partial \mu_\alpha} \left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) \left(V_\beta(x) - \langle V_\alpha \rangle\right) \right) \mathrm{d}x. \qquad (6.6)\end{aligned}$$

If we separate the second integral, we see that the two halves negate one-another

$$\begin{aligned}\int \tilde{p}(x;\mu(t))\left(V_\alpha(x)-\langle V_\alpha\rangle\right)\mathrm{d}x &= \int \tilde{p}(x;\mu(t))V_\alpha(x)\mathrm{d}x - \int \tilde{p}(x;\mu(t))\langle V_\alpha\rangle\mathrm{d}x \\ &= \int \tilde{p}(x;\mu(t))V_\alpha(x)\mathrm{d}x - \langle V_\alpha\rangle\int \tilde{p}(x;\mu(t))\mathrm{d}x \\ &= \langle V_\alpha\rangle - \langle V_\alpha\rangle = 0.\end{aligned}$$

We can now absorb the leading minus sign, leaving

$$\begin{aligned}\frac{\partial\mathcal{D}_{\mathcal{KL}}(\mu(t))}{\partial\mu_\alpha} = \int \tilde{p}(x;\mu(t))\left(V_\alpha(x)-\langle V_\alpha\rangle\right) \\ \times\left(\sum_{\beta=1}^{\text{cliques}}\mu_\beta(t)V_\beta(x)+\log\left(Z(\mu(t))\right)+\log\left(p(x;t)\right)\right)\mathrm{d}x.\end{aligned} \tag{6.7}$$

Now we wish to find the derivative of Equation (6.7) with respect to time. In order to accomplish this, we first name pieces of Equation (6.7). This will allow us to explain how we're going to proceed, and to perform sub-computations on these named pieces. Let $A = \tilde{p}(x;\mu(t))\left(V_\alpha(x)-\langle V_\alpha\rangle\right)$, $B = \sum_{\beta=1}^{\text{cliques}}\mu_\beta(t)V_\beta(x)$, $C = \log Z(\mu(t))$, $D = \log p(x;t)$. Then we can write the desired derivative as

$$\frac{d}{dt}\frac{\partial\mathcal{D}_{\mathcal{KL}}(\mu(t))}{\mu_\alpha} = \int\left[A\frac{\partial}{\partial t}B + A\frac{\partial}{\partial t}C + A\frac{\partial}{\partial t}D + (B+C+D)\frac{\partial}{\partial t}A\right]\mathrm{d}x. \tag{6.8}$$

Concentrating on the first two of these terms, $\int A\,\partial/\partial t\;B + A\,\partial/\partial t\;C\mathrm{d}x$, we see $\partial/\partial t\;B = \sum_\beta \partial\mu_\beta/\partial t\;V_\beta$ and, using the chain rule, $\partial/\partial t\;C = \sum_\beta \partial\mu_\beta/\partial t\;\partial\log Z/\partial\mu_\beta$. Then, once again using Equation (5.2),

$$\begin{aligned}\int A\frac{\partial}{\partial t}B + A\frac{\partial}{\partial t}C\mathrm{d}x &= \int A\left[\sum_\beta\frac{\partial\mu_\beta}{\partial t}\left(V_\beta-\langle V_\beta\rangle\right)\right]\mathrm{d}x \\ &= \int\tilde{p}(x;\mu(t))\sum_\beta\frac{\partial\mu_\beta}{\partial t}\left(V_\beta-\langle V_\beta\rangle\right)\left(V_\alpha-\langle V_\alpha\rangle\right)\mathrm{d}x.\end{aligned}$$

Shifting the integral inside the sum, we recognize the expression for the covariance between functions V_α and V_β, where $\langle (x - \langle x \rangle)(y - \langle y \rangle) \rangle = \mathrm{Cov}(x, y) = \langle xy \rangle - \langle x \rangle \langle y \rangle$, so

$$\begin{aligned} \int A \frac{\partial}{\partial t} B + A \frac{\partial}{\partial t} C \mathrm{d}x &= \sum_{\beta=1}^{\text{cliques}} \frac{\partial \mu_\beta(t)}{\partial t} \mathrm{Cov}(V_\alpha, V_\beta) \\ &= \sum_{\beta=1}^{\text{cliques}} \frac{\partial \mu_\beta(t)}{\partial t} \big(\langle V_\beta V_\alpha \rangle - \langle V_\alpha \rangle \langle V_\beta \rangle \big). \end{aligned} \tag{6.9}$$

Turning our attention now to the fourth term of Equation (6.8) is: $\int (B + C + D)\, \partial A / \partial t \, \mathrm{d}x$.

$$\begin{aligned} \int \left(\frac{\partial}{\partial t} \left((\tilde{p}(x; \mu(t)) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) \right) \right) \\ \times \left(\left(\sum_{\beta=1}^{\text{cliques}} \mu_\beta(t) V_\beta(x) \right) + \log \left(Z(\mu(t)) \right) + \log \left(p(x; t) \right) \right) \mathrm{d}x. \end{aligned}$$

Applying to product and chain rules to Equation (6.4), the time derivative of A, where $\partial / \partial t \; A = \partial \left[\tilde{p}(x; \mu(t)) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) \right] / \partial t$, is

$$\begin{aligned} \frac{\partial}{\partial t} A &= \left(\frac{\partial}{\partial t} \tilde{p}(x; \mu(t)) \right) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) + \tilde{p}(x; \mu(t)) \frac{\partial}{\partial t} \left(V_\alpha(x) - \langle V_\alpha \rangle \right) \\ &= \sum_\beta \left[\frac{\partial \mu_\beta}{\partial t} \tilde{p}(x) \left(\langle V_\beta \rangle - V_\beta(x) \right) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) \right] - \tilde{p}(x; \mu(t)) \frac{\partial}{\partial t} \langle V_\alpha \rangle \\ &= -\tilde{p}(x; \mu(t)) \sum_\beta \frac{\partial \mu_\beta}{\partial t} \left(V_\beta(x) - \langle V_\beta \rangle \right) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) \\ &\quad + \tilde{p}(x; \mu(t)) \sum_\beta \frac{\partial \mu_\beta}{\partial t} \big(\langle V_\alpha V_\beta \rangle - \langle V_\alpha \rangle \langle V_\beta \rangle \big) \end{aligned} \tag{6.10}$$

Note that the covariance between V_α and V_β has appeared again, and that it is being subtracted from terms which have the same form as the terms inside a covariance.

That is, integrating over the first part of Equation (6.10) would again produce the covariance between V_α and V_β, times the derivative of μ.

Terms such as $V_\beta(x) - \langle V_\beta \rangle$ recur regularly throughout this derivation. Therefore, we define a new notation. Let $\Delta x \equiv x - \langle x \rangle$. Then we can rewrite the covariance in Equation (6.9) as

$$\sum_{\beta=1}^{\text{cliques}} \frac{\partial \mu_\beta(t)}{\partial t} \big(\langle V_\beta V_\alpha \rangle - \langle V_\alpha \rangle \langle V_\beta \rangle \big) = \sum_{\beta=1}^{\text{cliques}} \frac{\partial \mu_\beta(t)}{\partial t} \langle \Delta \mathrm{V}_\alpha \Delta \mathrm{V}_\beta \rangle.$$

Additionally, Equation (6.10) fits the same pattern, with $x = (V_\beta(x) - \langle V_\beta \rangle)(V_\alpha(x) - \langle V_\alpha \rangle) = (\Delta V_\alpha \Delta V_\beta)$. So, we may rewrite this as

$$\frac{\partial}{\partial t} A = -\tilde{p}(x; \mu(t)) \sum_\beta \frac{\partial \mu_\beta}{\partial t} \Delta(\Delta V_\alpha \Delta V_\beta).$$

Plugging this in and copying the definitions for B, C, and D, we have

$$\begin{aligned} \int (B + C + D) \frac{\partial}{\partial t} A \mathrm{d}x = &- \int \tilde{p}(x; \mu(t)) \sum_\beta \frac{\partial \mu_\beta}{\partial t} \Delta(\Delta V_\alpha \Delta V_\beta) \left(\sum_{\gamma=1}^{\text{cliques}} \mu_\gamma(t) V_\gamma(x) \right) \mathrm{d}x \\ &- \int \tilde{p}(x; \mu(t)) \sum_\beta \frac{\partial \mu_\beta}{\partial t} \Delta(\Delta V_\alpha \Delta V_\beta) \log Z(\mu(t)) \mathrm{d}x \\ &- \int \tilde{p}(x; \mu(t)) \sum_\beta \frac{\partial \mu_\beta}{\partial t} \Delta(\Delta V_\alpha \Delta V_\beta) \log p(x; t) \mathrm{d}x. \end{aligned}$$

In the second of these terms, the factor $\log Z(\mu(t))$ is not a function of x. As a result, when the integral is evaluated, the resulting expected values all cancel. So this term

is zero, leaving

$$\int (B+C+D)\frac{\partial}{\partial t}A\mathrm{d}x = -\int \tilde{p}(x;\mu(t))\sum_{\beta}\frac{\partial \mu_\beta}{\partial t}\Delta(\Delta V_\alpha \Delta V_\beta)\left(\sum_{\gamma=1}^{\text{cliques}}\mu_\gamma(t)V_\gamma(x)\right)\mathrm{d}x \\ -\int \tilde{p}(x;\mu(t))\sum_{\beta}\frac{\partial \mu_\beta}{\partial t}\Delta(\Delta V_\alpha \Delta V_\beta)\log p(x;t)\mathrm{d}x. \tag{6.11}$$

Once again, we break apart an equation and consider the parts individually. Beginning with the first integral, we regroup the terms of the multiplication so that there is just one double-sum over cliques, then move terms which do no depend on x outside of the integral, and integrate. This leaves another expected value.

$$-\int \tilde{p}(x;\mu(t))\sum_{\beta}\frac{\partial \mu_\beta}{\partial t}\Delta(\Delta V_\alpha \Delta V_\beta)\left(\sum_{\gamma=1}^{\text{cliques}}\mu_\gamma(t)V_\gamma(x)\right)\mathrm{d}x \\ = -\sum_{\beta=1}^{\text{cliques}}\sum_{\gamma=1}^{\text{cliques}}\mu_\gamma(t)\frac{\partial \mu_\beta(t)}{\partial t}\langle V_\gamma \Delta\left(\Delta V_\alpha \Delta V_\beta\right)\rangle. \tag{6.12}$$

This expression belies some of the symmetry of this term. Note that, if $x = V_\gamma$ and $y = (\Delta V_\alpha \Delta V_\beta)$, the inner most term is $\langle x\Delta y\rangle$. From the definitions of Δ and expectation, this is equivalent to $\langle xy\rangle - \langle x\rangle\langle y\rangle$, which once again is $\mathrm{Cov}(x,y)$. Of course, covariance relationships are symmetric, so this is also $\mathrm{Cov}(y,x)$, which from the preceding argument is $\langle y\Delta x\rangle$. Thus,

$$\langle x\Delta y\rangle = \mathrm{Cov}(x,y) = \langle y\Delta x\rangle. \tag{6.13}$$

Applying this to Equation (6.12), we have finally

$$-\sum_{\beta=1}^{\text{cliques}}\sum_{\gamma=1}^{\text{cliques}}\mu_\gamma(t)\frac{\partial\mu_\beta(t)}{\partial t}\langle V_\gamma\Delta\left(\Delta V_\alpha\Delta V_\beta\right)\rangle=-\sum_{\beta=1}^{\text{cliques}}\sum_{\gamma=1}^{\text{cliques}}\mu_\gamma(t)\frac{\partial\mu_\beta(t)}{\partial t}\langle\Delta V_\alpha\Delta V_\beta\Delta V_\gamma\rangle. \tag{6.14}$$

Turning to the second half of Equation (6.11), the steps are similar, but the V_γ terms are replaced with $\log p(x)$ terms:

$$-\int\tilde{p}(x;\mu(t))\sum_\beta\frac{\partial\mu_\beta}{\partial t}\Delta(\Delta V_\alpha\Delta V_\beta)\log p(x;t)\mathrm{d}x=-\sum_{\beta=1}^{\text{cliques}}\frac{\partial\mu_\beta(t)}{\partial t}\langle\Delta V_\alpha\Delta V_\beta\Delta\log p(x;t)\rangle \tag{6.15}$$

The final piece of Equation (6.8) is

$$\begin{aligned}\int A\frac{\partial}{\partial t}D\mathrm{d}x&=\int\tilde{p}(x;\mu(t))\left(V_\alpha(x)-\langle V_\alpha\rangle\right)\frac{\partial}{\partial t}\log p(x;t)\mathrm{d}x\\&=\langle\Delta V_\alpha\frac{\partial}{\partial t}\log p(x;t)\rangle.\end{aligned} \tag{6.16}$$

Now that we have analyzed each of the parts of Equation (6.8), we can set it equal to zero and move the terms with a sum over cliques across the equals sign. Then we have as a solution

$$\begin{aligned}\langle\Delta V_\alpha\frac{\partial}{\partial t}\log\left(p(x;t)\right)\rangle=\sum_{\beta=1}^{\text{cliques}}\frac{\partial\mu_\beta(t)}{\partial t}\Bigg(&\langle\Delta V_\alpha\Delta V_\beta\Delta\log\left(p(x;t)\right)\rangle-\langle\Delta V_\alpha\Delta V_\beta\rangle\\&+\sum_{\gamma=1}^{\text{cliques}}\mu_\gamma(t)\langle\Delta\mathrm{V}_\alpha\Delta V_\beta\Delta V_\gamma\rangle\Bigg).\end{aligned} \tag{6.17}$$

Following the master equation for the derivative of p, as in Equation (5.1), and setting

$\partial\mu_\beta(t)/\partial t \equiv f_\beta(\mu(t))$ as in Equation (5.3), this is

$$\langle \Delta V_\alpha \frac{(W \cdot p(;t))(x)}{p(x;t)} \rangle = \sum_{\beta=1}^{\text{cliques}} f_\beta(\mu(t)) \left(\langle \Delta V_\alpha \Delta V_\beta \Delta \log p(x;t) \rangle - \langle \Delta V_\alpha \Delta V_\beta \rangle + \sum_{\gamma=1}^{\text{cliques}} \mu_\gamma(t) \langle \Delta V_\alpha \Delta V_\beta \Delta V_\gamma \rangle \right).$$

Then, defining a linear form for f_β, again as in Equation (5.3), we have

$$\langle \Delta V_\alpha \frac{(W \cdot p(;t))(x)}{p(x;t)} \rangle = \sum_{\beta=1}^{\text{cliques}} \sum_{A=1}^{\text{bases}} f_{\beta,A}(\mu(t)) \theta_A \left(\langle \Delta V_\alpha \Delta V_\beta \Delta \log p(x;t) \rangle - \langle \Delta V_\alpha \Delta V_\beta \rangle + \sum_{\gamma=1}^{\text{cliques}} \mu_\gamma(t) \langle \Delta V_\alpha \Delta V_\beta \Delta V_\gamma \rangle \right).$$

Now we define a vector $\mathbf{B}$ with α entries

$$\mathbf{B}_\alpha = \langle \Delta V_\alpha \frac{(W \cdot p(;t))(x)}{p(x;t)} \rangle, \tag{6.18}$$

and an A-by-α matrix $\mathbf{A}$ with entries

$$\mathbf{A}_{\alpha,A} = \sum_{\beta=1}^{\text{cliques}} f_{\beta,A}(\mu(t)) \left(\langle \Delta V_\alpha \Delta V_\beta \Delta \log p(x;t) \rangle - \langle \Delta V_\alpha \Delta V_\beta \rangle + \sum_{\gamma=1}^{\text{cliques}} \mu_\gamma(t) \langle \Delta V_\alpha \Delta V_\beta \Delta V_\gamma \rangle \right). \tag{6.19}$$

Then $\mathbf{B} = \mathbf{A}\theta$, and by calculating values for $\mathbf{A}$ and $\mathbf{B}$ we can solve for optimal θ.

We refer to this algorithm as *method two*. Method one derives from $\mathcal{D}_{\mathcal{KL}}(p\|\tilde{p})$; this derivation is included in Appendix A. Method one also results in a system of linear equations, but this system contains fewer terms than are present in Equations (6.18) and (6.19). The term in method one which corresponds to Equation 6.18 is not

weighted by the ratio $\tilde{p}/p$, and the terms in method one which correspond to Equation (6.19) include only covariances between pairs of potentials, omitting the terms with $\log p$ and the product of three potentials. This means that an implementation of method one may be faster than an implementation of method two, as the matrix computation stage of the two algorithms will include $O(n^2)$ operations in the case of method one, and $O(n^3)$ operations in the case of method two, where n is the number of potentials in the diagram. Section 6.5 presents a more detailed comparison of implementations of the two methods.

It will generally be true that this system of equations is under-determined, as the dimensions of the matrix $\mathbf{A}$ are $n \times m$, where n is the number of potentials in the MRF and m is the total number of bases – if the same set of bases is used for each potential, and that set has size $\hat{m}$, then $m = n\hat{m}$ – and vector $\mathbf{B}$ has length n. However, notice also that we haven't made any significant assumptions about p or $\tilde{p}$, beyond the definition of the relationship between $\tilde{p}$ and μ.

Therefore, we will build larger $\mathbf{A}$ and $\mathbf{B}$ matrices by *stacking* together in a block fashion several copies of Equations (6.23) and (6.24) together which have been computed using different distributions p and $\tilde{p}$, characterized by different values of μ. As long as these copies are linearly independent this will produce a fully constrained system of equations.

6.2 Implementation

We have created a full-featured implementation of Graph-Constrained Correlation Dynamics in the *Mathematica* programming language. This implementation leverages our Dependency Diagram software for functions related to Markov random fields, and

therefore we call the software *GCCDdd*. The source code, in the form of a *Mathematica* package and a set of notebooks defining example models, is available from `http://computableplant.ics.uci.edu/sw/gccd/`.

The specification of a GCCDdd model consists of a Dependency Diagram, a set of variable definitions, and a set of basis functions. The user must also specify the statistic functions of the MRF, as described in Section 5.3, but GCCDdd defines the potentials in terms of these statistics and μ automatically, with μ_i corresponding to the i^{th} function node in the diagram specification. That is, if the user inputs a diagram with a node labeled ϕ_x, and this is the first function node, GCCDdd will clear any existing definition for ϕ_x and define

ϕ_x`[args__] := `μ_1`*stat[`ϕ_x`][args]`

where two underscores, as `args__` is idiomatic *Mathematica* for an arbitrary set, so that this expression simply passes any arguments through to `stat[phi[x]]`. It is only left to the user to supply a definition of `stat[phi[x]]` which computes the statistic of interest on the variables linked to ϕ_x in the diagram. An optional argument may specify functions whose definitions should not be cleared and (re)defined in this way, but such functions are treated as fixed in time and no dynamics are be learned for them.

The parameter learning routine requires as further input a set of corresponding pairs of p and μ. The Dependency Diagram engine for generating sampling code is used to create a sampler in terms of unknown μ, which is reused repeatedly for different values of the parameters. Using this, GCCDdd approximates expected values with respect to $\tilde{p}$ via Metropolis sampling. $W \cdot p$ is also approximated using the same collection of samples, by evaluating $(W \cdot p)(x) = \sum_{y \in \text{samples}} W_{x,y} p(y)$ for $x \in$ samples. Since $(W \cdot p)(x)$ appears in an expected value with respect to $\tilde{p}$, and this is calculated

with this set of samples, it would not be beneficial to compute this value for states outside of the set of samples.

The process of computing the multiple copies of $\mathbf{A}$ and $\mathbf{B}$ needed for the stacking procedure described in Section 6.1 is embarrassingly parallel, and the implemented software uses the parallelization framework of *Mathematica* to allow the user the option of computing these terms in parallel. For situations when it is computationally intensive to prepare many matched pairs of p and μ, we have also implemented a custom scheduling program which makes maximally efficient use of all available processors to alternately compute these pairs and then solve for $\mathbf{A}$ and $\mathbf{B}$.

In addition to solving for θ by computing $\mathbf{A}$ and $\mathbf{B}$, the implemented package also includes the ability to perform belief propagation on, and to optimize the parameters of, discrete valued Markov random fields, as described in Section 7.3.

6.3 Example: A Single Binding Site

Consider a reaction network with two possible states and two unary reactions to move between them. This could represent a single binding site which is filled and emptied, or a single species which comes into existence and dies, or a switch which flips on and off. We will think of it as a single binding site that admits a molecule of species A, and say that the system is in state A or *on* if such a molecule is bound, and in state $\emptyset$ or *off* otherwise. The tendency for a molecule to bind we call k_{on}, and the tendency to unbind we call k_{off}.

6.3.1 Models

Drawn as a set of reaction arrows, such a system would be

$$\emptyset \underset{k_{\text{off}}}{\overset{k_{\text{on}}}{\rightleftharpoons}} A$$

This system is well-understood [54], but provides a tractable problem for explaining and testing GCCD.

The instantaneous possibility of the presence of the binding molecule can be viewed as a Bernoulli random variable. Call this variable $a \in [0, 1]$, with 0 representing unbound ($\emptyset$) and 1 representing bound (A). Then, using the exponential family formation of the Bernoulli distribution, a has probability

$$p(a, t) = \frac{e^{-\mu(t)}}{1 + e^{-\mu(t)}}. \tag{6.20}$$

This corresponds to an MRF with just the variable a and one potential $\phi(a) = \mu V(a)$ where V is the identity function, $V(a) = a$.

Now that we have an MRF formulation of the instantaneous probability of the system, all that is required is a set of basis functions. In general, this will be a modeling challenge. Here, however, we can derive optimal bases.

6.3.2 Derivation of Optimal Bases

Setting aside the MRF formulation momentarily, consider the standard one-parameter formulation of the Bernoulli distribution, $p(a, t) = p$. When viewed this way, it is

easy to derive (see for example [54]) from the master equation that

$$\frac{dp}{dt} = k_{on} - (k_{on} + k_{off})p. \tag{6.21}$$

This is all we'll need to know in order to derive the set of bases.

