Algorithms for Inferring the Information Topology of Statistical Mechanics Models

Kolten Barfuss

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Mark K. Transtrum, Advisor

Department of Physics and Astronomy

Brigham Young University

April 2017

ABSTRACT

Algorithms for Inferring the Information Topology of Statistical Mechanics Models

Kolten Barfuss
Department of Physics and Astronomy, BYU
Bachelor of Science

Many-parameter models of complex systems are ubiquitous, yet often difficult to interpret. To gain insight, these models are often simplified, sacrificing some of their global considerations as well as versatility. The task of finding a model that balances these features is of particular interest in statistical mechanics. Our group addresses the problem through a novel approach—the Manifold Boundary Approximation Method (MBAM). As the central step to this approach, we interpret models geometrically as manifolds. Many model manifolds have a set of boundary cells arranged in a hierarchy of dimension. Each of these boundary cells is itself a manifold which corresponds to a simpler version of the original model, with fewer parameters. Thus, a complete picture of all the manifold's boundary cells—the boundary complex—yields a corresponding family of simplified models. It also characterizes the relationships among the extreme behaviors of the original model, as well as relationships among minimal models that relate subsets of these extreme behaviors. This global picture of the boundary complex for a model is termed the model's manifold topology. Beginning in the context of statistical mechanics, this thesis defines a class of models—Superficially Determined Lattice (SDL) models—whose manifold topologies can be ascertained algorithmically. This thesis presents two algorithms. Given an SDL model, the Reconstruction Algorithm determines its manifold topology from minimal information. Given a model and desired extreme behaviors, the Minimal Model Algorithm finds the simplified model (with fewest parameters) that interpolates between all of the behaviors.

Keywords: Algorithms, Information Geometry, Topology, Manifold, Polytope, Algebraic Lattice, Statistical Mechanics

ACKNOWLEDGMENTS

I owe thanks to the many excellent professors and exemplary instructors with whom I associated. They taught me to think critically, improve my problem solving approach, and use mathematical tools with rigor. I wish to thank Drs. Reese, Durfee, Powers, Berrondo, Turley, Colton, and Hart for their effort in teaching and their patience with my questions and weaknesses. I am especially thankful to Dr. Transtrum. I have had the privilege of being instructed in classes twice by Dr. Transtrum. Both of these classes were exceptional learning experiences, and formative in ways that extended beyond the material. Dr. Transtrum has exhibited this same excellence and patience in mentoring me throughout this project, for which I thank him. Never did he let my shortcomings stop him from doing what he does best—helping the student to understand correctly and accomplish the work him or herself. I must also thank Garett Brown, my best friend during my undergraduate career. We have taken 25 courses together, and his sharp mind, brilliance, and insightful questions have enhanced my experience more than I can express. His help has been of great value to me, and I am a better student because of him. More importantly, Garett's friendship has been invaluable, and I am a better man for having known him and seen his selfless kindness. Finally, I express deepest gratitude to my wife and son, without whose happiness and support I can do nothing. Words cannot convey all that they do for me. Thank you Santiago, and thank you dearest Betsy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Many-parameter models are often difficult to interpret. The complicated interplay between parameters obscures the possible importance of any particular parameter. Moreover, such a model is often not suited to simpler situations of both theoretical and experimental interest. Nevertheless, simple (fewer-parameter) models are unsatisfying for their own reasons. They lack the global considerations and versatility of a more complete (but more complicated) model. The task of finding a model that balances these features is of particular interest in statistical mechanics. In the case of binary alloys, the goal is often to predict ground state atom configurations with accuracy and computational efficiency. This motivates the search for a predictive yet appropriately simple model through reducing the number of parameters in a complex model.

There are various methods of modeling alloys which achieve simplicity in different ways. Seko *et al.* used a cluster-expansion-based method based on few assumptions [1]. Hart's approach employed purely geometric assumptions to predict as-yet-unobserved ground state lattice arrangements [2]. Recently, Transtrum *et al.* presented a new method for understanding statisti-

cal mechanics (and other) models and obtaining useful reduced models—the Manifold Boundary Approximation Method (MBAM) [3, 4].