Since we seek a form of this derivative in terms of μ, we first solve Equation (6.20) for $\mu(t)$. We find that

$$\mu(t) = \log \frac{1 - p(a,t)}{p(a,t)}. \tag{6.22}$$

Then, leaving aside for the moment unnecessary references to p and μ as functions of a and t,

$$\begin{aligned}
\frac{\partial \mu}{\partial t} &= \frac{p}{1-p} \frac{\partial}{\partial t} \frac{1-p}{p} \\
&= \frac{p}{1-p} \left(-\frac{(1-p)\frac{\partial p}{\partial t}}{p^2} - \frac{\frac{\partial p}{\partial t}}{p} \right) \\
&= \frac{\partial p / \partial t}{(-1+p)p}.
\end{aligned}$$

Replacing with the derivative of p from Equation (6.21)

$$= \frac{k_{\text{on}} - (k_{\text{on}} + k_{\text{off}})p}{(-1+p)p}.$$

Copying in the definition of p in terms of μ

$$= \frac{e^{\mu}\left(1 + e^{-\mu}\right)\left(k_{\text{on}} - \frac{e^{-\mu}(k_{\text{on}} + k_{\text{off}})}{1+e^{-\mu}}\right)}{-1 + \frac{e^{-\mu}}{1+e^{-\mu}}}.$$

Simplifying

$$= -k_{\text{on}} - e^{\mu} k_{\text{on}} + k_{\text{off}} + e^{-\mu} k_{\text{off}}.$$

Now observe that this can be written as the linear combination

$$(1, e^{-\mu}, e^{\mu}) \cdot (k_{\text{off}} - k_{\text{on}}, k_{\text{off}}, -k_{\text{on}}).$$

This provides an ideal set of bases for this problem, along with the correct values of θ.

6.3.3 Checking the GCCD Equations Algebraically

Now it is possible to work through the calculations involved in Equations (6.18) and (6.19) for this example network, which provides a perfect opportunity to check the algorithm.

To begin with, we write the initial probability of both states in terms of the probability that the site is on: $p = \begin{pmatrix} p_{on} \\ 1 - p_{on} \end{pmatrix}$. Several terms in both equations rely on the value of $\langle V \rangle$, the expected value under the approximating distribution. Here, this is precisely the approximate probability $\tilde{p}(a = 1)$, which we will abbreviate $\tilde{p}$. The rate matrix $W = \begin{pmatrix} -k_{on} & k_{off} \\ k_{on} & -k_{off} \end{pmatrix}$, so

$$W \cdot p = \begin{pmatrix} -k_{on}\,(1 - p_{on}) + k_{off} p_{on} \\ k_{on}\,(1 - p_{on}) - k_{off} p_{on} \end{pmatrix}.$$

Another term from Equation (6.19) is $\langle \log p \rangle$, which is $\tilde{p} \log p_{on} + (1 - \tilde{p}) \log(1 - p_{on})$.

Given these pieces and a little algebra, we find that

$$\begin{aligned} \mathbf{B} = \Bigg(& \frac{(1-\tilde{p})\,\tilde{p}\,(k_{\text{on}}\,(1-p_{\text{on}}) - k_{\text{off}} p_{\text{on}})}{p_{\text{on}}} \\ & - \frac{(1-\tilde{p})\,\tilde{p}\,(-k_{\text{on}}\,(1-p_{\text{on}}) + k_{\text{off}} p_{\text{on}})}{1-p_{\text{on}}} \Bigg) \tilde{p}(1-\tilde{p}) \end{aligned} \tag{6.23}$$

and if we set

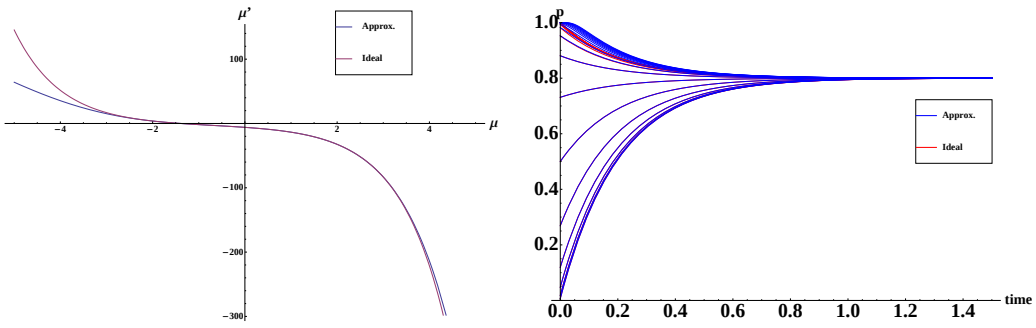
$$\begin{aligned} \mathbf{a} = \big(& -(1-\tilde{p})^2\,\tilde{p} - (1-\tilde{p})\,\tilde{p}^2 + (1-\tilde{p})\,\tilde{p}^2 \\ & \times (\log(1-p_{\text{on}}) - \log(1-p_{\text{on}})\,(1-\tilde{p}) - \log(p_{\text{on}})\,\tilde{p}) \\ & + (1-\tilde{p})^2\,\tilde{p}\,(\log(p_{\text{on}}) - \log(1-p_{\text{on}})\,(1-\tilde{p}) - \log(p_{\text{on}})\,\tilde{p}) \\ & - \log\left(\frac{\tilde{p}}{1-\tilde{p}}\right)\left((1-\tilde{p})^3\,\tilde{p} - (1-\tilde{p})\,\tilde{p}^3\right) \Bigg) \end{aligned}$$

then

$$\mathbf{A} = \mathbf{a} \begin{pmatrix} \tilde{p}(1-\tilde{p}) \\ \tilde{p}^2 \\ (1-\tilde{p})^2 \end{pmatrix}. \tag{6.24}$$

Notice that this produces one equation with three unknown values. As discussed in Section 6.1, this is to be expected. It will generally be true that the system of equations is under-determined. However, since $\tilde{p}$ is a function of μ and we know how to calculate μ to fit any p_{on} – this is simply Equation 6.22 – it is trivial to perform the stacking procedure described above.

When we build larger $\mathbf{A}$ and $\mathbf{B}$ matrices by stacking three copies of Equations (6.23) and (6.24) together for different values of p_{on} and $\tilde{p}$, we find that such a system can be solved to find the correct answer.



(a) $d\mu/dt$ (b) $p(t)$

Figure 6.1: GCCDdd Results When Optimal Bases Are Not Present

In general, it will not be true that we know p as an analytical function of μ or vice-versa, but it provides a foothold for the analytical solution here.

6.3.4 Testing the Implementation

The *Mathematica* implementation of GCCDdd requires that the user specify an MRF, in the form of a Dependency Diagram, evaluable functions for the statistics V of the MRF (where $\phi_i(x) = \mu_i V_i(x)$), basis functions, and matched pairs of p and μ.

We tested GCCD on this model by stacking six pairs of p and μ: $p = .5$, .33, .25, .2, and .8 and corresponding μ calculated as $\mu = -\log(p/1-p)$. Expectations were computed using 500 samples. In this regime, if the set of basis functions is the three listed above, the implementation always returns exactly the correct values for the three named bases. If the set of basis functions is expanded but includes the correct bases, the algorithm always finds some coefficients which produce exactly the correct values of $d\mu/dt$ as a function of μ, and typically will produce exactly the ideal coefficients (the three values above, and zero for all other coefficients) using the iterative pruning technique described later, in Section 6.4.4.

Even when the ideal bases are not present at all, the implementation typically finds a solution which produces perfect $d\mu/dt$ for μ in some range, such as μ corresponding to probabilities less than .95. Crucially, these solutions always show the correct stationary distribution, $p = k_{\text{on}}/k_{\text{off}} + k_{\text{on}}$. Figure 6.1a compares $d\mu/dt$ for an approximate solution and the ideal function. Here, $k_{\text{on}} = 4$ and $k_{\text{off}} = 1$, and the provided bases were $\{1, 2^{\mu_1}, \cos(\mu_1), \sin(\mu_1), \mu_1, \mu_1^2, \mu_1^3, 1/1 + \mu_1^2\}$. In this run, the returned coefficients produced

$$\frac{d\mu}{dt} = 15.7228 - 25.0512 \times 2^{\mu_1} + 3.39572\cos(\mu_1) - 1.82789\sin(\mu_1) + 14.1878\mu_1 + 6.15909\mu_1^2 + 0.264425\mu_1^3 - \frac{0.0673011}{1+\mu_1^2}.$$

Figure 6.1b compares the resulting time series of p for this approximation with the ideal answer, for all initial conditions between $\mu = -13$ and 1, in unit steps. As suggested by Figure 6.1a, the two are indistinguishable until μ between -3 and -4, which corresponds to p between 0.953 and 0.982. These figures are suggestive, but represent single runs. To quantify this, we repeated this particular experiment twenty-five times and calculated the KL divergence between the approximation and the true distribution at points between $t = 0$ and 1 at evenly spaced intervals of .05 on each run, from the worst-case starting distribution corresponding to $\mu = -13$. The results were so consistent that we were unable to draw error bars, with standard deviations on the order of 10^{-10}. The average divergence peaked at 0.047 at $t = 0.05$, and was less than 0.001 by $t = .35$. To put these numbers in context, true pairs of p and $\tilde{p}$ which produce KLD equal to 0.047 in this context include $(0.99, 0.95)$, $(0.9, 0.78)$, and $(0.7, 0.82)$. Pairs which diverge by 0.001 include $(0.99, 0.985)$, $(0.95, 0.94)$ and $(0.7, 0.72)$. This ability to find high quality approximations with suboptimal bases is crucial, because in general it may not be possible to derive optimal bases.

6.3.5 Extensions

The simplest extension of this model considers a single molecule moving up and down a chain of N states, representable as a set of uni-molecular reactions

$$A_i \underset{k_{r,i}}{\overset{k_{f,i}}{\rightleftharpoons}} A_{i+1}$$

for $1 \leq i < N$.

This system produces a multinomial distribution for the state of the molecule, which can be drawn as an MRF with a node for each of $N-1$ states, similar to the one above (where one node represented the two states "bound" and "not bound"), with no connections between them.

After performing a calculation analogous to the one above for $N = 2$ for N equal to three and four, we hypothesize that the general form for the optimal functions for parameter μ_i are $1 \times (k_{f,i} + k_{r,i-1} - k_{r,N-1})$ and $e^{-\mu_{N-1}} \times k_{f,N-1}$, plus if $i < N-1$ then $e^{\mu_i - \mu_{i+1}} \times -k_{r,i}$ and if $i = N-1$ then $e^{\mu_{N-1}} \times k_{r,N-1}$. GCCDdd produces this solution for N as large as we have checked, up to five.

6.4 Example: Cooperative Binding

As an extension of the above model, imagine now two binding sites whose states are related. In particular, suppose that binding is *cooperative*, in that the propensity for each site to transition from unbound to bound is greater when the other site is bound, and additionally that the propensity to transition from bound to unbound is lower when both states are bound.

6.4.1 Reaction Network Model

There are a number of ways you might represent this with a reaction network. For instance, you might simply count bound sites, such as $B(i)$ where $i \in (0, 1, 2)$. In the arrow notation of chemical networks, this model is

$$B(0) \underset{\rho_b}{\overset{\rho_f}{\rightleftharpoons}} B(1)$$

$$B(1) \underset{\rho_b c_b}{\overset{\rho_f c_f}{\rightleftharpoons}} B(2)$$

where c_b and c_f represent measures of cooperativity. For c_f, a value greater than one represents more cooperative, while a value less than one represents greater inhibition as it approaches zero. This relationship is reversed for c_b.

This network appears as a subset of a larger calmodulin/CaM-kinase pathway described in [63], and investigated using GCCD in greater detail in Chapter 7. The two binding sites in this small model represent sites where calcium binds to calmodulin. Each calmodulin has two ends, on each of which are two calcium binding sites, which interact cooperatively. In the larger CaMKII model, the calmodulin goes on to bind to a kinase, which in turn forms dimers and six-dimer holoenzymes as part of a synaptic pathway involved in the formation of memory [20, 6]. In the biological system, the binding rate is not changed by the presence of a first bound calcium, but unbinding is slowed when two are present. Nonetheless, the cooperative binding model is biologically relevant (see for example [55]), and is slightly more interesting.

GCCD will be used to describe the time-evolution of the probability distribution over this system. Figure 6.2 shows an approximate form of such a time-evolution. In this figure, four probabilities are tracked: the probability that neither site is bound (p_1), the probability that each site is bound alone (p_2 and p_3), and the probability

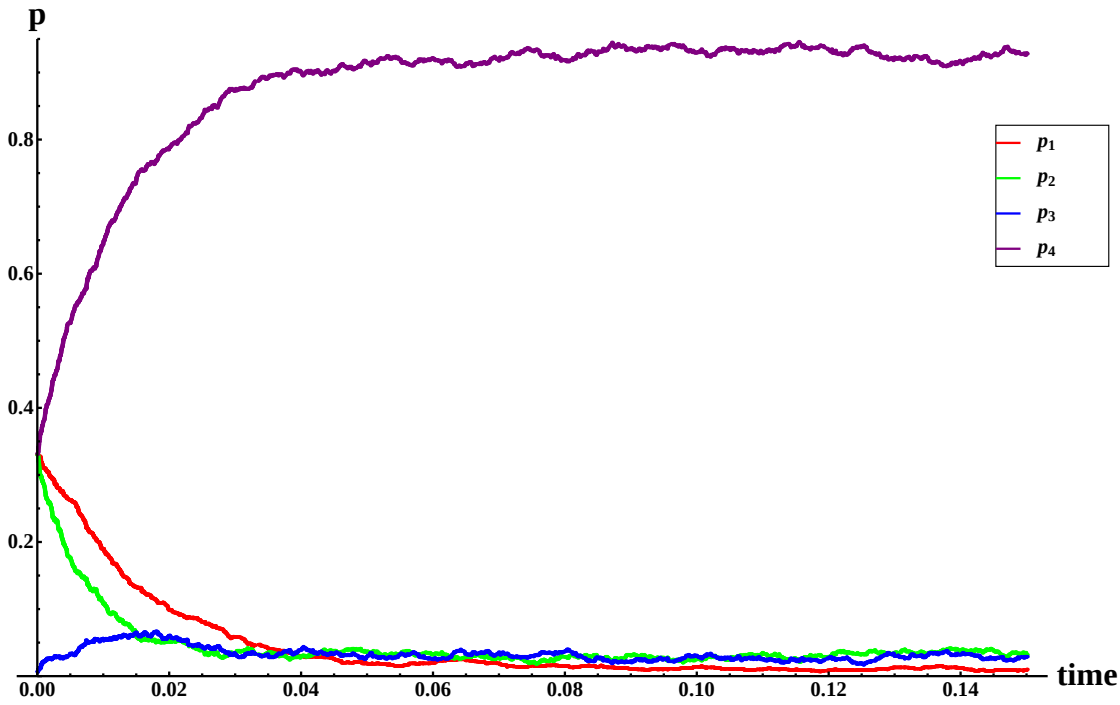


Figure 6.2: Time evolution of probabilities for cooperative binding, statistics calculated from results of 1000 stochastic simulations.

that both are bound (p_4). This figure was produced by simulating this system 1000 times and computing averages. GCCD will provide a set of equations which provide a continuous solution to this problem, and which can be started from any initial conditions without regenerating these 1000 simulations.

6.4.2 Markov random field Model

While the counting representation presented in the previous section allows for a compact reaction network, it is less convenient for the MRF representation. Instead, we will represent the model with two binary variables, each of which has a value of plus or minus one, in a manner similar to the variable in Section 6.3.1. In addition to the identity statistic for each of these, we also add an interaction statistic, which evaluates their product. This statistic will be positive one when the binding sites

Figure 6.3: Markov random field for Cooperative Binding.

agree, and negative one when they are in different states. This model is depicted in Figure 6.3.

If the two variables are b_0 and b_1, then the probability of this system is proportional to

$$e^{\mu_1 b_0 + \mu_2 b_0 b_1 + \mu_3 b_1}, \tag{6.25}$$

and we can affect its change in time by changing the values of μ.

Throughout this section, it will be necessary to write the probability distribution as a vector, and to refer to the four states with some shorthand. In such instances, we will refer to $p = \{p_1, p_2, p_3, p_4\}$, with the states ordered as follows:

$$\{(b_0 = -1, b_1 = -1), (-1, 1), (1, -1), (1, 1)\}$$

.

6.4.3 Derivation of f and θ

We begin the derivation of the optimal bases and coefficients by comparing two forms of the change in the probability distribution in time: the expression $W \cdot p$ from the master equation and the application of the chain rule to Equation 6.25.

We begin by writing down the propensity matrix W for the master equation:

$$\begin{pmatrix} -2\rho_f & \rho_b & \rho_b & 0 \\ \rho_f & -\rho_b - c_f\rho_f & 0 & c_b\rho_b \\ \rho_f & 0 & -\rho_b - c_f\rho_f & c_b\rho_b \\ 0 & c_f\rho_f & c_f\rho_f & -2c_b\rho_b \end{pmatrix}.$$

Then, by applying the dot product, we have

$$\frac{dp}{dt} = \begin{pmatrix} -2\rho_f p_1 + \rho_b(p_2 + p_3) \\ \rho_f\left(p_1 - c_f p_2\right) - \rho_b\left(p_2 + c_b(-1 + p_1 + p_2 + p_3)\right) \\ \rho_f\left(p_1 - c_f p_3\right) - \rho_b\left(p_3 + c_b(-1 + p_1 + p_2 + p_3)\right) \\ c_f\rho_f(p_2 + p_3) + 2c_b\rho_b(-1 + p_1 + p_2 + p_3) \end{pmatrix}. \tag{6.26}$$

Note that we have substituted $p_4 = \left(1 - \sum_{i=1}^{3} p_i\right)$.

Setting this aside, we write out the probability vector in terms of Equation 6.25, properly normalized,

$$p = \begin{pmatrix} \frac{e^{\mu_1 - \mu_2 + \mu_3}}{e^{-\mu_1-\mu_2-\mu_3} + e^{\mu_1+\mu_2-\mu_3} + e^{\mu_1-\mu_2+\mu_3} + e^{-\mu_1+\mu_2+\mu_3}} \\ \frac{e^{\mu_1 + \mu_2 - \mu_3}}{e^{-\mu_1-\mu_2-\mu_3} + e^{\mu_1+\mu_2-\mu_3} + e^{\mu_1-\mu_2+\mu_3} + e^{-\mu_1+\mu_2+\mu_3}} \\ \frac{e^{-\mu_1 + \mu_2 + \mu_3}}{e^{-\mu_1-\mu_2-\mu_3} + e^{\mu_1+\mu_2-\mu_3} + e^{\mu_1-\mu_2+\mu_3} + e^{-\mu_1+\mu_2+\mu_3}} \\ \frac{e^{-\mu_1 - \mu_2 - \mu_3}}{e^{-\mu_1-\mu_2-\mu_3} + e^{\mu_1+\mu_2-\mu_3} + e^{\mu_1-\mu_2+\mu_3} + e^{-\mu_1+\mu_2+\mu_3}} \end{pmatrix}. \tag{6.27}$$

As we evolve the probabilities in time, we will only explicitly track the change in the probabilities of three of the four states. Because we have formulated W so such that its columns sum to zero, the master equation conserves total probability [74], so this will be sufficient. If we differentiate $\{p_1, p_2, p_3\}$ with respect to t, and pull out a common factor of $w = \left(1 + e^{2(\mu_1+\mu_2)} + e^{2(\mu_1+\mu_3)} + e^{2(\mu_2+\mu_3)}\right)$, we have the following

vector in terms of $d\mu/dt$

$$\begin{pmatrix} \frac{dp_1}{dt} \\ \frac{dp_2}{dt} \\ \frac{dp_3}{dt} \end{pmatrix} = \frac{1}{w^2} \begin{pmatrix} 2e^{2(\mu_1+\mu_3)} \left(\left(1+e^{2(\mu_2+\mu_3)}\right) \frac{d\mu_1}{dt} - e^{2\mu_2} \left(e^{2\mu_1}+e^{2\mu_3}\right) \frac{d\mu_2}{dt} \right. \\ \left. + \left(1+e^{2(\mu_1+\mu_2)}\right) \frac{d\mu_3}{dt} \right) \\ 2e^{2(\mu_1+\mu_2)} \left(\left(1+e^{2(\mu_2+\mu_3)}\right) \frac{d\mu_1}{dt} + \left(1+e^{2(\mu_1+\mu_3)}\right) \frac{d\mu_2}{dt} \right. \\ \left. - e^{2\mu_3} \left(e^{2\mu_1}+e^{2\mu_2}\right) \frac{d\mu_3}{dt} \right) \\ -2e^{2(\mu_2+\mu_3)} \left(e^{2\mu_1} \left(e^{2\mu_2}+e^{2\mu_3}\right) \frac{d\mu_1}{dt} - \left(1+e^{2(\mu_1+\mu_3)}\right) \frac{d\mu_2}{dt} \right. \\ \left. - \left(1+e^{2(\mu_1+\mu_2)}\right) \frac{d\mu_3}{dt} \right) \end{pmatrix}$$

Solving this for $d\mu/dt$, then factoring out common terms $r = -\frac{1}{4}e^{-2\mu_1-2\mu_2-2\mu_3}$ and w gives

$$\frac{d\mu}{dt} = rw \begin{pmatrix} e^{2\mu_2} \left(1+e^{2(\mu_1+\mu_3)}\right) \frac{dp_1}{dt} + e^{2\mu_3} \left(1+e^{2(\mu_1+\mu_2)}\right) \frac{dp_2}{dt} \\ \quad + e^{2\mu_1} \left(-1+e^{2(\mu_2+\mu_3)}\right) \frac{dp_3}{dt} \\ e^{2\mu_2} \left(-1+e^{2(\mu_1+\mu_3)}\right) \frac{dp_1}{dt} + e^{2\mu_3} \left(1+e^{2(\mu_1+\mu_2)}\right) \frac{dp_2}{dt} \\ \quad + e^{2\mu_1} \left(1+e^{2(\mu_2+\mu_3)}\right) \frac{dp_3}{dt} \\ e^{2\mu_2} \left(1+e^{2(\mu_1+\mu_3)}\right) \frac{dp_1}{dt} + e^{2\mu_3} \left(-1+e^{2(\mu_1+\mu_2)}\right) \frac{dp_2}{dt} \\ \quad + e^{2\mu_1} \left(1+e^{2(\mu_2+\mu_3)}\right) \frac{dp_3}{dt} \end{pmatrix}.$$

Or, replacing the dp/dt terms as in Equation 6.26, and subsequent p terms as in

Equation 6.27

$$\frac{d\mu}{dt} = r \begin{pmatrix} -e^{2\mu_1+4\mu_2}\rho_b - e^{4\mu_2+2\mu_3}\rho_b + e^{2\mu_1}c_b\rho_b - e^{2\mu_3}c_b\rho_b \\ \quad - 2e^{2\mu_1+2\mu_2+2\mu_3}c_b\rho_b + e^{4\mu_1+2\mu_3}\rho_f + 2e^{2\mu_1+2\mu_2+2\mu_3}\rho_f \\ \quad - e^{2\mu_1+4\mu_3}\rho_f + e^{4\mu_1+4\mu_2+2\mu_3}c_f\rho_f + e^{2\mu_1+4\mu_2+4\mu_3}c_f\rho_f \\ e^{2\mu_1+4\mu_2}\rho_b + 2e^{2\mu_1+2\mu_2+2\mu_3}\rho_b + e^{4\mu_2+2\mu_3}\rho_b - e^{2\mu_1}c_b\rho_b \\ \quad - e^{2\mu_3}c_b\rho_b - 2e^{2\mu_1+2\mu_2+2\mu_3}c_b\rho_b - e^{4\mu_1+2\mu_3}\rho_f \\ \quad - 2e^{2\mu_1+2\mu_2+2\mu_3}\rho_f - e^{2\mu_1+4\mu_3}\rho_f + 2e^{2\mu_1+2\mu_2+2\mu_3}c_f\rho_f \\ \quad + e^{4\mu_1+4\mu_2+2\mu_3}c_f\rho_f + e^{2\mu_1+4\mu_2+4\mu_3}c_f\rho_f \\ -e^{2\mu_1+4\mu_2}\rho_b - e^{4\mu_2+2\mu_3}\rho_b - e^{2\mu_1}c_b\rho_b + e^{2\mu_3}c_b\rho_b \\ \quad - 2e^{2\mu_1+2\mu_2+2\mu_3}c_b\rho_b - e^{4\mu_1+2\mu_3}\rho_f + 2e^{2\mu_1+2\mu_2+2\mu_3}\rho_f \\ \quad + e^{2\mu_1+4\mu_3}\rho_f + e^{4\mu_1+4\mu_2+2\mu_3}c_f\rho_f + e^{2\mu_1+4\mu_2+4\mu_3}c_f\rho_f \end{pmatrix}.$$

This can be rewritten to make it clear that it is a linear combination of functions of μ. In particular, it picks out a constant and a set of eight bases of the form $e^{\pm 2(\mu_i \pm \mu_j)}$ for particular signs and combinations of i and j.

$$\frac{d\mu}{dt} = \frac{1}{4} \begin{pmatrix} 2(c_b\rho_b - \rho_f) + e^{2(-\mu_1+\mu_2)}\rho_b + e^{2(\mu_2-\mu_3)}\rho_b \\ \quad + e^{-2(\mu_1+\mu_2)}c_b\rho_b - e^{-2(\mu_2+\mu_3)}c_b\rho_b - e^{2(\mu_1-\mu_2)}\rho_f \\ \quad + e^{2(-\mu_2+\mu_3)}\rho_f - e^{2(\mu_1+\mu_2)}c_f\rho_f - e^{2(\mu_2+\mu_3)}c_f\rho_f \\ 2(c_b\rho_b - \rho_b + \rho_f - c_f\rho_f) - e^{2(-\mu_1+\mu_2)}\rho_b - e^{2(\mu_2-\mu_3)}\rho_b \\ \quad + e^{-2(\mu_1+\mu_2)}c_b\rho_b + e^{-2(\mu_2+\mu_3)}c_b\rho_b + e^{2(\mu_1-\mu_2)}\rho_f \\ \quad + e^{2(-\mu_2+\mu_3)}\rho_f - e^{2(\mu_1+\mu_2)}c_f\rho_f - e^{2(\mu_2+\mu_3)}c_f\rho_f \\ 2(c_b\rho_b - \rho_f) + e^{2(-\mu_1+\mu_2)}\rho_b + e^{2(\mu_2-\mu_3)}\rho_b \\ \quad - e^{-2(\mu_1+\mu_2)}c_b\rho_b + e^{-2(\mu_2+\mu_3)}c_b\rho_b + e^{2(\mu_1-\mu_2)}\rho_f \\ \quad - e^{2(-\mu_2+\mu_3)}\rho_f - e^{2(\mu_1+\mu_2)}c_f\rho_f - e^{2(\mu_2+\mu_3)}c_f\rho_f \end{pmatrix}. \tag{6.28}$$

6.4.4 GCCDdd Results

To test this set of equations, we applied the GCCDdd implementation to an instance of this model with the following parameter settings: $\rho_f = 32$, $\rho_b = 10$, $c_f = 5$, $c_b = .5$. Qualitatively this means that, cooperativity aside, binding is three times as likely as unbinding, while the companion site being bound makes binding five times as likely, and unbinding half as likely. The probability distribution will equilibrate in a state where binding is equally, and highly, likely for each site.

For the bias parameters μ_1 and μ_3, this means they should have steady state values which are negative. A negative coefficient times a positive variable, raised to a negative power, will contribute a net positive power of e to the energy. These parameters should also have the same steady state value. Similarly, the interaction parameter should, in the long-term, have a negative value.

Plugging the stated parameter values into Equation (6.28) gives these coefficients

$$\begin{pmatrix} -13.5 & 2.5 & -8. & 1.25 & -40. & 8. & 2.5 & -1.25 & -40. \\ -66.5 & -2.5 & 8. & 1.25 & -40. & 8. & -2.5 & 1.25 & -40. \\ -13.5 & 2.5 & 8. & -1.25 & -40. & -8. & 2.5 & 1.25 & -40. \end{pmatrix}. \tag{6.29}$$

We applied the *Mathematica* implementation of GCCDdd to this problem, using the stacking procedure outlined in Section 6.1. Section 6.5 investigates the number of instances it is necessary to stack, and how long it is necessary to run the MCMC sampling routine, in order to generate useful results. Here, we set aside those results and supplied the solver with 125 initial conditions, using all ways of setting each of the parameters μ_i to each value between negative and positive one in steps of one half. We also used 1500 samples for computing expectations. Additionally, we initialized the system with many more basis functions than the correct nine per MRF potential.

We allowed the algorithm to run iteratively, discarding at each iteration bases for which the coefficients fell below a particular thresh hold – in this case three-tenths. This procedure isn't strictly necessary, in that the numerical solution of the ODEs is typically not degraded without it, but it demonstrates the accuracy and flexibility of the method.

We have employed the following heuristic procedure for setting the thresh hold parameter. First, we apply GCCDdd once without the iterative discarding of bases. We then numerically solve the resulting ODEs and evaluate them qualitatively. We then set the thresh hold just high enough to discard the bases with the smallest coefficients. We repeat this procedure until the ODEs begin to change dramatically, or there are signs that the solution is no longer of high quality. The only justification for this procedure is that it is empirically effective at producing a minimal set of bases functions, and it can be skipped entirely without loss of quality in the solutions.

The coefficients produced by GCCDdd were

$$\begin{pmatrix} -13.53 & 2.49 & -7.95 & 1.25 & -40 & 8.01 & 2.51 & -1.27 & -39.99 \\ -66.46 & -2.50 & 8.01 & 1.24 & -40 & 8.02 & -2.51 & 1.26 & -39.99 \\ -13.52 & 2.51 & 8.00 & -1.24 & -39.99 & -7.99 & 2.50 & 1.26 & -39.99 \end{pmatrix}.$$

Figure 6.4 shows the evolution of μ produced by learning with GCCDdd. The chosen initial condition confers a preference for b_0 to be unbound, b_1 to be bound, and for the two sites to agree on their binding. This translates to a distribution with an equal probability close to one-third for each of the cases that both are bound, neither are bound, or just b_1 is bound, with a small probability reserved for the state where only b_0 is bound.

In addition to the fact that the long-term behavior is qualitatively correct, this image

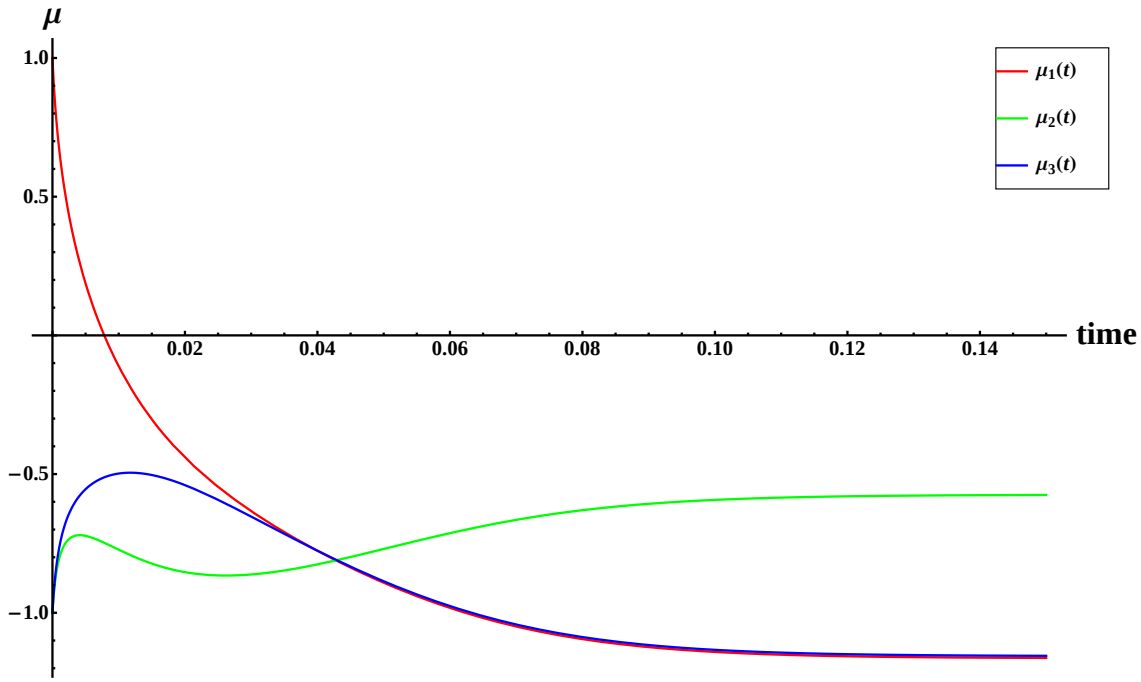


Figure 6.4: Time evolution of μ parameters for cooperative binding.

illustrates other interesting facts about the system. In particular, the behavior of the parameter μ_2 – the interaction parameter – tracks the time scale on which the system first moves toward distributions under which it is less important that the two sites agree than it is that at least one is bound, and then back toward states where it is important they agree, as they bootstrap one-another up, and finally toward the steady state, where they may occasionally disagree as one binding site briefly falls off before quickly jumping back on.

The time series of μ parameters in Figure 6.4 corresponds to a time-evolution of p shown in Figure 6.5. Again, this is qualitatively correct, in that, in the long term, the "all on" state is by far the most probable, the "all off" state is very unlikely, and the remaining states are indistinguishable. This can be compared to Figure 6.2, which shows statistics calculated after stochastically simulating 1000 such systems in the *Plenum* software package.

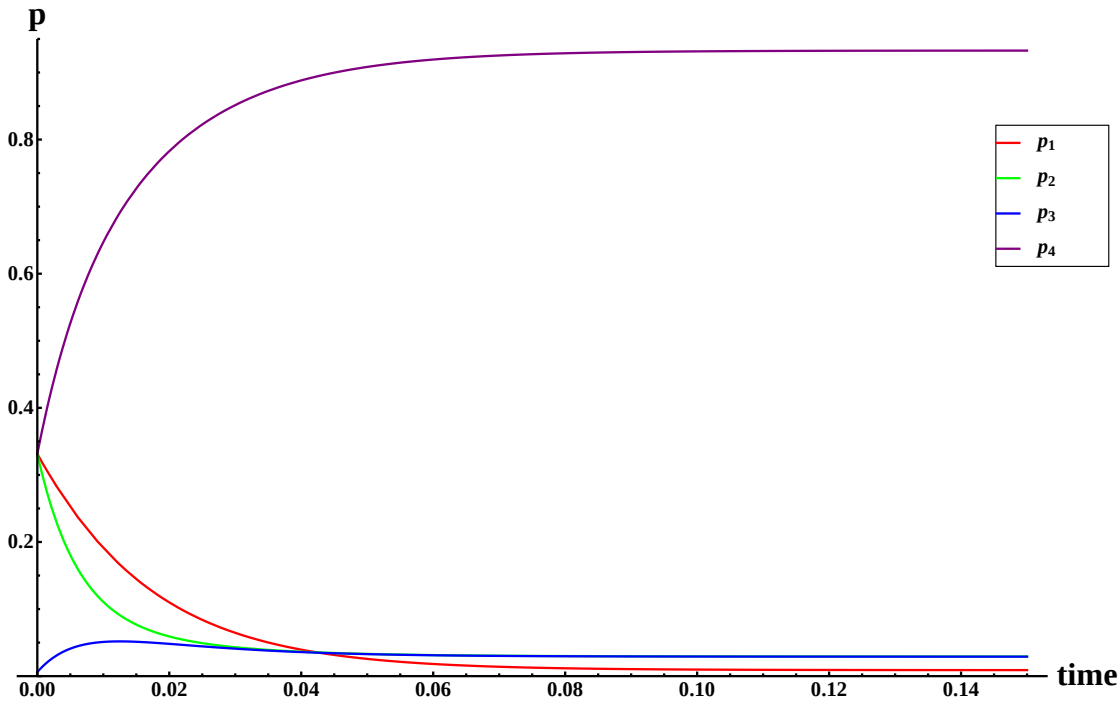


Figure 6.5: Time evolution of probabilities for cooperative binding, results of numerical solution to ODEs generated by GCCDdd.

6.5 GCCD Sensitivity Analysis

Because the cooperative binding problem of Section 6.4 is the most complex one which we have characterized completely, we have used it to analyze some properties of the GCCDdd implementation. Here, we consider the affect of several quantities and variables of interest on several metrics for the results of GCCD. In each of the following examples, we use the same model and rate parameters as Section 6.4.

The variables are: the number of samples used to approximate $\tilde{p}$; the quality of the match between p and $\tilde{p}$, in terms of ℓ_1-norm; the quality of the match between sample-approximated $\tilde{p}$ and the exact distribution corresponding to μ, in terms of ℓ_1-norm; the number of instances which are stacked together; and the method of GCCD (either the derivation in this chapter from $\mathcal{D}_{\mathcal{KL}}(\tilde{p})\|p)$, or the algorithm derived from the complementary divergence, $\mathcal{D}_{\mathcal{KL}}(p\|\tilde{p})$). For historical reasons, we refer to the

derivation in Section 6.1 as "method two." Method one is derived in Appendix A, and has solution

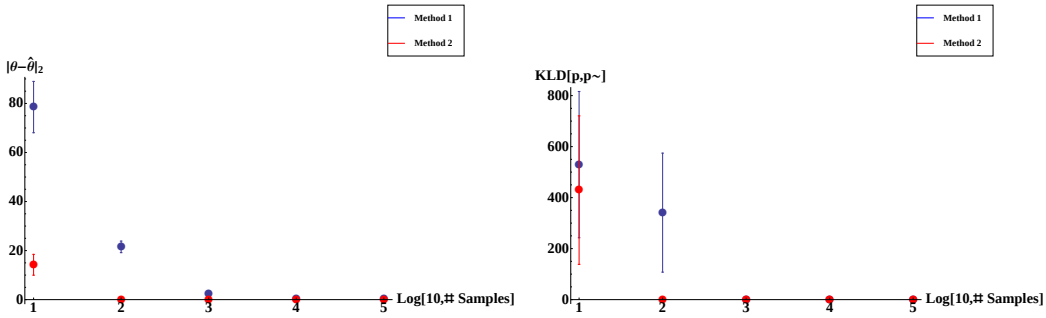
$$\int (W \cdot p(;t)(x))\Delta V_\alpha(x)\mathrm{d}x = -\sum_{\beta=1}^{\text{cliques}} f_{\beta,A}(\mu(t))\langle V_\alpha \Delta V_\beta\rangle. \tag{6.30}$$

The metrics are Euclidean distance of the calculated coefficient vector from the correct answer in Equation 6.29, referred to here as $\hat{\theta}$, and time-summed KL-divergence in each of the two directions. The divergences were summed over 100 time points evenly spaced between $t = 0$ and $t = .15$. In the event that the ODEs became stiff or could not be evaluated numerically before a time point, a large divergence of 40 was used, making the worst possible distance on this metric 4000.

In the following analysis, each data point represents the mean of sixteen samples with the appropriate settings, with error bars representing standard error. In the event that the point for method one is not visible, it is obscured by the point for method two, because the values are equal.

6.5.1 Sampling Concerns

The algorithms derived in Section 6.1 and Appendix A consist of equations in terms of expected values. The GCCDdd implementation of GCCD samples the relevant distributions using Markov-Chain Monte Carlo samplers, which in turn are generated by the Dependency Diagram software. For a technique which relies on approximation via sampling, it is natural to ask how samples are required to produce a result of desired quality? Figure 6.6 compares results on three metrics for both GCCDdd solutions, for increasingly many samples. Because, for a given number of samples generated by MCMC on a particular problem, the effective sample size depends to



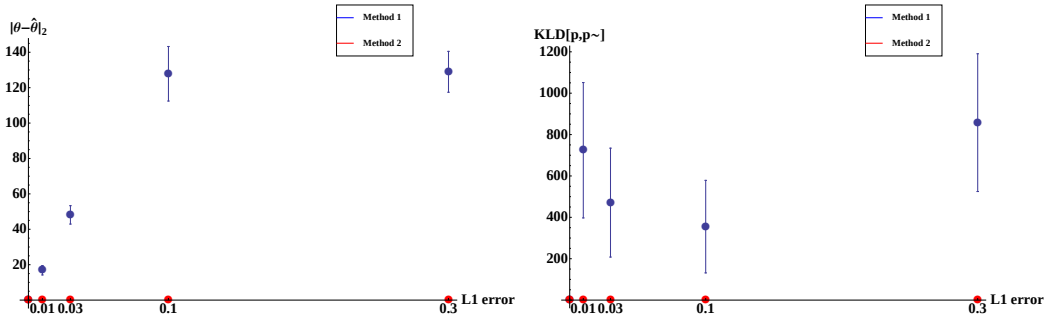
(a) Distance from $\hat{\theta}$ (b) Divergence: p to $\tilde{p}$

Figure 6.6: $\log_{10}$ of number of samples generated along the x-axis. In all cases, 20 instances were stacked, and no error was introduced. Method two finds a perfect solution with 100 or more samples, while method one may require 1000.

some extent on the distribution being sampled, these "samples" were generated by a nonrandom process capable of producing samples precisely distributed, within the resolution allowed by the desired total number of samples. To accomplish this, the distribution was discretized into a histogram with the stated total number of samples distributed appropriately into the bins. This allows us to demonstrate, apart from any particular sampling technique, what results are possible for a particular effective sample size. For this four-state system, 1000 samples are sufficient to represent any probability distribution accurately to three decimal places. However, GCCDdd solved for all twenty twenty-seven free coefficients and produced a solution which is exact to within 10^{-11}, with a standard deviation on the error of 2.4×10^{-11}.

A further comment on this figure. Note that we have plotted only one direction of the KL divergence. While the values of the two divergences are not actually the same, in every one of our experiments the difference between $\mathcal{D}_{\mathcal{KL}}(p, \tilde{p})$ and $\mathcal{D}_{\mathcal{KL}}(\tilde{p}, p)$ is sufficiently small that nothing additional is gained from displaying both, so we will show only $\mathcal{D}_{\mathcal{KL}}(p, \tilde{p})$.

Another natural question is, how much noise in the sampling process will be toler-



(a) Distance from $\hat{\theta}$

(b) Divergence: p to $\tilde{p}$

Figure 6.7: ℓ_1 error introduced as "incorrect" samples. In all cases, 20 instances were stacked, and 10^5 total samples were used. Method two finds an ideal solution in all cases, while method one's solution deteriorates as error is added.

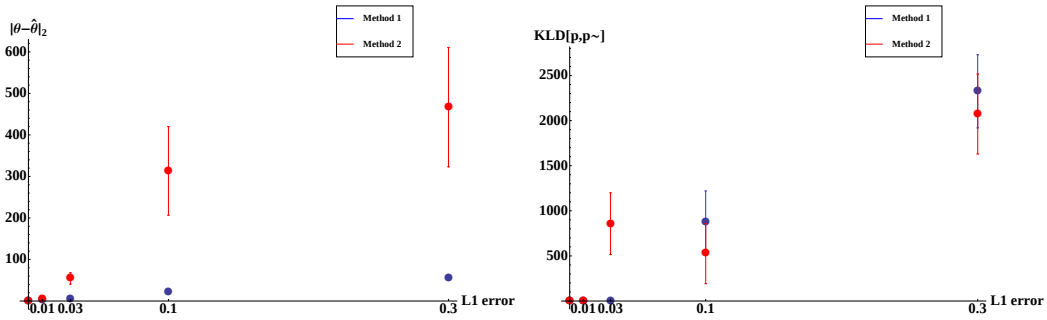
```
(* tgt: true distribution. erri: desired L1 error *)
l1n[tgt_, erri_] :=
  Module[{e, n},
   n = Length[tgt];
   e = rl1n[erri, n]; (* generate an error vector of same length *)
   While[Min[tgt + e] < 0, (* while this produces an invalid   *)
    e = rl1n[erri, n];     (* distribution, reject and retry   *)
   ];
   e
  ];
rl1n[err_, n_] :=
  Module[{p1, p2, errv},
   p1 = Normalize[RandomReal[{0, 1}, n], Total];
   p2 = Normalize[RandomReal[{0, 1}, n], Total];
   errv = (p1 - p2);
   errv = errv*err/Total[Abs[errv]];
   errv
  ];
```

Figure 6.8: Rejection sampler for probability distributions with desired ℓ_1 error

ated? This question is crucial, because the algorithm can be distilled to calculating a series of expectations with respect to the approximating distribution. Since GCCDdd accomplishes this by summing over samples, it is important to consider to what extent noise in the sampling process will affect the results of GCCD.

Figure 6.7 shows the results of an experiment adding errors to the deterministic sample-generating process used in the previous experiment. Error was added in the following way. The MCMC sampler was replaced by a "sampler" which calculated the exact probabilities corresponding to μ by summing energies over the four states, as in the previous experiment. The rejection sampling scheme described in Figure 6.8 was used to tweak these probabilities to produce the desired ℓ_1 error between the set of samples and the true probability. Then, "samples" were deterministically produced to correspond to the tweaked distribution, up to the resolution of the total number of samples used. This experiment is one of the principle reasons why we focus on "method 2" in this chapter. At least for this model, this method appears to be work perfectly without respect to errors in the sampling process, up to three tenths of the samples being drawn in a way designed not to match the target distribution.

This result is incredibly surprising. Because this algorithm is solving a set of equations consisting of expectations with respect to this set of samples, we expected the results to deteriorate as the sampling error increased, as was true for method one. Method two, however, found the correct solution in every trial. One possibility is that it is a coincidence of the model, that for instance the errors in this particular model cancel in a non-obvious way. The fact that method two out-performs method one is perhaps intuitive, in that, as mentioned at the end of Section 6.1, method two computes higher-order terms, and perhaps thereby incorporates additional information. However, this does not explain method two's apparent ability to find ideal solutions even with a large proportion of incorrectly distributed samples.



(a) Distance from $\hat{\theta}$

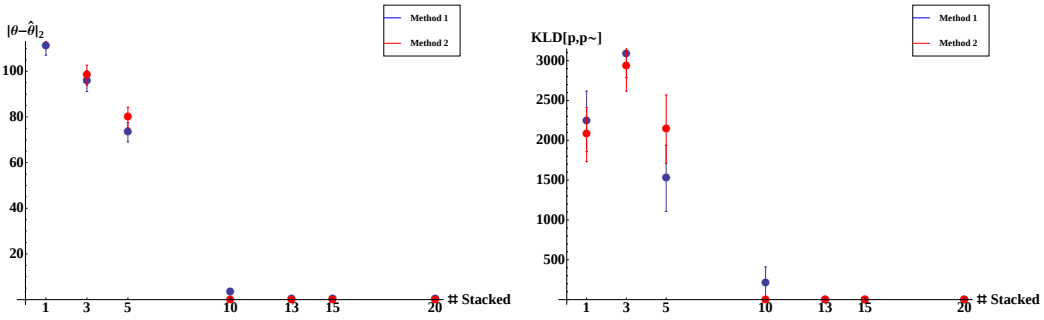
(b) Divergence: p to $\tilde{p}$

Figure 6.9: ℓ_1 error introduced into p. In all cases, 20 instances were stacked, and 10^5 total samples were used. Solutions with method one diverge less from $\hat{\theta}$, but neither performs well on the divergence metric with error greater than 0.03.

6.5.2 Input Concerns

As described in Section 6.2, the optimization routine requires matched pairs of p and μ. Figure 6.9 examines how accurately these pairs must represent one-another. This was produced by adding error, using the method from Figure 6.8 as in the previous experiment, to the input probabilities. Here, method one appears to have a slight advantage. The distance from the returned coefficients is uniformly better in every case for method one. Unfortunately, for the more important metric – quality of simulation, in terms of KL divergence – the two methods are essentially equivalent, with method one having an advantage only in the case of added error 0.03. This indicates that GCCD may be sensitive to the quality of the match between input pairs μ and p. This is striking, because μ parametrizes the approximating distribution $\tilde{p}$, and in general it may not be possible, for a given diagram, to provide a set of parameters μ which match a particular distribution p with the desired level of accuracy.

As described in Section 6.3.3, the dimensions of the matrix $\mathbf{A}$ are $n \times m$, where n is the number of potentials in the MRF and m is the number of bases, and vector $\mathbf{B}$ has length n. In order to achieve a solution which is fully constrained, we stack together



(a) Distance from $\hat{\theta}$

(b) Divergence: p to $\tilde{p}$

Figure 6.10: Number of p-μ pairs stacked. In all cases, 10^5 total samples were used and no error was introduced. Both methods require enough instances to produce a matrix $\mathbf{A}$ of full rank.

copies of $\mathbf{A}$ and $\mathbf{B}$, where the distribution p and the parameters μ of $\tilde{p}$ are varied. Figure 6.10 examines how many copies it is necessary to stack. A likely answer seems like m/n, which here is 9. Indeed, for this model, 10 seems to suffice.

Finally, in Figure 6.11 we plot the run time of each algorithm, as a function of the number of samples generated. For small numbers of samples, the methods are almost indistinguishable. As the number of samples grows, it becomes clear that method two is slower. As discussed in Section 6.1, method two is slower is because of the expectation over a trio of statistics present in Equation (6.14), which makes method two cubic in the size of the graph, while method one is quadratic. This difference becomes more apparent with more samples because the size of the multiplicative constant is driven predominantly by the process of averaging over samples, though here the size of the graph – three function nodes – is not so large that the difference between n^2 and n^3 is striking.

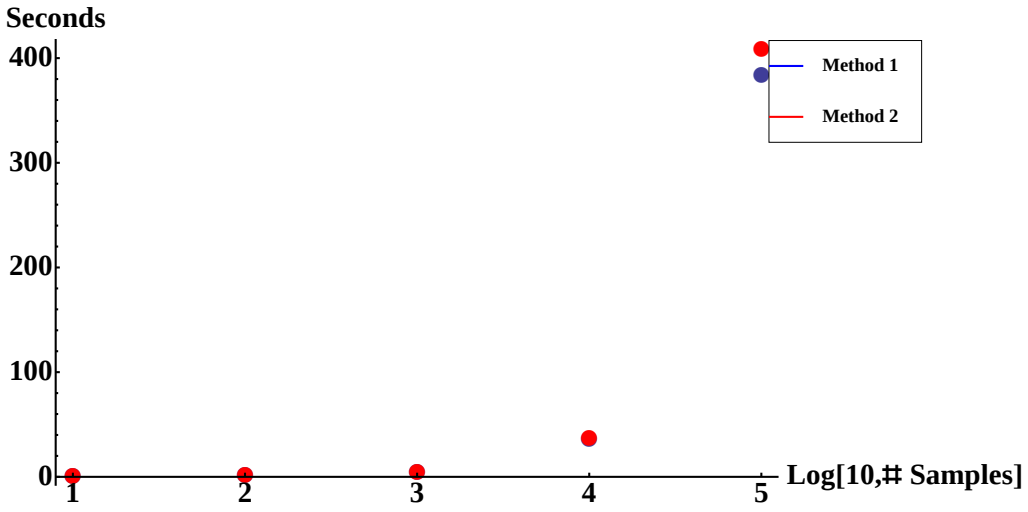


Figure 6.11: Run time comparison, $\log_{10}(\#\text{samples})$ on x-axis. In all cases, 20 instances were stacked and no error was introduced. For a small model using fewer than 100,000 samples, the two methods are indistinguishable.

6.6 Conclusions

We have demonstrated on a series of test problems that both methods of GCCD are capable of producing high-quality approximations for the time-evolution of probability distributions characterized by chemical reaction networks. Sections 6.3 and 6.4 develop ideal basis functions for two model systems, and in so doing provide some intuition about the algebraic forms of basis functions – in each case, the set of ideal basis functions was comprised entirely of functions formed by exponentiating a linear combination of the parameters μ. Furthermore, Figure 6.1 demonstrates the ability of GCCD to find good approximations when the ideal bases are not available.

Section 6.5 compares two methods of optimizing the parameters of a GCCD model, and evaluates each in terms of the quality of its optimization. Method two, developed in Section 6.1, is shown to be slower (see Figure 6.11) for the same model and same

number of samples. But, for small models, the number of samples generated is the dominant factor in determining run time, and the difference in the two methods is not so great to overcome the advantages of method two. The most significant advantage is method two's ability to produce high-quality optimizations even when the sampling process used to approximate the necessary expectations is far from optimal, as is demonstrated in Figure 6.7. This may explain the why, in Figure 6.6, method two requires fewer samples to find an ideal solution. The performance of methods one and two is otherwise quite similar, in that both are sensitive to mismatches between the input pairs of p and μ (Figure 6.9), and both require the same number of such pairs as input in order to produce a good approximation (Figure 6.10).

Both methods have a demonstrated ability to find optimal approximations, but we recommend method two, due to its tolerance for noise in the sampling process.

Chapter 7

GCCD: A Learning Approach

7.1 Introduction

Some of our attempts to model reaction networks with GCCD have met with more limited success than the examples of the previous chapter. Section 6.5.2, and Figure 6.9 in particular, highlights the importance of the accuracy of the match between p and μ as input to the GCCD algorithms derived in Section 6.1. Additionally, the approximation derived therein hinges on solving a linear system of equations, and we have not found any guarantees for the stability of the approximation when this system is ill-conditioned.

To overcome these limitations, we have developed a second approach to GCCD modeling. The foundations of this method are the same – an MRF, a set of basis functions – but the input and methodology are unique. With this technique, we will attempt to learn the coefficients of the basis functions by approximating the desired derivatives from data and fitting functions to these approximations with regularized linear regression.

Given an MRF and a set of basis functions as described in Sections 5.3 and 5.4, it is necessary to find the weights θ which allow the distribution, as specified by the parameters μ, to most accurately evolve in time. Fitting a GCCD model to data consists of several steps. First, the expected values of each of the factors in the MRF is computed from the data at a series of time points (Section 7.2.4). Then, from these statistics, a series of parameter values for the MRF is learned at each time point [1] (Section 7.3). From this series, approximate derivatives are computed (Section 7.4). Finally, these approximate derivatives are fit as a linear combination of a fixed set of basis functions, as in Equation (5.3) (Section 7.5).

We will approach this problem through an extended example, which models part of the calcium/calmodulin-dependent protein kinase cascade.

7.2 CaMKII

Calcium (Ca^{2+}) and calmodulin (CaM) are important molecules present in nearly every form of tissue [37], which work to relay signals from membranes into relevant machinery in many cell types [71]. Binding between these molecules is typically followed by CaM binding with one of several different kinases. The role of these molecules in the post-synaptic spine is essential to long-term potentiation and long-term depression, two processes crucial for the formation of memory.

Calcium enters the post-synaptic spine through NMDA receptors, which open ion channels in response to depolarization and binding of glutamate released from the pre-synaptic spine following stimulation [38, 17]. Calmodulin (CaM) is a regulatory protein which competes with several other Ca^{2+} targets to bind this influx of calcium. CaM consists of two "ends," each of which contains two Ca^{2+} binding domains [3].

Calmodulin is in turn competitively bound by at least six targets in neurons, one of which is CaMKII [38]. CaMKII consists of twelve CaM-binding subunits, each with a phosphorylation domain, arranged in a ring of six dimers [68]. Two of the molecule's four isoforms of CaMKII, α and β, are concentrated most heavily in the brain, and make up 2% of hippocampal proteins [72]. CaMKII is a vital part of the chemical process underling learning and memory, through its effect on long-term potentiation (LTP) and long-term depression (LTD) [45, 65]. Ca^{2+}/CaM complex binding to CaMKII stimulate its catalytic activity, and also initiate autophosphorylation [72]. Phosphorylation allows CaMKII to continue its catalytic activity independent of further calcium binding [63]. A diagram of subsequent CaMKII targets and the larger network controlling LTP and LTD available in [38] contains greater than fifty species.

Significant effort has been put into computational models of the Ca^{2+}/CaM/CaMKII pathway. Notably, the Cellerator framework for deterministic ODE simulation was used to build a model of the interaction between Ca^{2+}, CaM, and CaMKII monomers and dimers [63]. We apply GCCD to this model.

7.2.1 Probabilistic Graphical Model

We represent this model with the Markov random field in Figure 7.1. This model has binary variables for calcium binding to CaM receptors and for CaMKII phosphorylation, as well as indicator variables for individual CaM and CaMKII molecules binding and for CaMKII dimerization.

The variables in the second row represent binding of calcium to calmodulin, where $\mathrm{CaM}_{end,num,cam}$ represents the presence or absence of calcium on the numth binding site at the end end of the camth calmodulin molecule, where each runs from one to two. The functions in the top row represent biases on the likelihood of calcium

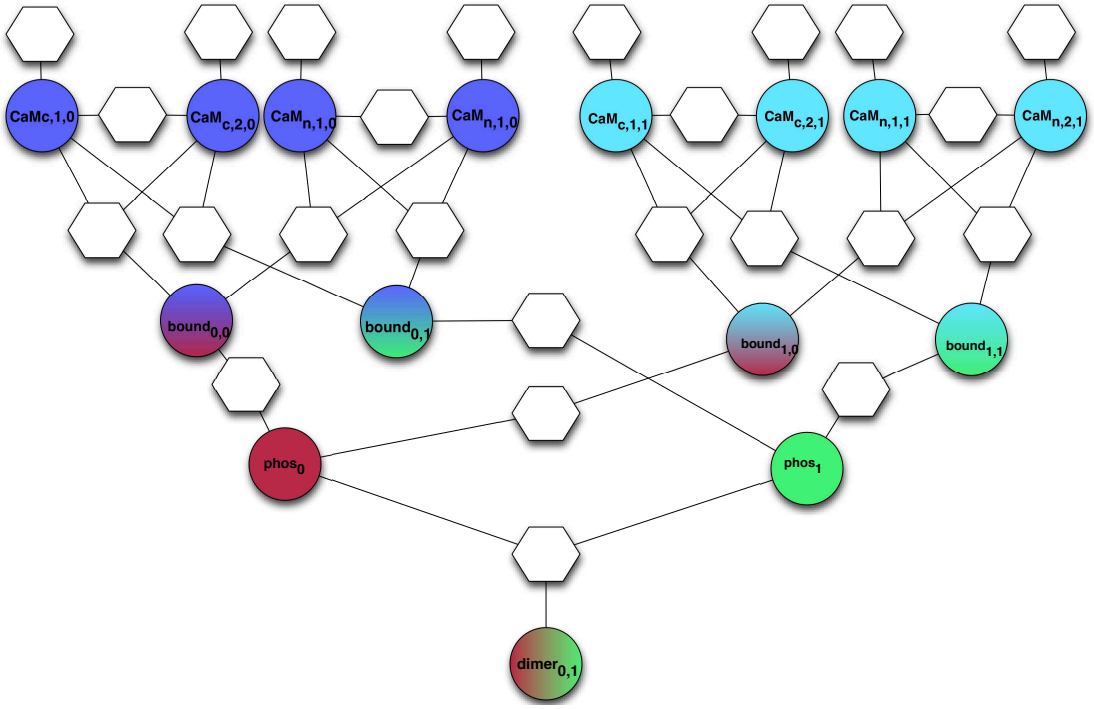


Figure 7.1: An MRF model of calcium binding, CaM/CaMKII interaction, and CaMKII dimerization.

binding, and the functions in the second row represent interactions between the two binding sites on each end of one calmodulin. The multicolored variables in the forth row represent binding of a calmodulin to a CaMKII molecule, so that $\text{bound}_{cam,kk}$ represents the camth CaM binding to the kkth CaMKII. (Not pictured are factors which limit these variables so that only one per calmodulin and one per CaMKII is "on" at one time. These factors don't have time-dependent dynamics, and so are elided here.) In the sixth row are variables which represent the phosphorylation of CaMKII, and in the eighth row a variable representing a possible dimerization of the two CaMKII molecules.

7.2.2 Preparing to Fit Ordinary Differential Equations

The MRF cannot represent continuous-time dynamics, and GCCD overcomes this limitation by specifying differential equations for the parameters of the MRF. In Section 5.4, we proposed a specific form for these dynamics which allows a GCCD model to be learned and also solved numerically as ordinary differential equations. As in the previous chapter, these differential equations are defined here to be linear combinations of a set of basis functions. That is, given a set $\mathcal{B}_\alpha$ of bases $f_A(\mu)$, the dynamics for MRF parameter μ_α is a linear combination of the form

$$\frac{d}{dt}\mu_\alpha = f_\alpha(\mu|\theta) = \sum_A \theta_A f_{\alpha A}(\mu)$$

The remaining work of modeling required for GCCD is the selection of these basis functions.

The basis functions fit to the CaMKII model were the unary functions $\frac{1}{1+|\mu_i|}$, e^{μ_i}, $e^{-\mu_i}$, $e^{-\frac{(\mu_i-c)^2}{2}}$ for c taking values between -3 and 3 in steps of size one-half, and μ_i^k for integer $k \in [1,5]$, and the binary functions $\mu_i\mu_j$, $e^{-2(\mu_i-\mu_j)^2}$, and $e^{-2(\mu_i+\mu_j)^2}$.

These basis functions include a variety of function types drawn from two principle sources: low-order polynomial terms, and exponentiated sums and differences. These motivation for these terms comes from a desire to mimic the optimal basis solutions in Sections 6.3 and 6.4. The power of two was included because it is important, in this learning framework, to avoid functions with poles. It is not uncommon for a function to model the discrete data points beautifully, but tend toward infinity in the space between two points. Given the desire to solve the system of differential equations continuously, this behavior is untenable. A potential area for future research might be a process for learning the basis functions, perhaps using a technique such as

symbolic regression [69]. We performed a few experiments with the Eureqa software package for symbolic regression in which we attempted to learn basis functions for use in GCCD, but never found any functions which, when used as basis functions in a GCCD model, improved our results. Nonetheless, the potential for progress here deserves more dedicated research.

7.2.3 Generating Simulation Data

We generated data sets using two stochastic simulation engines, Plenum and MCell, on related models.

Plenum [76] implements the stochastic parametrized grammar framework [52]. Two features make Plenum a particularly nice environment in which to model CaMKII. The ability to parametrize reactions and corresponding rates makes it possible to write out this model in a compact form, without generating unique rules for, eg, calcium binding to each possible position of a dimer in each possible state of binding and phosphorylation. Additionally, the ability to specify continuous, deterministic rules incorporating ordinary differential equations (ODEs) intermixed with stochastic reactions provides a natural way of implementing calcium spikes as a continuous function of time.

MCell [39] is a stochastic simulation system uniquely capable of modeling spatial dynamics and diffusion. An MCell simulation does not rely on the "well mixed" assumptions of SSA, as each molecule is tracked in time and space individually, and molecules are required to move into physical proximity before participating in a reaction.

The experiments below contain data from simulations with two different volumes. The

MCell experiments model a space with volume 8×10^{-21}, while the Plenum volume is one quarter of the size, 2×10^{-21}. The key numbers of molecules to consider are calcium when spiking, calcium between spikes, CaM, and CaMKII, and in the described simulations those numbers are, MCell: 48, .48, 145, 385 and Plenum: 12, .12, 36, 96. These correspond to micromolar concentrations of 10, .1, 30, and 80. Fractional calciums are used in the Plenum simulation to approximate a process whereby calcium flows in an out at a rate that leaves an expected fractional number of calcium at any point in time, but which is very expensive to simulate. Both models used a "square wave" spike, in which calcium was either injected steadily to maintain a particular maximum count (MCell) or fixed at a particular level for the duration of a spike (Plenum).

The Plenum model is reproduced here, split into Model 1 and Model 2. Model 1 contains the reactions which deal with calcium – the `clock` species which allows timed spikes, the spiking reactions themselves, and binding of calcium to CaM, both free CaM and that bound to CaMKII. Model 2 contains the reactions via which CaM binds to CaMKII, CaMKII dimerizes, and the dimers autophosphorylate. The calcium level and free CaM are represented by numbers, as `Ca[ii]` and `CaM[num]`, respectively. Other species are represented individually, as plainly as possible. The reaction rates are all scaled by `timeMultiplier`, which was set at 10^4. Because the `spikeOn` and `spikeOff` reactions are designed to be effectively instantaneous, they require reaction times much faster than the other reactions, but must fire within precise windows in time. The scaling effectively slows down the model's timing system, `clock`, extending the width of these windows to ensure that Plenum doesn't miss a firing.

```
(* this keeps track of time in the simulation, so we can have timed spikes *)
c==clock[time,state] -> c, solving[time'==1/timeMultiplier],

(* add Ca if it's time to do so *)
{c==clock[time,s], Ca[ii]} -> {clock[time,1], Ca[spikeTrainSize]},
  with[If[ii < spikeTrainSize, spikeOn[time,s]/timeMultiplier, 0]],

(* remove Ca if it's time to do so *)
{c==clock[time,s], Ca[ii]} -> {clock[time,0], Ca[baseSize]},
  with[If[ii > 0,spikeOff[time,s]/timeMultiplier, 0]],

(* Ca binding/unbinding CaM *)
{Ca[ii], CaM[n,c]} -> {Ca[ii], CaM[n+1,c]},
  with[ii * If[n < amax, kloadn[n,c], 0]/timeMultiplier],
{Ca[ii], CaM[n,c]} -> {Ca[ii], CaM[n-1,c]},
  with[If[n > 0, kunloadn[n,c], 0]/timeMultiplier],
{Ca[ii], CaM[n,c]} -> {Ca[ii], CaM[n,c+1]},
  with[ii * If[c < amax, kloadc[n,c], 0]/timeMultiplier],
{Ca[ii], CaM[n,c]} -> {Ca[ii], CaM[n,c-1]},
  with[If[c > 0, kunloadc[n,c], 0]/timeMultiplier],

(* Ca binding/unbinding CaM bound to CaMKII *)
{Ca[ii], Kk[a0,b0,0]} -> {Ca[ii], Kk[a0+1,b0,0]},
  with[ii*If[a0<amax, kload2n[a0,b0,0], 0]/timeMultiplier],
{Ca[ii], Kk[a0,b0,0]} -> {Ca[ii], Kk[a0,b0+1,0]},
  with[ii*If[b0<amax, kload2c[a0,b0,0],0]/timeMultiplier],
{Ca[ii], Kk[a0,b0,0]} -> {Ca[ii], Kk[a0-1,b0,0]},
  with[If[a0>0,kunload2n[a0,b0,0],0]/timeMultiplier],
{Ca[ii], Kk[a0,b0,0]} -> {Ca[ii], Kk[a0,b0-1,0]},
  with[If[b0>0,kunload2c[a0,b0,0],0]/timeMultiplier],
```

Model 1: Plenum model of Ca+/CaMKII binding.

```
(* CaM binding/unbinding free CaMKII *)
{CaM[n,c], CaMKII[num]} -> {Kk[n,c,0], CaMKII[num-1]},
  with[num*kon2[n,c,p0]/timeMultiplier],
{Kk[a0,b0,0], CaMKII[num]} -> {CaM[a0,b0], CaMKII[num+1]},
  with[koff2[a0,b0,0]If[a0>=0&&b0>=0,1,0]/timeMultiplier],

(* Dimerization *)
{Kk[a0,b0,p0], Kk[a1,b1,p1]} -> {Dimer[a0,b0,p0,a1,b1,p1]},
  with[If[p0<1||p1<1 && a0<=a1,kdimerize[a0,b0,p0,a1,b1,p1]/timeMultiplier,0]],
{Dimer[a0,b0,p0,a1,b1,p1]} -> {Kk[a0,b0,p0],Kk[a1,b1,p1]},
  with[kundimerize[a0,b0,p0,a1,b1,p1]/timeMultiplier],

(* phosphorylation *)
Dimer[a0,b0,p0,a1,b1,p1] -> {Kk[a0,b0,1],Kk[a1,b1,p1]},
  with[If[a0>=0&&b0>=0&&p0<1,kautotop[a0,b0,a1,b1,p1],0]/timeMultiplier],
Dimer[a0,b0,p0,a1,b1,p1] -> {Kk[a0,b0,p0],Kk[a1,b1,1]},
  with[If[a1>=0&&b1>=0&&p1<1,kautobot[a0,b0,p0,a1,b1],0]/timeMultiplier]
```

Model 2: Plenum model of Ca+/CaMKII binding.

7.2.4 Computing Simulation Statistics

Given some simulation data which presents, for a volume of some size, a time series of the number of molecules in each of the possible states of these two species, we compute the expected values of the various functions of the MRF. Since in such a data set the various molecules are indistinguishable, it suffices to calculate only eight expected values: biases for calcium binding on each end ($\text{CaM}_{c|n,i,j}$for i 1 or 2 and CaM index j), interactions for calcium binding on each end ($\text{CaM}_{c|n,1,i}\text{CaM}_{c|n,2,i}$ for CaM index i), interactions between calcium binding and CaM binding ($\text{CaM}_{c|n,1,i}\text{CaM}_{c|n,2,i}\text{bound}_{i,j}$ for CaM index i and CaMKII index j), an interaction between CaM binding and phosphorylation ($\text{bound}_{i,j}\text{Kkp}_j$for CaM index i and CaMKII index j), and an interaction between dimerization and phosphorylation ($\text{Kkp}_i\text{Kkp}_j\text{dimer}_{i,j}$for CaMKII indices i and j). Some of these statistics are represented by more than one potential in the MRF in Figure 7.1, but the learning approach that we take will only learn one

copy of the dynamics for each.

Figures 7.2 and 7.3 show such time series. Note that the data has been restricted to periods where calcium is spiking into the simulation. Each spike lasts 16×10^{-3}seconds, and the spiking rate is 8Hz. Because GCCD is based on time-independent dynamics (see Section 5.2 and especially Equation (5.1)), and because the dynamics are different between times when calcium is spiking into the model and inter-spike time, the inter-spike time is elided here and in the corresponding figures in following sections. A method of fitting together models for a complete picture is presented in Section **??**.

This process requires accounting for the translation of data in molecule-count format into statistics corresponding to expected values of MRF functions. For instance, a count such as pKCaM1N2C_KCaM1N1C: 3 represents, among other things, three instances of $\text{Kkp}_i = 1$ and three instances of $\text{dimer}_{j,k} = 1$, but also many more instances of $\text{dimer}_{j,l} = -1$. It is necessary to account for all of this together when computing $\left\langle \text{Kkp}_i \text{Kkp}_j \text{dimer}_{i,j} \right\rangle$. In particular, these statistics therefore encode some information about the number of molecules represented. Additionally, it may seem that we conflate the expected state of one molecule of many in a single simulation with the expected state of one molecule across many simulations. However, these statistics average over many simulations as well, and our inability to distinguish molecules between time steps essentially forces our hand in averaging across them within a time slice as well.

7.3 Markov random field Parameter Inference

The next phase takes these time series of these eight statistics as input, and learns a time series of parameters μ that will induce the MRF to express them. Figure 7.4

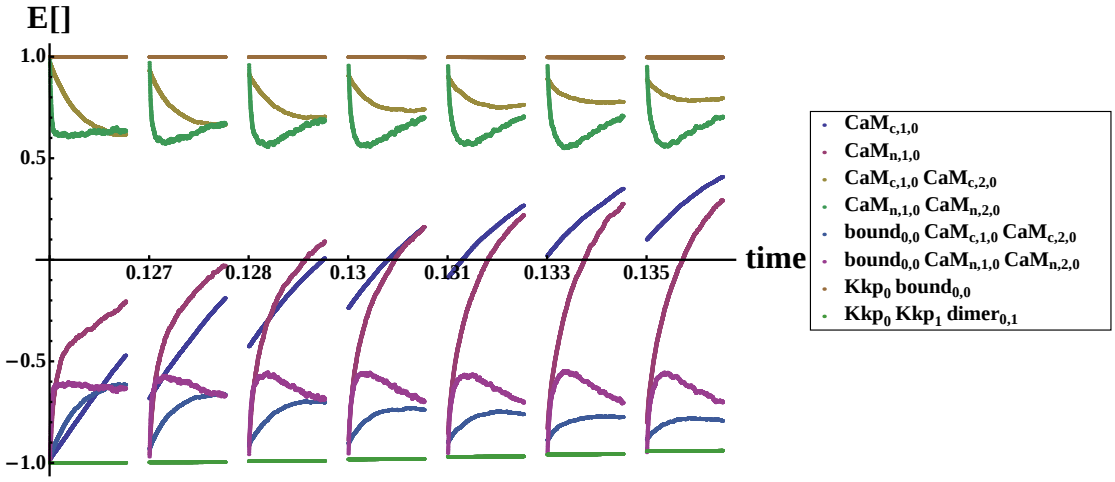


Figure 7.2: Time series of eight key statistics, from simulation data generated by MCell

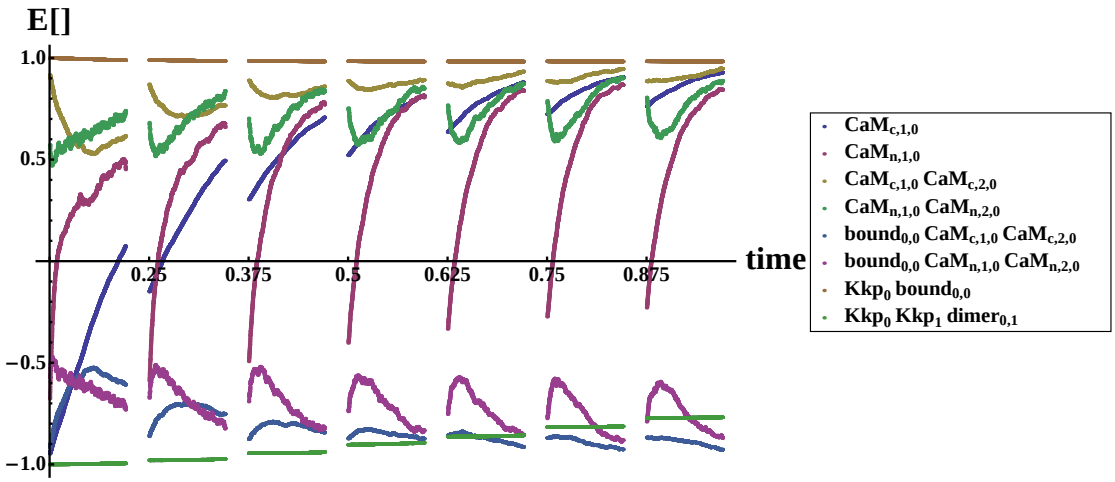


Figure 7.3: Time series of eight key statistics, from simulation data generated by Plenum

shows such a time series for the statistics in Figure 7.2.

This learning process considers an MRF of a fixed size, where the relevant eight statistics calculated in Section 7.2.4 represent the expected values of eight factors repeated an appropriate number of times. The use of an MRF to represent a reduced model means that we explicitly represent two CaM and two CaMKII, which is the minimal amount of duplication of information necessary to represent all of the important binding.

Learning the MRF parameters μ is performed using gradient descent at each time point, with the well-known Boltzmann machine learning algorithm [1]. This requires evaluating, for a parameter setting μ', expected values $\langle\phi\rangle_{\mu'}$ for all of the factors ϕ of the MRF. This is achieved using exact probabilities via belief propagation [62] and the junction tree algorithm [43]. Junction trees are computed using a *Mathematica* package developed by Yaroslav Bulatov [10]. For MRF models where the tree-width prohibits exact inference, it will be necessary to consider approximations instead. However, since one of the goals of GCCD is model reduction, it may be common to keep the MRF representation compact.

7.4 Derivative Approximation

From this time series it is possible to approximate the derivative of each parameter value, by calculating finite differences and applying Gaussian smoothing. The discrete samples were convolved with a discrete subset of the first derivative of a Gaussian with σ=3; the Gaussian was discretized at steps of size 10^{-4}. The resulting kernel is depicted in Figure 7.6. Figures 7.7 and 7.8 show the results of the derivative approximation process for the two simulation data sets.

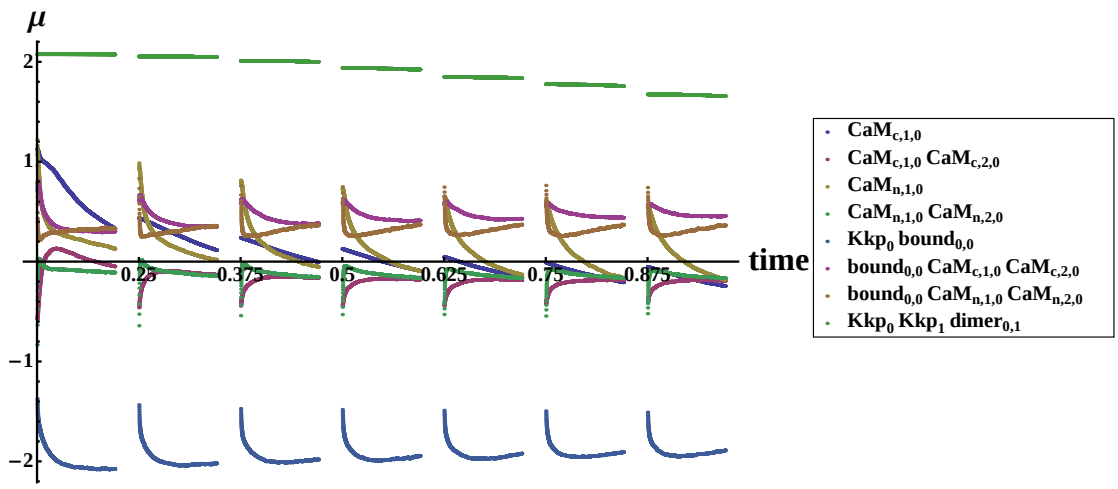


Figure 7.4: Time series of MRF parameter values learned from MCell simulation results.

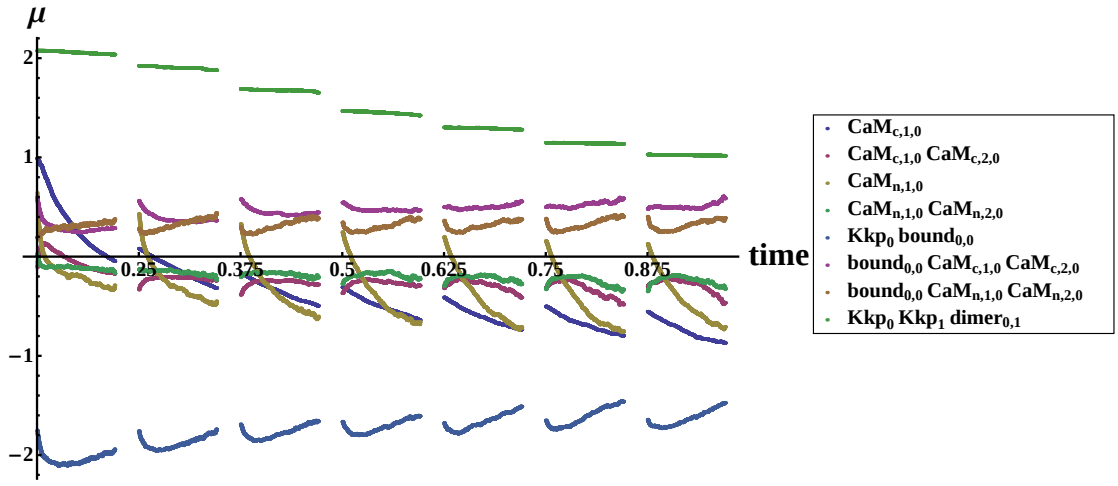


Figure 7.5: Time series of MRF parameter values learned from Plenum simulation results.

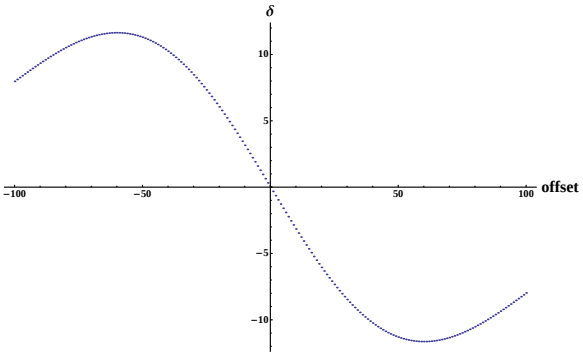


Figure 7.6: Gaussian derivative kernel, used to take the smoothed derivative of learned μ values.

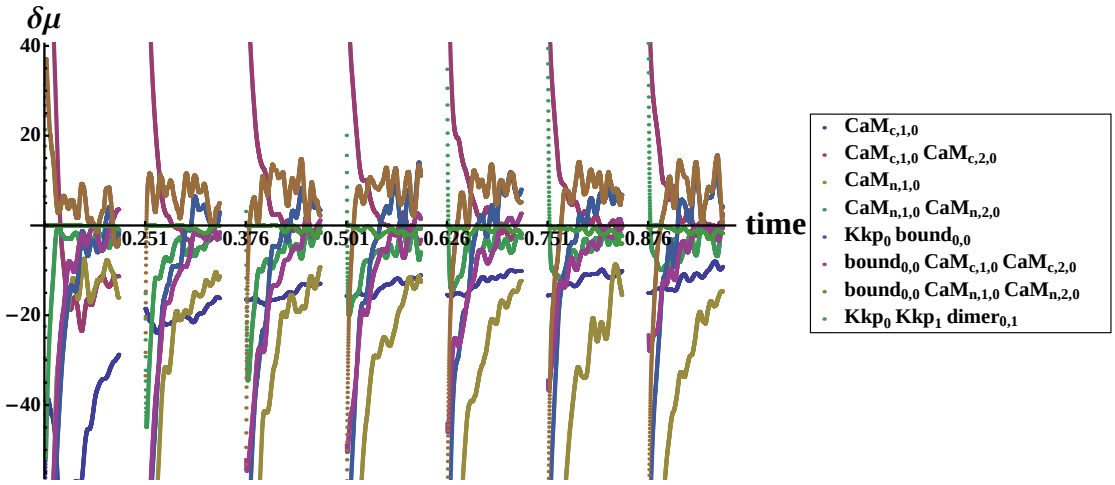


Figure 7.7: Approximate derivatives of MRF parameters (MCell).

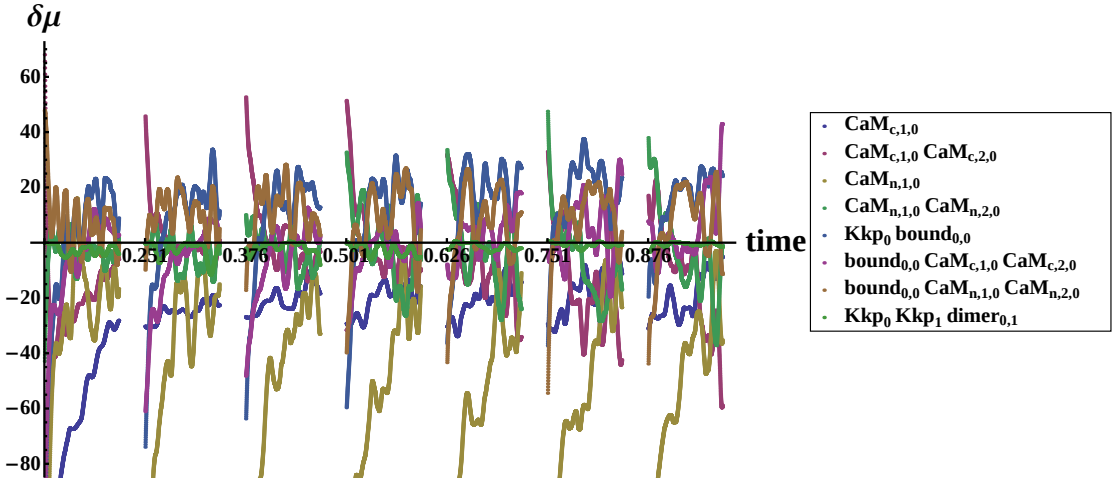


Figure 7.8: Approximate derivatives of MRF parameters (Plenum).

7.5 Function Fitting

Given the smoothed $\partial\mu/\partial t$ data, we construct a set of data sets, each of which has at each time point input variables consisting of all eight μ and an output variable corresponding to one of the derivatives. This gives us eight functions to fit.

We employ linear regression and lasso regularization for this learning [22]. This choice allows the learning method to fit in the same ODE framework developed in Chapter 6, and lasso will keep the function representation compact. Lasso refers to the "least absolute shrinkage and selection operator," and is a method of regularized regression which shrinks some coefficients and sets others to zero. We have performed exploratory experiments to attempt to use lasso to find promising bases to use with either method one or method two derived previously, but with limited success.

In particular we have fit each derivative as a linear combination of the 253 basis functions described in Section 7.2.2, each of which involves one or two of the μ parameters. Use of the lasso algorithm requires setting a parameter λ, which controls the number of nonzero coefficients in the resulting solution. This parameter was set using cross-validation, for which each of the "calcium spikes" was held out in turn.

We performed this process three times: once using MCell data, once using Plenum data, and once using all of the data together. Across these three optimizations, 103 of the 253 available bases were assigned non-zero coefficients in fitting at least one of the derivatives.

One hope for the learning process was that it might provide some information about the *types* of functions which make valuable basis functions for this model, as we would have liked to have compared positive results on this method with an approach to this model using the methods of Chapter 6. Figure 7.9 provides an overview of the bases

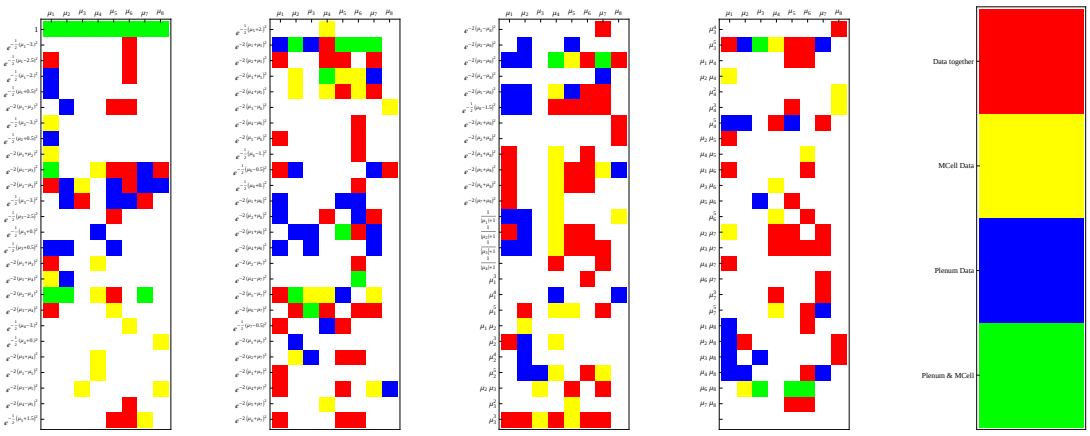


Figure 7.9: Color-coded matrix showing which bases (rows) were used in fitting which derivatives (columns) to which data set(s) (colors). The 150 basis which were never used are not drawn.

selected by lasso, where the presence of an appropriately-colored square represents a non-zero coefficient when learning with a particular data set for the relevant basis in fitting the relevant derivative. Unfortunately, we were unable to draw conclusions from this analysis that proved valuable for the other GCCD methods.

The result of this process is a set of ODEs, which can be solved numerically and compared to the time series of parameter values. Figures 7.10 and 7.11 show the first spike of the MCell and Plenum data sets respectively, each along with numerically solved differential equations initialized to the first time point of the relevant data set.

In each figure, the learned values of μ are represented in different colors and the numerically-solved ODEs are drawn in red. In each case, the overlap is sufficiently good that the red lines are almost entirely obscured by the data. This demonstrates GCCD's ability to match a time series of parameter values. This ability allows GCCD to simulate the time-evolution of the modeled probability distribution.

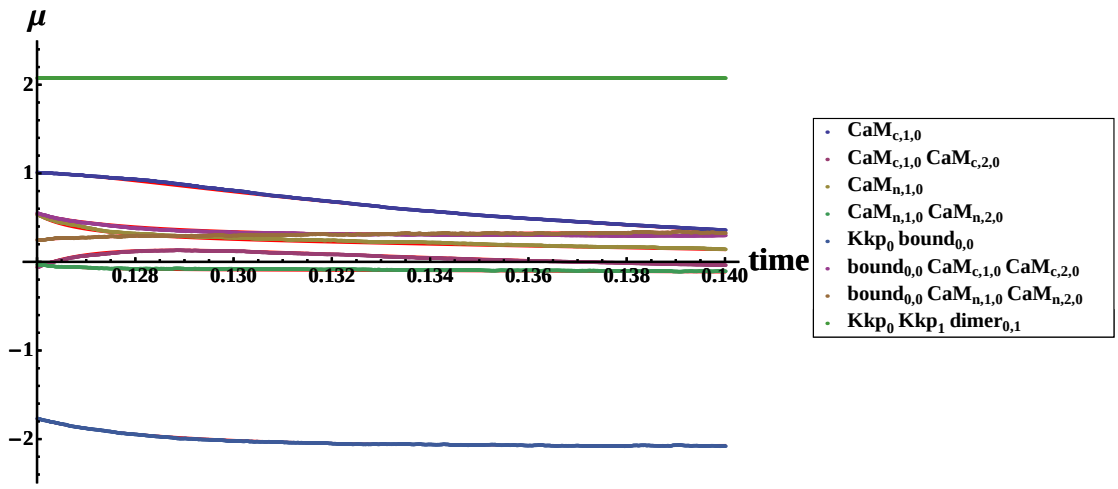


Figure 7.10: Set of ordinary differential equations with learned coefficients (red lines) versus time series of eight MRF parameter values (colored lines) (MCell).

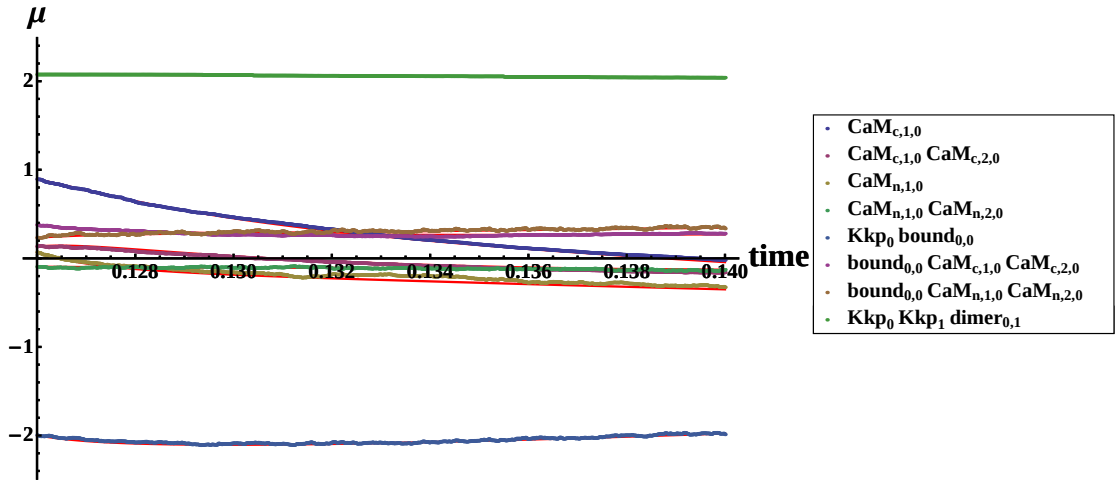


Figure 7.11: Set of ordinary differential equations with learned coefficients (red lines) versus time series of eight MRF parameter values (colored lines) (Plenum).

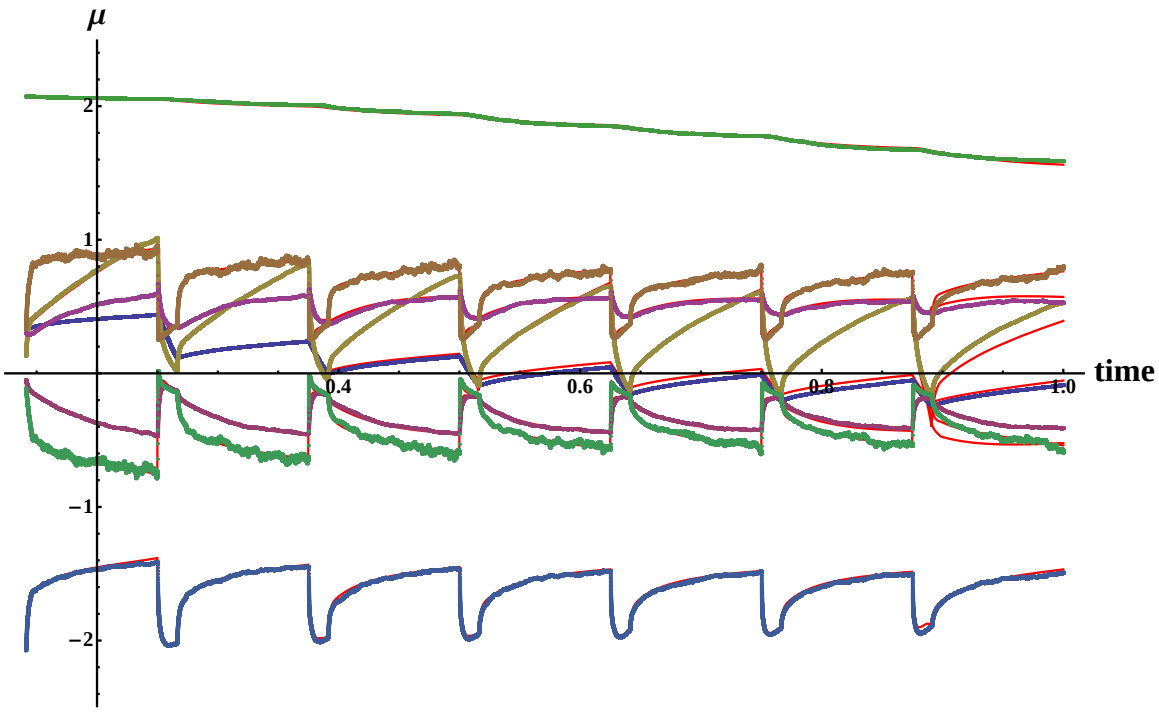


Figure 7.12: Set of ordinary differential equations with learned coefficients (red lines) versus time series of eight MRF parameter values (colored lines) (MCell), spike train.

7.6 A Complete Model

As described in Section 7.2.4, our figures so far have focused on only part of the CaMKII model. In order to adequately model both regimes – when calcium is abundant and when it is scarce – we have opted to fit differential equations to data from both of these times independently, and tojoin them into a complete model by numerically solving each for an appropriate amount of time before switching to the other. In order to achieve the highest quality of fit on MCell data, we have actually fit four models. One models for the very fast changes which occur early in a spike, and runs for 1.6ms; one models the duration of the spike, and runs for about 14ms; one models the early inter-spike time, and runs for 8ms; the last models the rest of the inter-spike time, and runs for about 110ms. Figure 7.12 shows a plot analogous to Figure 7.10, but for a series of successive calcium spikes and corresponding intermediate times.

7.7 Predicting New Spiking Patterns

After achieving the preceding encouraging results, we began to consider a GCCD model for a more biologically accurate spiking model. In particular, the spiking behavior of real neurons does not match the square wave we used to simulate calcium influx in Model 1. A more realistic family of models of calcium transport, called alpha functions [18], take the form

$$\alpha(t|t_1, t_2) = c * e^{-\frac{t}{t1}} \left(1 - e^{-\frac{t}{t2}}\right)$$

Here, t_1 controls how quickly calcium is added to the cell, and t_2 controls how quickly it is removed. The constant c sets the peak height.

We implemented a Plenum model equivalent to that in Section 7.2.3, but replaced the square wave with an alpha model using constants $c = 5747.53$, $t_1 = 5.9$, and $t_2 = 1244.55$. This choice of parameters introduces the same total amount of calcium, but it is introduced more gradually. The CaM/CaMKII binding portion of this model is exactly the same as Model 2, with the calcium reactions of Model 1 replaced by Model 3.

In Model 3, we have moved calcium from a species explicitly represented in the state to a function of time. The function `ca[time]` implements the $\alpha(t|5.9, 1244.55)$ function described above.

From this model, we followed the procedures described above – for simulating data, computing statistics, learning MRF parameters, approximating derivatives, and function fitting – for systems which spiked at a variety of rates. In particular, we simulated 2Hz, 4Hz, 7Hz, 8Hz, and 10Hz, for one second each. We then set aside the 8Hz data, and learned one model of "spiking" dynamics and one model of "inter-spike" dynam-

```
(* this keeps track of time in the simulation, so we can have timed \
spikes *)
c == clock[time] -> c, solving[time' == 1/timeMultiplier],

(* Ca binding CaM  corresponds to Pepke et. al, Figure 2A *)
{cc == clock[time], CaM[n, c]} -> {cc, CaM[n + 1, c]},
  with[ca[time]* If[n < amax, kloadn[n, c], 0]],
{CaM[n, c]} -> {CaM[n - 1, c]}, with[
 If[n > 0,  kunloadn[n, c], 0]],
{cc == clock[time], CaM[n, c]} -> {cc, CaM[n, c + 1]},
  with[ca[time]* If[c < amax, kloadc[n, c], 0]],
{CaM[n, c]} -> {CaM[n, c - 1]},
  with[If[c > 0,  kunloadc[n, c], 0]],


(* Ca binding CaM bound to CaMKII corresponds to Pepke et al, Figure 2B *)
{cc == clock[time], Kk[a0, b0, 0]} -> {cc, Kk[a0 + 1, b0, 0]},
  with[ca[time]*If[a0 < amax && a0 >= 0, kload2n[a0, b0, 0], 0]],
{cc == clock[time], Kk[a0, b0, 0]} -> {cc, Kk[a0, b0 + 1, 0]},
  with[ca[time]*If[b0 < amax && b0 >= 0, kload2c[a0, b0, 0], 0]],
{Kk[a0, b0, 0]} -> {Kk[a0 - 1, b0, 0]},
  with[If[a0 > 0, kunload2n[a0, b0, 0], 0]],
{Kk[a0, b0, 0]} -> {Kk[a0, b0 - 1, 0]},
  with[If[b0 > 0, kunload2c[a0, b0, 0], 0]],
```

Model 3: Plenum model of Ca+/CaMKII binding with alpha-function spiking.

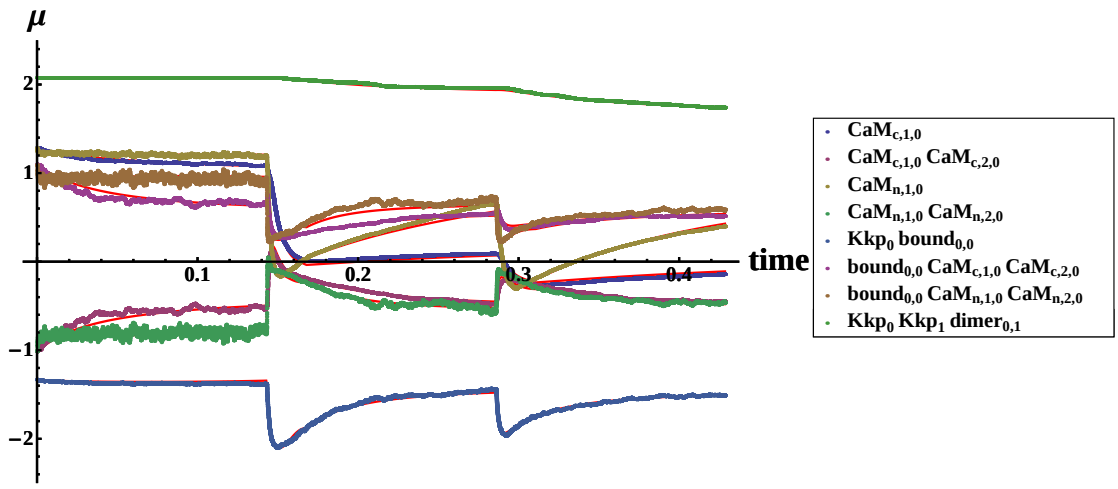


Figure 7.13: Set of ordinary differential equations with learned coefficients (red lines) versus time series of eight MRF parameter values (colored lines) (Plenum), alpha spike train at 7Hz.

ics for all of the remaining data combined. This represents a potentially confounding simplification for the model, as the amount of calcium present changes dramatically for the duration of the spiking model, which means the dynamics are not constant for this period. A future model which incorporates the value of calcium over time may be able to improve the results.

The present model is capable of reproducing MRF parameters for spiking behavior at a variety of rates, by always solving the spiking dynamics for a fixed time, but running the inter-spike dynamics for the appropriate length for a particular spiking rate. Figures 7.13 and Figure 7.14 show the fit to input parameters, while Figure 7.15 shows the fit to held out data. Unfortunately, at this time the model becomes unstable after a few spikes and the fit deteriorates. Additional work is required to extend these results to arbitrary series of spikes.

In summary, GCCD has a demonstrated ability to model input data with a continuously evolving probability distribution, by optimizing and solving a set of differential equations for the parameters of such a distribution. This capability would allow

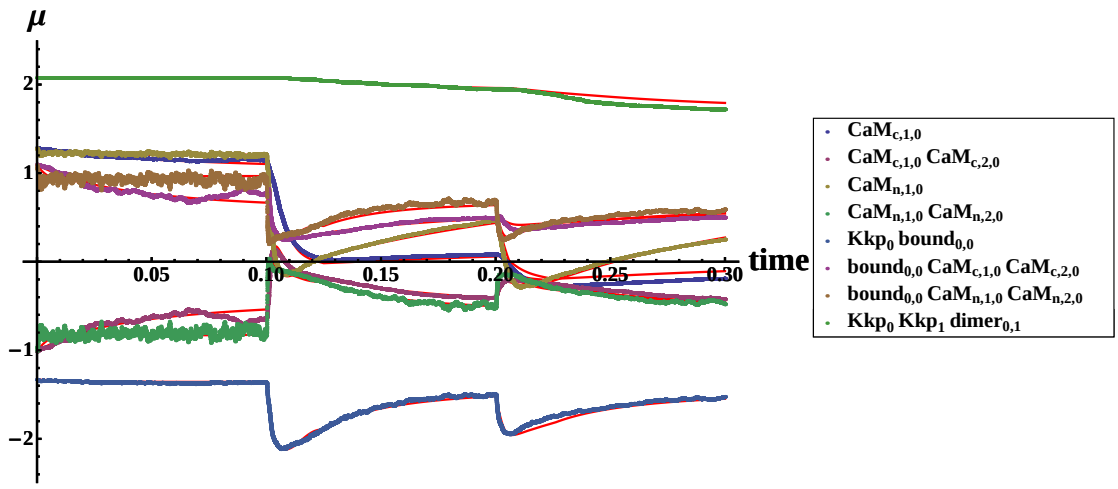


Figure 7.14: Set of ordinary differential equations with learned coefficients (red lines) versus time series of eight MRF parameter values (colored lines) (Plenum), alpha spike train at 10Hz.

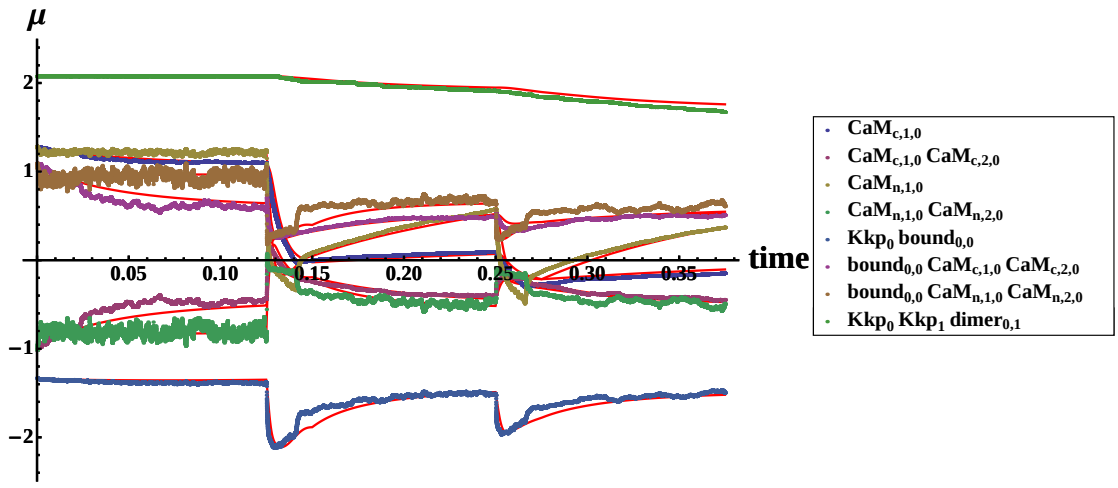


Figure 7.15: Set of ordinary differential equations with learned coefficients (red lines) versus held out time series of eight MRF parameter values (colored lines) (Plenum), alpha spike train at 8Hz.

GCCD to be used in place of a stochastic model for a subset of a larger modeling effort. In this scenario, a stochastic model for a subset of the species in a larger model would be used to generate data, which would be processed with GCCD as in this chapter. Then, GCCD would be used to circumvent the need to sample trajectories over these species in the larger model, while allowing the larger model to produce an appropriately-distributed sample of this subset of species at any time.

Additionally, GCCD has a demonstrated ability to interpolate over modeling scenarios, as in the spiking rate simulations above. Therefore there is also potential to use GCCD to avoid producing many stochastic sample trajectories for every starting condition of a model. In this scenario, GCCD would be fit to stochastic simulation data for a subset of relevant starting conditions, and used to predict trajectories over a wider range of conditions. Finally, GCCD might even be used to fit a probabilistic model of experimental data, which would be processed in the same was as the samples generated in this chapter.

Chapter 8

Conclusion

8.1 Dependency Diagrams

Dependency Diagrams constitute an extension of graphical models to better express variable-structure systems, to express iterative algorithms for sampling and inference, and to support graph-level transformations from models to such algorithms. We have given definitions of node and link types for indexing, gating, and constraints including those required for iteration, as well as their semantic map Ψ to PDFs. We defined a set of graph-level transformations $\mathcal{E}$ that can eliminate some link types in favor of others, and a graph transformation $\mathcal{T}$ that maps models to MCMC algorithms. We have also implemented these data types, the functional Ψ, the transformation $\mathcal{T}$, and translation to autogenerated code using a computer algebra system, and have demonstrated the implementation on several small examples of widely varying character: clustering, reaction rate inference, and a geometric prior for sparse graphs which we argue is relevant to current work in computational biology and potentially many other applications. These examples exhibit not only the power and accuracy

of the automated inference framework built into the Dependency Diagram system, but also the flexibility of Dependency Diagrams to encode numerous types of models. Because the semantics extend chain and factor graphs, Dependency Diagrams are not restricted to first-order generalizations of Bayesian networks or Markov random fields; the examples presented here were encoded using a combination of directed and undirected models, and were handled by the same automated inference system. This distinguishes Dependency Diagrams from systems built upon extensions of Bayesian networks (such as BUGS and BLOG) and from those built upon extensions of Markov random fields (such as Markov logic networks), and means that the most "natural" representation of each individual model need not be sacrificed to the restrictions of the modeling paradigm.

Many limitations remain to be overcome both in the mathematics and in the implementation. Applying a single generic MCMC engine, and entirely embedded in a computer algebra system, the generated code is inefficient at sampling and inference. But the early stages of autocode generation could be modified to generate much more efficient update steps and also to recognize and exploit special-case opportunities such as gating links, and the final stage could be replaced with output to a conventional compiled language. Many other node and link types remain to be implemented and tried out for usability. For example nested indexing would introduce non-uniqueness in model representation, but could enable semantics-preserving transformations to divide-and-conquer algorithms. The diagram layout and rendering is crude as yet; but it is both improvable and possibly self-applicable. Finally Dependency Diagrams may also be self-applicable in search algorithms for model selection, for example by compactly expressing suitable graph priors.

In short, Dependency Diagrams represent a significant extension of existing probabilistic graphical modeling frameworks because they allow more information to be

encoded directly into the graphical structure of a model. We have presented a novel use of this information: the conversion of a model diagram into an algorithmic diagram, and the subsequent generation of runnable code from the algorithmic diagram. We believe that future applications and algorithmic research within the Dependency Diagram system will prove valuable to many fields.

8.2 Graph-Constrained Correlation Dynamics

The GCCD framework provides a capability that has been neglected in the existing literature on stochastic reaction networks: the ability to model a goal-directed approximation of the probability distribution over states in time. Because an optimized GCCD model is deterministic and compact, it has the potential to be exceptionally useful in applications where it is necessary to represent the distribution over a subset of a larger stochastic model, but desirable to do so with a minimal commitment of computational resources.

Additional work is required to understand which models, in terms both of reaction networks and Markov random fields, are good candidates for GCCD modeling. Further additional research might profitably illuminate a more comprehensive set of, or general theory for, the basis functions necessary to represent particular classes of MRFs. These avenues of research would dramatically increase the general applicability of GCCD modeling.

GCCD also presents many additional opportunities for future development. For instance, the solution of Equation 6.16 is presently computed using matrix pseudo-inversion. However, the empirical technique for generating sparsity described in Section 6.4 coupled with the success of lasso in generating sparse solutions in Chapter 7

suggests it may be fruitful to consider additional numerical solutions, such as lasso or perhaps a form of Bayesian regularized regression, for methods 1 and 2. Positive results along these lines may even be able to reduce the number of instances it is necessary to stack.

Nonetheless, the results displayed by GCCD in fitting the small examples and the CaMKII model demonstrate that GCCD is a valuable addition to the stochastic modeling toolbox.

GCCD encompasses two derivations of approximations which are optimal in terms of Kullback-Leibler divergence, as well as a learning method which derives an approximation from data. Each of these methods is capable of producing high-quality approximations for the time-evolution of probability distributions characterized by stochastic processes. Sections 6.3 and 6.4 develop ideal basis functions for two model systems, and in so doing provide some intuition about the algebraic forms of basis functions – in each case, the set of ideal basis functions was comprised entirely of functions formed by exponentiating a linear combination of the parameters μ.

Furthermore, we have demonstrated the ability of GCCD to find good approximations when the ideal bases are not available, and to model systems of biological significance. GCCD shows tremendous potential for model reduction, as it will allow sub-modules of larger stochastic models to be replaced by deterministic equation solving, without sacrificing the ability to represent or sample a distribution over states at any point in time.

Bibliography

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985. ISSN 03640213. doi: 10.1016/S0364-0213(85)80012-4. URL http://doi.wiley.com/10.1016/S0364-0213(85)80012-4.

[2] D. F. Anderson and D. J. Higham. Multi-level Monte Carlo for stochastically modeled chemical kinetic systems. *Arxiv preprint arXiv11072181*, page 26, 2011. URL http://arxiv.org/abs/1107.2181.

[3] Y. S. Babu, C. E. Bugg, and W. J. Cook. Structure of calmodulin refined at 2.2 A resolution. *Journal of Molecular Biology*, 204(1):191–204, 1988. URL http://www.ncbi.nlm.nih.gov/pubmed/1474585.

[4] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966. ISSN 00034851. URL http://www.jstor.org/stable/2238772.

[5] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The {ALARM} Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, pages 247–256, London, Aug. 1989. Springer-Verlang.

[6] M. K. Bennett, N. E. Erondu, and M. B. Kennedy. Purification and characterization of a calmodulin-dependent protein kinase that is highly concentrated in brain. *The Journal of Biological Chemistry*, 258(20):12735–12744, 1983. URL http://www.ncbi.nlm.nih.gov/pubmed/6313675.

[7] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, Series B*, 36(2):192–236, 1974.

[8] A. Bhan and E. Mjolsness. Static and dynamic models of biological networks. *Complexity*, 11(6):57–63, 2006. ISSN 1076-2787. doi: http://dx.doi.org/10.1002/cplx.v11:6.

[9] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

ISSN 0001-0782. doi: http://doi.acm.org/10.1145/362342.362367. URL http://portal.acm.org/citation.cfm?id=362342.362367.

[10] Y. Bulatov. Tree decomposition package, 2011. URL http://mathematica-bits.blogspot.com/2011/01/tree-decomposition-package.html.

[11] W. L. Buntine. Operations for Learning with Graphical Models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994. URL citeseer.ist.psu.edu/6938.html.

[12] W. L. Buntine. Chain graphs for learning. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence*, pages 46–54. Morgan Kaufmann, 1995.

[13] Y. Cao and L. Petzold. Slow Scale Tau-leaping Method. *Computer Methods in Applied Mechanics and Engineering*, 197(43-44):3472–3479, 2008. ISSN 18792138. doi: 10.1016/j.cma.2008.02.024. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2753989&tool=pmcentr

[14] S. Chib and E. Greenberg. Understanding the {M}etropolis-{H}astings Algorithm. *The American Statistician*, (4):327–335.

[15] H. Conzelmann, J. Saez-Rodriguez, T. Sauter, B. N. Kholodenko, and E. D. Gilles. A domain-oriented approach to the reduction of combinatorial complexity in signal transduction networks. *BMC Bioinformatics*, 7(1):34, 2006. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1413560&tool=pmcentr

[16] H. Conzelmann, D. Fey, and E. D. Gilles. Exact model reduction of combinatorial reaction networks. *BMC systems biology*, 2(1):78, 2008. URL http://dx.doi.org/10.1186/1752-0509-2-78.

[17] C. W. Cotman, D. T. Monaghan, O. P. Ottersen, and J. Storm Mathisen. Anatomical organization of excitatory amino acid receptors and their pathways. *Trends in Neuroscience*, 10:273–280, 1987.

[18] A. Destexhe, Z. F. Mainen, and T. J. Sejnowski. Synthesis of models for excitable membranes, synaptic transmission and neuromodulation using a common kinetic formalism. *Journal of Computational Neuroscience*, 1(3):195–230, 1994. URL http://www.ncbi.nlm.nih.gov/pubmed/8792231.

[19] R. L. Dobruschin. The Description of a Random Field by Means of Conditional Probabilities and Conditions of Its Regularity. *Theory of Probability and its Applications*, 13(2):197–224, 1968.

[20] K. M. Franks and T. J. Sejnowski. Complexity of calcium signaling in synaptic spines. *BioEssays*, 24(12):1130–1144, 2002. URL http://www.ncbi.nlm.nih.gov/pubmed/12447978.

[21] B. Frey. Extending Factor Graphs so as to Unify Directed and Undirected Graphical Models. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 226–257, San Francisco, CA, 2003. Morgan Kaufmann.

[22] J. Friedman, T. Hastie, and R. Tibshirani. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal Of Statistical Software*, 33(1):1–22, 2010. ISSN 15487660. URL `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2929880&tool=pmcentr`

[23] S. Geman and D. Geman. Stochastic relaxation, {G}ibbs distributions, and the {B}ayesian restoration of images. {*IEEE*} *Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[24] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*. Springer-Verlag, 2001.

[25] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977. ISSN 00223654. doi: 10.1021/j100540a008. URL `http://pubs.acs.org/doi/abs/10.1021/j100540a008`.

[26] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58(1):35–55, 2007. ISSN 0066426X. doi: 10.1146/annurev.physchem.58.032806.104637. URL `http://www.ncbi.nlm.nih.gov/pubmed/17037977`.

[27] S. D. Guikema and J. P. Goffelt. A Flexible Count Data Regression Model for Risk Analysis. *Risk Analysis*, 28(1):213–223, Feb. 2008.

[28] W. K. Hastings. {M}onte {C}arlo sampling methods using {M}arkov chains and their applications. *Biometrika*, 57(1):97–109, Apr. 1970. doi: 10.1093/biomet/57.1.97.

[29] D. Heckerman. A Tutorial on Learning with Bayesian Networks. Technical report, Microsoft Research, Redmond, Washington, Mar. 1995. URL `http://citeseer.ist.psu.edu/41127.html`.

[30] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[31] D. Heckerman, C. Meek, and D. Koller. Probabilistic Entity-Relationship Models, PRMs, and Plate Models. In *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields*, Banff, Canada, 2004.

[32] F. Horn and R. Jackson. General mass action kinetics. *Archive for Rational Mechanics and Analysis*, 47(2):81–116, 1972. ISSN 00039527. doi: 10.1007/BF00251225. URL http://www.springerlink.com/index/10.1007/BF00251225.

[33] V. Isham. An introduction to spatial point processes and Markov random fields. *International Statistics Review*, 49:21–43, 1981.

[34] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4(4):269–282, 1990. URL http://www.citeulike.org/user/wasteland93/article/854502.

[35] H. Jönsson, M. G. Heisler, B. E. Shapiro, E. M. Meyerowitz, and A. Author. An auxin-driven polarized transport model for phyllotaxis. *Proceedings of the National Academy of Sciences*, 103(5):1633–1638, Jan. 2006. ISSN 0027-8424. doi: 10.1073/pnas.0509839103. URL http://dx.doi.org/10.1073/pnas.0509839103.

[36] J. B. Kadane, G. Shmueli, T. P. Minka, S. Borle, and P. Boatwright. Conjugate Analysis of the {C}onway-{M}axwell-{P}oisson Distribution. *Bayesian Analysis*, 1(3):363–374, 2006.

[37] S. Kakiuchi, S. Yasuda, R. Yamazaki, Y. Teshima, K. Kanda, R. Kakiuchi, and K. Sobue. Quantitative determinations of calmodulin in the supernatant and particulate fractions of mammalian tissues. *Journal of Biochemistry*, 92(4):1041–1048, 1982. URL http://www.ncbi.nlm.nih.gov/pubmed/7174634.

[38] M. B. Kennedy, H. C. Beale, H. J. Carlisle, and L. R. Washburn. Integration of biochemical signalling in spines. *Nature Reviews Neuroscience*, 6(6):423–434, 2005. URL http://www.ncbi.nlm.nih.gov/pubmed/15928715.

[39] R. A. Kerr, T. M. Bartol, B. Kaminsky, M. Dittrich, J.-C. J. Chang, S. B. Baden, T. J. Sejnowski, and J. R. Stiles. FAST MONTE CARLO SIMULATION METHODS FOR BIOLOGICAL REACTION-DIFFUSION SYSTEMS IN SOLUTION AND ON SURFACES. *SIAM journal on scientific computing a publication of the Society for Industrial and Applied Mathematics*, 30(6):3126, 2008. ISSN 10957197. doi: 10.1137/070692017. URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=Abst

[40] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques (Adaptive Computation and Machine Learning)*. The MIT Press, 2009. ISBN 0262013193. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/026

[41] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47(2):498–520, Feb. 2001. URL http://citeseer.ist.psu.edu/kschischang01factor.html.

[42] S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996. ISBN 0-19-852219-3.

[43] S. L. Lauritzen and D. J. Spiegelhalter. 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. ISSN 00359246. URL http://www.jstor.org/stable/2345762.

[44] M. Leisink and B. Kappen. Bound Propagation. *Journal of Artificial Intelligence Research*, 19:139–154, 2003.

[45] J. Lisman, H. Schulman, and H. Cline. The molecular basis of CaMKII function in synaptic and behavioural memory. *Nature Reviews Neuroscience*, 3(3):175–190, 2002. ISSN 1471003X. doi: 10.1038/nrn753. URL http://www.ncbi.nlm.nih.gov/pubmed/11994750.

[46] D. Lord, S. D. Guikema, and S. R. Geedipally. Application of the Conway-MaxwellPoisson generalized linear model for analyzing motor vehicle crashes. *Accident Analysis & Prevention*, 40(3):1123–1134, May 2008.

[47] D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter. WinBUGS – A Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, 10(4):325–337, 2000. ISSN 0960-3174. doi: http://dx.doi.org/10.1023/A:1008929526011.

[48] B. Milch and S. Russell. First-order probabilistic languages: Into the unknown. *Inductive Logic Programming*, pages 10–24, 2007. doi: 10.1007/978-3-540-73847-3_3. URL http://portal.acm.org/citation.cfm?id=1417648.

[49] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. {BLOG}: Probabilistic Models with Unknown Objects. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, Aug. 2007.

[50] E. Mjolsness. Labeled graph notations for graphical models. Technical Report ICS 04-03, University of California, Irvine, Mar. 2004.

[51] E. Mjolsness. Variable-Structure Systems from Graphs and Grammars. Technical Report ICS 05-09, University of California, Irvine, Oct. 2005.

[52] E. Mjolsness and G. Yosiphon. Stochastic process semantics for dynamical grammars. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):329–395, Aug. 2006. ISSN 1012-2443. doi: 10.1007/s10472-006-9034-1. URL http://dx.doi.org/10.1007/s10472-006-9034-1.

[53] E. Mjolsness, D. Orendorff, P. Chatelain, and P. Koumoutsakos. An exact accelerated stochastic simulation algorithm.

The Journal of chemical physics, 130(14):144110, 2009. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2852436&tool=pmcentr

[54] E. Mjolsness, P. Baldi, and B. E. Shapiro. *Systems Biology: Computational and Mathematical Methods, in preparation*. 2012.

[55] J. Monod, J. Wyman, and J. P. Changeux. ON THE NATURE OF ALLOSTERIC TRANSITIONS: A PLAUSIBLE MODEL. *Journal of Molecular Biology*, 12(December):88–118, 1965. URL http://www.ncbi.nlm.nih.gov/pubmed/14343300.

[56] J. M. Mooij and H. J. Kappen. Loop Corrections for Approximate Inference on Factor Graphs. *Journal of Machine Learning Research*, pages 1113–1143, May 2007.

[57] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 447–467, San Francisco, CA, 1999. Morgan Kaufmann.

[58] J. Neville and D. Jensen. Relational Dependency Networks. *Journal of Machine Learning Research*, 8:653–692, 2007. ISSN 1533-7928.

[59] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167, 2003. URL http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0303516.

[60] U. Nodelman, C. R. Shelton, and D. Koller. Continuous Time Bayesian Networks. *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, (June):378–387, 2002. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.1813&rep=rep1&

[61] V. Pavlovic, J. M. Rehg, and K. P. Murphy. A dynamic Bayesian network approach to figure tracking using learned dynamic models. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1(Iccv 99):94–101 vol.1, 1999. doi: 10.1109/ICCV.1999.791203. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=791203.

[62] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7.

[63] S. Pepke, T. Kinzer-Ursem, S. Mihalas, and M. B. Kennedy. A Dynamic Model of Interactions of Ca2+, Calmodulin, and Catalytic Subunits of Ca2+/Calmodulin-Dependent Protein Kinase II. *PLoS Computational Biology*, 6(2):15, 2010. URL http://www.ncbi.nlm.nih.gov/pubmed/20168991.

[64] C. Peterson and B. Soderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1:3–22, 1989.

[65] H. J. Pi, N. Otmakhov, D. Lemelin, P. De Koninck, and J. Lisman. Autonomous CaMKII can promote either long-term potentiation or long-term depression, depending on the state of T305/T306 phosphorylation. *Journal of Neuroscience*, 30 (26):8704–8709, 2010. URL http://www.ncbi.nlm.nih.gov/pubmed/20592192.

[66] N. Pržulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004. ISSN 1367-4803. doi: 10.1093/bioinformatics/bth436. URL http://dx.doi.org/10.1093/bioinformatics/bth436.

[67] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006. ISSN 0885-6125. doi: http://dx.doi.org/10.1007/s10994-006-5833-1.

[68] O. S. Rosenberg, S. Deindl, R.-J. Sung, A. C. Nairn, and J. Kuriyan. Structure of the autoinhibited kinase domain of CaMKII and SAXS analysis of the holoenzyme. *Cell*, 123(5):849–860, 2005. URL http://www.ncbi.nlm.nih.gov/pubmed/16325579.

[69] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. URL http://www.ncbi.nlm.nih.gov/pubmed/19342586.

[70] G. Shmueli, T. P. Minka, J. B. Kadane, S. Borle, and P. Boatwright. A useful distribution for fitting discrete data: revival of the {C}onway-{M}axwell-{P}oisson distribution. *Journal Of The Royal Statistical Society Series C*, 54(1):127–142, 2005. URL http://ideas.repec.org/a/bla/jorssc/v54y2005i1p127-142.html.

[71] T. R. Soderling. The Ca2+calmodulin-dependent protein kinase cascade. *Trends in Biochemical Sciences*, 24(6):232–236, 1999. ISSN 09680004. doi: 10.1016/S0968-0004(99)01383-3. URL http://dx.doi.org/10.1016/S0968-0004(99)01383-3.

[72] T. R. Soderling, B. Chang, and D. Brickey. Cellular signaling through multifunctional Ca2+/calmodulin-dependent protein kinase II. *The Journal of Biological Chemistry*, 276(6):3719–3722, 2001. URL http://www.ncbi.nlm.nih.gov/pubmed/11096120.

[73] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, Sept. 1981. ISSN 0018-9448.

[74] N. G. Van Kampen. *Stochastic Processes in Physics and Chemistry*, volume 11 of *North-Holland personal library*. North-Holland, 1992. ISBN 0444893490. doi: 10.2307/2984076. URL http://www.jstor.org/stable/2984076.

[75] A. V. Werhli and D. Husmeier. Reconstructing Gene Regulatory Networks with Bayesian Networks by Combining Expression Data with Multiple Sources of Prior

Knowledge. *Statistical Applications in Genetics and Molecular Biology*, 6(1), 2007. URL `http://www.bepress.com/sagmb/vol6/iss1/art15`.

[76] G. Yosiphon. *Stochastic Parameterized Grammars: Formalization, Inference and Modeling Applications*. PhD thesis, University of California, Irvine, 2009.

Appendices

A Derivation of Method One

Beginning with the definition of KL divergence, $\mathcal{D}_{\mathcal{KL}}(p\|\tilde{p}) = -p\log(\tilde{p}/p)$ (or equivalently, $p\log(p/\tilde{p})$). As in the derivation of method two, we will first differentiate with respect to μ, and then with respect to t.

From the definition of the Markov random field,

$$\begin{aligned}\mathcal{D}_{\mathcal{KL}}(p\|\tilde{p}) &= \int p(x;t)\log\left[p(x;t)\,/\tilde{p}(x;\mu(t))\right]dx \\ &= \int p(x;t)\left(\left(\sum_{\beta=1}^{\text{cliques}}\mu_\beta(t)V_\beta(x)\right) + \log[Z(\mu(t))] + \log[p(x;t)]\right)dx.\end{aligned}$$

Now we can begin evaluating the derivative $\partial\mathcal{D}_{\mathcal{KL}}(p\|\tilde{p})/\partial\mu_\alpha$. The derivative of the sum over cliques is clear, and the true distribution p does not depend on μ, therefore

$$\begin{aligned}\frac{\partial}{\partial\mu_\alpha}\mathcal{D}_{\mathcal{KL}}(p\|\tilde{p}) &= \int p(x;t)\frac{\partial}{\partial\mu_\alpha}\left(\sum_{\beta=1}^{\text{cliques}}\mu_\beta(t)V_\beta(x) + \log[Z(\mu(t))] + \log[p(x;t)]\right)dx \\ &= \int p(x;t)\frac{\partial}{\partial\mu_\alpha}\left(\sum_{\beta=1}^{\text{cliques}}\mu_\beta(t)V_\beta(x) + \log[Z(\mu(t))]\right)dx \\ &= \int p(x;\mu(t))\left(V_\alpha(x) + \frac{\partial}{\partial\mu_\alpha}\log[Z(\mu(t))]\right)dx.\end{aligned}$$

Referring once again to Equation 5.2, $\partial \log Z(\mu(t))/\partial \mu_\alpha = -\langle V_\alpha \rangle$, so

$$\frac{\partial}{\partial \mu_\alpha} \mathcal{D}_{\mathcal{KL}}(p \| \tilde{p}) = \int p(x;t) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) dx$$

.

Differentiating this with respect to time, we first apply the chain rule, and then integrate the rightmost term.

$$\begin{aligned}
\frac{\partial}{\partial t} \mathcal{D}_{\mathcal{KL}} &= \frac{\partial}{\partial t} \left(\int p(x;t) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) dx \right) \\
&= \int \left(\frac{\partial}{\partial t} p(x;t) \right) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) dx - \int p(x;t) \frac{\partial}{\partial t} \langle V_\alpha \rangle dx
\end{aligned}$$

Applying the master equation, and applying the result for $\partial/\partial t \, \langle V_\alpha \rangle$ first used in Equation (6.10),

$$\begin{aligned}
\frac{\partial}{\partial t} \mathcal{D}_{\mathcal{KL}} &= \int \left([W \cdot p(;t)](x) \right) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) dx - \frac{\partial}{\partial t} \langle V_\alpha \rangle \\
&= \int \left([W \cdot p(;t)](x) \right) \left(V_\alpha(x) - \langle V_\alpha \rangle \right) dx - \sum_{\beta=1}^{\text{cliques}} \frac{\partial \mu_\beta(t)}{\partial t} \langle V_\alpha \left(\langle V_\beta \rangle - V_\beta \right) \rangle
\end{aligned}$$

Returning to the Δ notation introduced in Section 6.1, so that $\Delta x = x - \langle x \rangle$, and then assuming linear dynamics as in Equation (5.3),

$$\begin{aligned}
\frac{\partial}{\partial t} \mathcal{D}_{\mathcal{KL}} &= \int \left([W \cdot p(;t)](x) \right) \left(\Delta V_\alpha(x) \right) dx + \sum_{\beta=1}^{\text{cliques}} \frac{\partial \mu_\beta(t)}{\partial t} \langle V_\alpha \Delta V_\beta \rangle \\
&= \int \left([W \cdot p(;t)](x) \right) \left(\Delta V_\alpha(x) \right) dx + \sum_{\beta=1}^{\text{cliques}} f_\beta(\mu(t)) \langle V_\alpha \Delta V_\beta \rangle \\
&= \int \left([W \cdot p(;t)](x) \right) \left(\Delta V_\alpha(x) \right) dx + \sum_{\beta=1}^{\text{cliques}} \sum_{A=1}^{\text{bases}} f_{\beta,A}(\mu(t)) \theta_A \langle V_\alpha \Delta V_\beta \rangle
\end{aligned}$$

Setting this equal to zero, and moving terms with the rate matrix across the equals sign, we have

$$\int ([W \cdot p(;t)](x))\,(\Delta V_\alpha(x))\,dx = -\sum_{A=1}^{\text{bases}} \sum_{\beta=1}^{\text{cliques}} f_{\beta,A}(\mu(t))\theta_A \langle V_\alpha \Delta V_\beta \rangle \tag{A.1}$$

Now define a vector $\mathbf{B}$ with α entries

$$\mathbf{B}_\alpha = \int ([W \cdot p(;t)](x))\,(\Delta V_\alpha(x))\,dx$$

and an A-by-α matrix $\mathbf{A}$ with entries

$$\mathbf{A}_{\alpha,A} = -\sum_{\beta=1}^{\text{cliques}} f_{\beta,A}(\mu(t)) \langle V_\alpha \Delta V_\beta \rangle$$

Then $\mathbf{B}= \mathbf{A}\theta$, and we have a second algorithm for minimizing the KL divergence by solving a system of linear equations.

B Dependency Diagram Implementation: Finite Difference Engine

The following implements a finite difference engine used for simplifying the acceptance probability ratio for automatically generated Metropolis sampling algorithms. It is capable of simplifying a considerably wider range of differences than the symbolic algebra system built into *Mathematica*.

The two parts of this system are the functions `Diff` and `Simp`. `Diff` calculates the change in an expression in terms of a difference `Ch[x]` in a single changed variable or sub-expression `x`. `Simp` simplifies algebraic expressions, with particular attention paid

to putting them into forms which `Diff` is capable of recognizing and manipulating.

```
SimpDiff[e_,v_] :=
    Module[{s,d},
           s = Simp[Unevaluated[e]];
           d = Diff[s,v];
           d
          ];

Diff[e1_, v_] :=
    (0)/; MyFreeQ[e1,v];
Diff[v_, v_] :=
    Ch[v];
Diff[w_?NumberQ, v_] :=
    0;
Diff[w_?AtomQ, v_?AtomQ] :=
    If[ MatchQ[w,v],
        Ch[v],
        0
      ];
Diff[Times[e1_,e2_], v_] :=
    Module[{e1d, e2d},
           e1d = SimpDiff[e1,v];
           e2d = SimpDiff[e2,v];
           (e1*e2d) + (e2*e1d) + (e1d*e2d)
          ];
Diff[e1s_Plus, v_] :=
    Module[{tmp},
           tmp = PlusToList[e1s];
           Plus[ListToSequence[Map[SimpDiff[#,v]&, tmp]]]
          ];
Diff[Product[e1_, {e2_,min_,max_}], v_] :=
    SimpDiff[E^(Log[Product[e1,{e2,min,max}]]), v];
Diff[sum[e_,{it_,min_,max_}], v_] :=
    Module[{d,di=0,dj=0},
           d = SimpDiff[e,v];
           Module[{minch = 0,maxch=0},
                  If[!MatchQ[Head[min],ListLength] && !MyFreeQ[min,v],
                     minch = SimpDiff[min,v];
                    ];
                  If[!MatchQ[Head[max],ListLength] && !MyFreeQ[max,v],
                     maxch = SimpDiff[max,v];
                    ];
```

```
                    di =  ((If[maxch > 0,1,0]*
                            sum[d + e, Evaluate[{it,max+1,max+maxch}]]) -
                           (If[-maxch > 0,1,0]*
                            sum[d+e,Evaluate[{it,max+maxch+1,max}]]));
                    dj = ((-If[minch > 0,1,0]*
                           sum[d + e, Evaluate[{it,min,min+minch-1}]]) +
                          (If[-minch > 0,1,0]*
                           sum[d+e,Evaluate[{it,min+minch,min-1}]]));
                  ];
             Simp[Evaluate[sum[d, {it,min,max}]]] + di + dj
            ];
Diff[e_, h_[e_,k__]] :=
    Ch[e] /; MatchQ[h,Part] || MatchQ[h, Subscript];
Diff[h_[e_,i_], h_[e_,k_]] :=
    h[Ch[e],k]*KroneckerDelta[i,k] /; MatchQ[h,Part] || MatchQ[h, Subscript];
Diff[h_[e_,i__], h_[e_,k__]] :=
    Module[ {len,ii,kk,u,uu},
            len = Min[Length[{k}],Length[{i}]];
            ii = Part[{i},1;;len];
            kk = Part[{k},1;;len];
            Which[
         Length[{k}]==Length[{i}],
     u = {k},
        Length[{k}]>Length[{i}],
        u = kk,
         Length[{k}]<Length[{i}],
         u = Join[kk,Part[{i},len+1;;]]
                ];
            uu = ListToSequence[u];
            h[Ch[e],uu]*Fold[Times,1,MapThread[KroneckerDelta[#1,#2]&,{ii,kk}]]
          ] /; MatchQ[h,Part]||MatchQ[h, Subscript];
Diff[h_[e1_?AtomQ, i__], h_[e2_?AtomQ, k__]] :=
    0 /; (!MatchQ[e1,e2] && (MatchQ[h,Part]||MatchQ[h, Subscript]));
Diff[h_[e1_, i__],e2_] :=
    Module[{foo},
           foo = SimpDiff[e1,e2];
           (* Print[foo, " " , h[foo,i], "f"];*)
           If[Union[Map[IntegerQ,{i}]]=={True},
              Hold[h][foo,i],
              h[foo,i]
             ]
          ] /; !MyFreeQ[e1,e2] && !MatchQ[Head[e1],DDList]
                               && (MatchQ[h,Part]||MatchQ[h, Subscript]);
Diff[Part[DDList[e_,l_], i__], h_[e_,k__]] :=
    Module[ {indexDeltas,il = {},ii,kk,ll},
```

```
            ii = {i};
            kk = {k};
            ll = l;
            indexDeltas = MapThread[(Which[
                ListQ[#1] && ii!={},
                AppendTo[il,First[ii]];
                ii = Rest[ii],
                !ListQ[#1],
                AppendTo[il,#1]
                                        ];
                                    KroneckerDelta[Last[il],#2])&,
                                   {ll,kk}
                                  ];
             Part[Ch[e], ListToSequence[il]] * Times[ListToSequence[indexDeltas]]
          ] /; MatchQ[h,Part]||MatchQ[h,Subscript];
Diff[Part[DDList[e1_,l_],i__], h_[e2_,k__]] :=
    0 /; (MatchQ[h,Part] || MatchQ[h,Subscript]) && !MatchQ[e1,e2];
Diff[Part[list_,spec__], v_] :=
    Module[{specd, newspec},
           specd = Map[SimpDiff[#,v]&,{spec}];
           newspec = MapThread[#1 + #2&, {{spec},specd}];
           Part[list,ListToSequence[newspec]] - Part[list,spec]
          ] /; MyFreeQ[list,v] && !MyFreeQ[{spec},v];
Diff[Log[h_[e_,i_]], h_[e_,k_]] :=
    (Log[h[e,k]+h[Ch[e],k]]-Log[h[e,k]])*KroneckerDelta[i,k]
        /; MatchQ[h,Part]||MatchQ[h, Subscript];
Diff[Log[Factorial[e1_]], v_] :=
    Module[{d,it,check},
           d = SimpDiff[e1, v];
           If[!MatchQ[d,0],
              it = IteratorFor[e1];
              check = If[d>0,1,0];
              ((check*2)-1)sum[Log[it],
                               Evaluate[{it,
                                         (e1+1+(1-check)d),
                                         (e1+(check*d))}]],
              (* else *)
              0
             ]
          ];
Diff[Log[e_], v_] :=
    Log[e + SimpDiff[e,v]] - Log[e];
Diff[Pochhammer[e1_,e2_], v_] :=
    SimpDiff[Factorial[e1+e2-1]/Factorial[e1-1],v];
Diff[Factorial[e1_], v_] :=
```

```
    Module[{d,it,check},
           d = SimpDiff[e1, v];
           it = IteratorFor[e1];
           check = If[d>0,1,0];
           If[!MatchQ[d,0],
              check(Factorial[e1](Product[it,Evaluate[{it,e1+1,e1+d}]]-1))
                + ((1-check)(Factorial[e1+d]
                             *(1-Product[it,Evaluate[{it,e1+1+d,e1}]]))),
              0
             ]
          ];
Diff[Power[e1_, e3_?IntegerQ], v_] :=
    Module[{tmp},
           tmp = SimpDiff[Power[e1,Abs[e3]],v];
           -tmp / (Power[e1,2*Abs[e3]] + (tmp * Power[e1,Abs[e3]]))
          ] /; e3 < 0;
Diff[Power[e1_, e3_?IntegerQ], v_] :=
    Module[{tmp},
           tmp = SimpDiff[e1,v];
           Sum[Binomial[e3,iterz]*Power[e1,e3-iterz]*Power[tmp,iterz],
             {iterz, 1, e3}]
          ];
Diff[Power[e1_, e2_],v_] :=
    (Power[e1, SimpDiff[e2,v]]-1)Power[e1, e2] /; MyFreeQ[e1,v]
                                                  && !MyFreeQ[e2,v];
Diff[Power[e1_,e2_],v_] :=
    Module[{tmp},
           tmp = SimpDiff[e1,v];
           Power[(e1+tmp),e2] * Power[(e1+tmp), SimpDiff[e2,v]] - Power[e1,e2]
          ];
Diff[If[Greater[e1_, 0], 1, 0], v_] :=
    Module[{d,e},
           d = (If[e1+SimpDiff[e1,v] > 0, 1, 0] -
                If[e1 > 0, 1, 0]);
           e = Simp[Evaluate[d]];
           e
          ];
Diff[DDList[e1_, e2_], e3_[e1_,e4__]] :=
    Module[ {indexDeltas},
            indexDeltas = MapThread[If[ MatchQ[#1,_List],
                                        1,
                                        KroneckerDelta[#1,#2]
                                      ]&,
                                    {e2, {e4}}
                                   ];
```

```
            Part[Ch[e1], ListToSequence[IndListToPSpec[e2]]]
               * Times[ListToSequence[indexDeltas]]
          ] /; MatchQ[e3,Part] || MatchQ[e3,Subscript];
Diff[Abs[e1_], e2_] :=
    Module[{tmp},
           tmp = SimpDiff[e1,e2];
           -tmp - 2*e1*If[e1 > 0, 1, 0] + 2*(e1 + tmp) If[e1 + tmp > 0, 1, 0]
          ];
Diff[head_[args__], v_] :=
    Module[ {newargs,seq},
            newargs = Map[(# + SimpDiff[#,v])&, {args}];
            seq = ListToSequence[newargs];
            (* I hope this doesn't break anything? *)
            If[Intersection[Attributes[head],{HoldFirst,Hold,HoldAll}]=={},
               head[seq] - head[args],
               head[Evaluate[seq]] - head[args]
              ]
          ];(* /; !MatchQ[head,ListLength];*)
Diff[e1_, v_] :=
    (cd[e1,v]);

On[Part::pspec];

Simp[a_?AtomQ] := a;
Simp[N[e_]] := Simp[e];
Simp[Log[Product[e1_, e2__]]] :=
    Simp[sum[Log[e1], e2]];
Simp[Log[Power[E, e1_]]] :=
    Simp[e1];
Simp[Log[e1_, Power[e1_, e2_]]] :=
    Simp[e2];
Simp[Log[es_Times]] :=
    Module[{tmp},
           tmp = TimesToList[es];
           Plus[ListToSequence[Map[Simp[Evaluate[Log[#]]]&,tmp]]]
          ];
Simp[Log[Power[e1_, e2_]]] :=
    Simp[e2] * Simp[Log[e1]];
Simp[Log[e1s_Times + e2_]] :=
    Module[ {heads,list,ks,ksl={},e1l,e1,s,exp},
            ksl = Cases[TimesToList[e1s],_KroneckerDelta];
            e1l = Complement[TimesToList[e1s],ksl];
            ks = Times[ListToSequence[ksl]];
            e1 = Times[ListToSequence[e1l]];
            If[ ksl!={},
```

```
          s = Log[e2] + ks(Log[e2 + e1]-Log[e2]);
          Simp[Evaluate[s]],
                 exp = Expand[e1s];
                 If[MatchQ[Head[exp],Times],
                    Log[Simp[e1s + e2]],
                    Simp[Evaluate[Log[exp+e2]]]
                   ]
               ]
           ] /; !FreeQ[{e1s},KroneckerDelta];
Simp[Log[e1_]] :=
    Log[Simp[e1]];
Simp[Product[e1_,e2_]] :=
    Product[Simp[e1],e2];
Simp[Product[e1_,e2s__]] :=
    Module[ {e},
            e = Fold[Product[#1,#2]&,
                     Product[e1,Evaluate[First[Reverse[{e2s}]]]],
                     Rest[Reverse[{e2s}]]];
            Simp[Evaluate[e]]
          ];
Simp[Power[Power[e1_, e2_], e3_]] :=
    Simp[Power[e1,e2*e3]];
Simp[Power[e1_+e2_,i_?IntegerQ]] :=
    Module[{ssum},
           ssum = Sum[Binomial[i, iterz]*Power[e1,i-iterz]*Power[e2,iterz],
                     {iterz, 0, i}];
           Simp[Evaluate[ssum]]
          ] /; i>0;
Simp[Power[KroneckerDelta[e1_, e2_], e3_]] :=
    KroneckerDelta[e1,e2] /; e3 > 0;
Simp[Power[e1s_Times + e2_ ,pow_?NumberQ]] :=
    Module[ {heads,list,ks,ksl={},e1l,e1,s,exp},
            ksl = Cases[TimesToList[e1s],_KroneckerDelta];
            e1l = Complement[TimesToList[e1s],ksl];
            ks = Times[ListToSequence[ksl]];
            e1 = Times[ListToSequence[e1l]];
            If[ ksl!={},
          s = Power[e2,pow] + ks(Power[e2 + e1,pow]-Power[e2,pow]);
          Simp[Evaluate[s]],
                 exp = Expand[e1s];
                 If[MatchQ[Head[exp],Times],
                    Power[Simp[e1s + e2],pow],
                    Simp[Evaluate[Power[exp+e2,pow]]]
                   ]
               ]
```

```
          ] /; !FreeQ[{e1s},KroneckerDelta];
Simp[Power[e1_, e2_]] :=
    Simp[e1]^Simp[e2];
Simp[sum[e1_, e2s__List]] :=
    Module[ {e},
            e = Fold[sum[#1,#2]&,
                     sum[e1, Evaluate[First[Reverse[{e2s}]]]],
                     Rest[Reverse[{e2s}]]];
            Simp[Evaluate[e]]
          ] /; Length[{e2s}] > 1;
Simp[sum[es_Plus, {e3_,it1_,it2_}]] :=
    Module[{each, sech,list},
           list = PlusToList[es];
           each = Map[sum[#, {e3,it1,it2}]&, list];
           sech = Map[Simp, each];
           Plus[ListToSequence[sech]]
          ];
Simp[sum[KroneckerDelta[e1_, e2_], {e4_,min_,max_}]] :=
    Simp[sum[Times[1,KroneckerDelta[e1, e2]], {e4,min,max}]];
Simp[sum[KroneckerDelta[e1_, e2_] * e3_, {e4_,min_,max_}]] :=
    Module[ {p,q,r,s},
            Which[
        MatchQ[e1,e4],
                r = ReplaceAll[e3, e1->e2];
         Simp[Evaluate[r]],
         MatchQ[e2,e4],
        r = ReplaceAll[e3, e2->e1];
                Simp[Evaluate[r]],
         True,
         p = Position[Unevaluated[e1],Unevaluated[e4]];
         q = Position[Unevaluated[e2],Unevaluated[e4]];
        Which[
                    p !=  {} && q !=  {},
                    sum[Simp[KroneckerDelta[e1,e2]*e3],{e4,min,max}],
                    p!=  {} || q!= {},
                    r = Solve[e1== e2,e4];
                    s = ReplaceAll[e3,r[[1,1]]];
                    Simp[Evaluate[s]]
                     ]
                ]
          ] /; Position[Unevaluated[e1],Unevaluated[e4]]!={}
                || Position[Unevaluated[e2],Unevaluated[e4]]!={};
Simp[sum[e1_ * e2_, {e3_,it1_,it2_}]] :=
    Module[ {r},
            r = Simp[sum[e1,{e3,it1,it2}]];
```

```
            Simp[Evaluate[e2*r]]
          ] /; MyFreeQ[e2,e3];

Simp[sum[Times[e1__], {e2_,min_,max_}]] :=
    Module[ {ex},
            ex = Expand[Times[e1]];
            If[ !MatchQ[Head[ex],Times],
                Simp[Evaluate[sum[ex, {e2,min,max}]]],
                sum[Simp[Times[e1]],{e2,min,max}]
              ]
          ];
Simp[sum[e1_, {e2_,min_,max_}]] :=
    Module[{r,pass,parts,nums,nmz,RepNum,newr,sss,BackNum},
           r = Simp[e1];
           If[SameQ[r,e1],
              sum[r, {e2,min,max}],
              (* this next hunk is an attempt to work around a Mathematica bug
                 where sometimes symbolic sums with "numbers" in them cause
                 infinite recursion. Seriously.
                 It's been documented since at least 2005:
               http://forums.wolfram.com/mathgroup/archive/2005/Sep/msg00686.html

                 Note: this bug appears to be fixed in M7, but leaving this in
                 for now.
                 *)
               parts = Position[r, piece_ /; NumberQ[piece] && !IntegerQ[piece]];
               nums  = Map[Part[r, ListToSequence[#]] &, parts];
               nmz = {};
               RepNum[exp_, pos_] :=
               Module[{nm},
                      nm = SymbolNotInExpression[r, "n"];
                      AppendTo[nmz, nm];
                      ReplacePart[exp, pos -> nm]
                     ];
               newr = Fold[RepNum, r, parts];
               sss = sum[newr, Evaluate[{e2,min,max}]];
               BackNum[exp_, {rep_, nr_}] :=
               Module[{pos},
                      pos = Position[exp, rep];
                      ReplacePart[exp, pos -> nr]
                     ];
               pass = Fold[BackNum, sss, MapThread[{#1, #2} &, {nmz, nums}]];
               (* end hack around Mathematica sucking. *)

               Simp[Evaluate[pass]]
```

```
            ]
          ] ;
Simp[If[e1_*KroneckerDelta[e2_,e3_]>0, 1, 0]] :=
    KroneckerDelta[e2,e3]*Simp[If[e1>0, 1, 0 ]];
Simp[If[h1_[Ch[e1_], e2__]*ks_ + h2_[e1_,e3__] > 0, 1, 0] -
      If[h2_[e1_,e3__]>0, 1, 0]] :=
    Module[ {heads,list},
            If[ MatchQ[Head[ks],Times],
                list = ReplacePart[ks,0->List],
                list = {ks}
              ];
            heads = Union[Map[Head,list]];
            If[ MatchQ[heads,{KroneckerDelta}],
                (If[h1[Ch[e1],e2] + h2[e1,e3] > 0, 1, 0] -
                 If[h2[e1,e3] > 0, 1, 0]) * ks,
                If[h1[Ch[e1],e2]*ks + h2[e1,e3] > 0, 1, 0] -
                If[h2[e1,e3] > 0, 1, 0]
              ]
          ] /; ((MatchQ[h1,Part]||MatchQ[h1, Subscript])
                 && (MatchQ[h2,Part]||MatchQ[h2, Subscript]));
Simp[If[Times[e1s__] + e2_ > 0, 1, 0]] :=
    Module[ {heads,list,ks,ksl={},e1l,e1,s},
            ksl = Cases[{e1s},_KroneckerDelta];
            e1l = Complement[{e1s},ksl];
            ks = Times[ListToSequence[ksl]];
            e1 = Times[ListToSequence[e1l]];
            If[ ksl!={},
          s = Heaviside[e2] + ks(Heaviside[e2 + e1]-Heaviside[e2]);
          Simp[Evaluate[s]],
          Heaviside[Simp[Times[e1s] + e2]]
               ]
          ] /; !FreeQ[{e1s},KroneckerDelta];
Simp[If[Times[-1,e1_]>0,1,0]] := Simp[1-Heaviside[e1]];
Simp[es_Plus] :=
    Module[{temp},
           temp = PlusToList[es];
           Plus[ListToSequence[Map[Simp[Evaluate[#]]&, temp]]]
          ];
Simp[KroneckerDelta[e1_,e2_]] :=
    Module[ {s},
            s = Quiet[Solve[e1==e2]];
            If[ s == {},
                0,
                KroneckerDelta[e1,e2]
              ]
```

```
          ];
Simp[es_Times] :=
    Module[{tmp},
           tmp = TimesToList[es];
           Times[ListToSequence[Map[Simp,tmp]]]
          ];
Simp[Abs[e1_*KroneckerDelta[e2_,e3_]]] :=
    Abs[e1]*KroneckerDelta[e2,e3];
Simp[Abs[e1_*e2_]] :=
    Simp[Abs[e1]]*Simp[Abs[e2]];
Simp[Abs[KroneckerDelta[e2_,e3_]]] :=
    KroneckerDelta[e2,e3];
Simp[Abs[KroneckerDelta[i_,y_]-KroneckerDelta[j_,y_]]] :=
    KroneckerDelta[i,y] + KroneckerDelta[j,y]
        - (2KroneckerDelta[i,j]KroneckerDelta[i,y]KroneckerDelta[j,y]);
Simp[Abs[e1_Plus]] :=
    Module[ {fac},
            fac = Factor[e1];
            If[ MatchQ[Head[fac],Plus],
                Abs[Simp[Plus[e1]]],
                Simp[Evaluate[Abs[fac]]]
              ]
          ];
Simp[Abs[e1_]] :=
    Abs[Simp[e1]];
Simp[h_[h_[e1_,e2__],e3__]] :=
    Simp[h[e1,e2,e3]] /; MatchQ[h,Part]||MatchQ[h,Subscript];
Simp[h_[Plus[e1_,e2_],e3__]] :=
    Simp[h[e1,e3]+h[e2,e3]] /; MatchQ[h,Part]||MatchQ[h,Subscript];
Simp[h_[Times[e1_?NumberQ,e2_],e3__]] :=
    Simp[e1*h[e2,e3]] /; MatchQ[h,Part]||MatchQ[h,Subscript];
Simp[Part[list_, spec__]] :=
    Module[ {heads,ks,ksl={},e1l,e1,s,exp,processone,processed,org,aft,kr},
       processone =
         Function[{arg},
           Module[{e1s,e2},
             If[MatchQ[arg, (e1sm_Times + e2m_)
                       /; !FreeQ[{e1sm},
                                 KroneckerDelta]
                  &&((e1s=e1sm)||True)&&((e2=e2m)||True)],
               ksl = Cases[TimesToList[e1s],_KroneckerDelta];
               e1l = Complement[TimesToList[e1s],ksl];
               ks = Times[ListToSequence[ksl]];
               e1 = Times[ListToSequence[e1l]];
               If[ ksl!={},
```

```
                  {e2,e1,ks},
                  {e2,e1s,1}
                ],
              {arg,0,1}
            ]
          ]
        ];
        processed = Map[processone,{spec}];
        org = Map[#[[1]]&,processed];
        aft = Map[#[[1]]+#[[2]]&,processed];
        kr = Map[#[[3]]&,processed];
        Part[list,ListToSequence[org]] +
          (Times[ListToSequence[kr]]
            *(Part[list,ListToSequence[aft]]-Part[list,ListToSequence[org]]))
    ] /; Cases[{spec}, (e1s_Times + e2_) /; !FreeQ[{e1s}, KroneckerDelta]]!={};
Simp[f_[h1_[Ch[e1_], e2__]*ks_ + h2_[e1_,e3__]] - f_[h2_[e1_,e3__]]] :=
   Module[ {heads,list},
           If[ MatchQ[Head[ks],Times],
               list = ReplacePart[ks,0->List],
               list = {ks}
             ];
           heads = Union[Map[Head,list]];
           If[ MatchQ[heads,{KroneckerDelta}],
               (f[h1[Ch[e1],e2] + h2[e1,e3] > 0, 1, 0] -
                f[h2[e1,e3] > 0, 1, 0]) * ks,
               f[h1[Ch[e1],e2]*ks + h2[e1,e3] > 0, 1, 0] -
               f[h2[e1,e3] > 0, 1, 0]
             ]
         ] /; ((MatchQ[h1,Part]||MatchQ[h1, Subscript])
                && (MatchQ[h2,Part]||MatchQ[h2, Subscript]));
Simp[e1_] := e1;
```

C GCCDdd: Significant Methods

This function computes the expected values and other terms involved in solving the system of equations necessary to optimize the parameters θ.

```
computeExpectations[mus_, trueParg_, Ww_, obs_:{}] :=
```

```
Module[{stats, singles, poDelt, WdotP, samples, compExp,
 singExp, deltas, deltStats, deltStatSamples, mylps, threaded,
     twos, lps, threes, wParts1, wParts2, method1twos, fromStates, toStates, t

       samples = getDistributionSamples[mus, obs];
       trueP = trueParg;

       compExp := computeExpectation[#, samples]&;

       stats = Table[makeStat[i], {i,1,Length[GVparams]}];

       singles = Map[Function[{state}, # /. state]&, stats];
       singles = Append[singles, Function[{state}, Log[trueP[state]]]];
       singExp = Map[compExp, singles];
       singExp = N[singExp];

       deltas  = MapThread[Function[{state},#1[state] - #2]&,
                           {singles, singExp}];
       deltas  = N[deltas];

       poDelt  = Most[deltas];

       deltStats = stats - Most[singExp];
       deltStatSamples = Map[deltStats /. # &, samples];

       twos = Fold[Plus[#1,doubler[#2]]&,
                   doubler[deltStatSamples[[1]]],
                   Rest[deltStatSamples]] / Length[deltStatSamples];

       threaded = Thread[{Map[stats /. #&, samples], deltStatSamples}];

       method1twos = Fold[
           Plus[#1, vMult[#2[[1]], #2[[2]]]] &,
           vMult[threaded[[1]][[1]],threaded[[1]][[2]]],
           Rest[threaded]]/Length[threaded];

       mylps = Map[Last[deltas], samples];
       threaded = Thread[{mylps, deltStatSamples}];
       lps = Fold[
           Plus[#1, thirddim[{#2[[1]]}, doubler[#2[[2]]]]] &,
           thirddim[{threaded[[1]][[1]]}, doubler[threaded[[1]][[2]]]],
           Rest[threaded]]/Length[threaded];

       threes = Fold[Plus[#1, tripler[#2]] &,
                     tripler[deltStatSamples[[1]]],
```

```
                    Rest[deltStatSamples]]/Length[deltStatSamples];

          WdotP = computeWdotP[samples, trueP, Ww];

          wParts1 = Map[computeSum[Function[{state},
                                      (#[state] * WdotP[state])],
                                samples]&, poDelt];
          wParts2 = Map[compExp[Function[{state},
                                     (#[state] *
                                      WdotP[state])/trueP[state]]]&,
                    poDelt];
          wParts2 = N[wParts2];

          {twos, lps, threes, wParts1, wParts2, method1twos}
         ];
```

This function uses the results of the preceding function to arrange the matrix $\mathbf{A}$ and the vector $\mathbf{B}$.

```
abv[tp_, mu_, Ww_, method_, obs_:{}] :=
    Module[{bVec, aMatr, mus, lmbases, trueP, twos,
            lps, threes, wParts1, wParts2, mlps, mparams, m12s},
           trueP[state_] := tp[state];
           mus = Table[Subscript[Global`\[Mu], i] -> mu[[i]],
                       {i, 1, Length[mu]}];
           {twos, lps, threes, wParts1, wParts2, m12s} =
               computeExpectations[mus, trueP, Ww, obs, statesGenerator];

   lmbases = N[SparseArray[Flatten[
                      Table[ids[i],
                            {i, 1, Length[GVbases]}]] /. mus]];

           {aMatr, bVec} =
```

```
      If[method==2,
         mlps = lps[[All,All,1]] /. mus;
         mparams = N[GVparams /. mus];
         {Transpose[Transpose[(lmbases)].
                    ((N[mlps] - N[twos] + N[mparams].N[threes]))],
          wParts2}
         ,
         (* else *)
{-(m12s.lmbases), wParts1}
        ];
 {bVec, aMatr}
];
```