Applying MBAM allows for selection of a model that achieves specific desired extreme behaviors. Here the term extreme behavior means a characteristic dynamical phase—a macroscopic, collective behavioral regime of the system [4]. The result of MBAM is a family of models of varying complexity that achieve different sets of extreme behaviors. The simplest have only two distinct extreme behaviors; the full model includes all original parameters and connects all of the extreme behaviors. Through information topology, MBAM clarifies the connection between a model and its possible simplifications [4]. It grants power to choose the simplest possible model that gives desired extreme behaviors. In this paper, I define a class of models for which MBAM can be automated and present the two algorithms essential to this implementation.

1.2 Other Methods

Many other methods for model reduction exist. The present work does not attempt a full comparison of these with MBAM, though it does highlight some of the latter's advantages. There is no intent, however, to present this approach as either superior or inferior to others. The paradigm that motivates MBAM is distinct, as are some of its key insights and results. Therefore, the goal is to develop MBAM as a tool for model exploration and characterization, with the hope of adding new understanding to that yielded by other methods. With this in mind, I discuss briefly one other relevant method used in statistical mechanics.

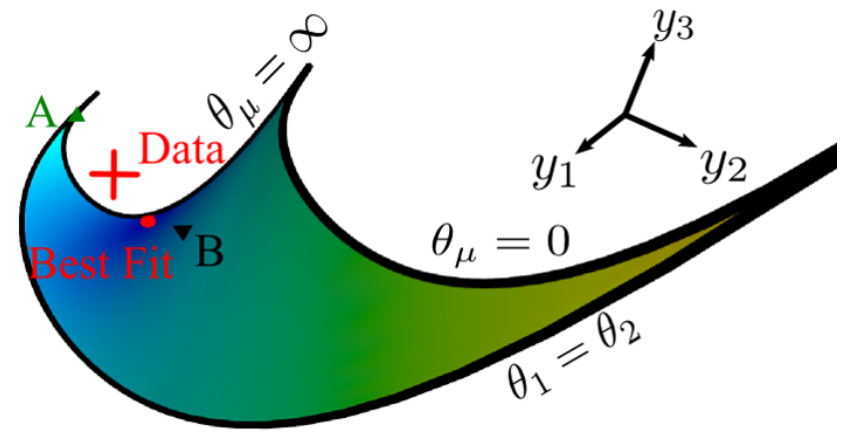
Compressed sensing is a typical tool for dealing with complex models in statistical mechanics. This method allows the theorist to set all but a few parameters to zero [5]. Despite the problem being under-determined, compressed sensing results in a surprisingly accurate model [5–8]. Though MBAM is independent of compressed sensing, a connection between the two is suspected.

Our research group is looking into the relationship, but more investigation is needed before any conclusive statements can be made.

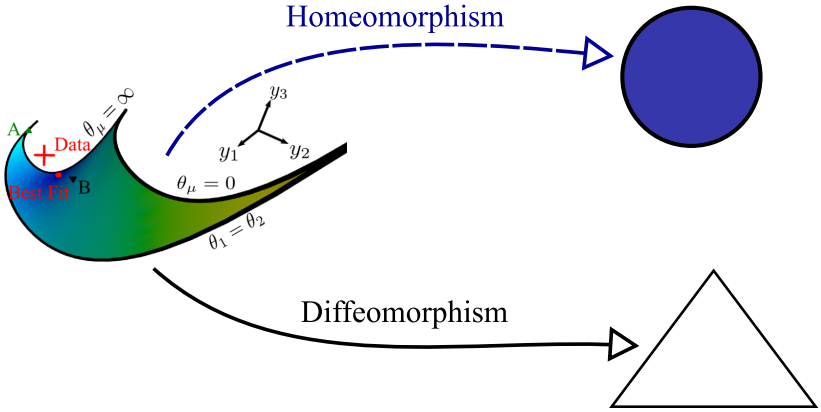
1.3 Manifold Boundary Approximation Method (MBAM)

The Manifold Boundary Approximation Method aims to understand, approximate, and simplify a complex model by looking to the boundary of its model manifold. For many models, the corresponding manifold is naturally divided into boundary cells (see Fig. 1.1) by cusps (the sharp points in the boundary). The set of boundary cells of a manifold, known as the boundary complex, has a hierarchical structure in most cases. Transtrum *et al.* show that each element of the boundary complex leads to a different simplified model [4]. Therefore, the goal of MBAM is to deduce properties of the model and discover useful simplifications through study of the boundary complex. To this end, the principal focus of this thesis is solving the general problem of characterizing the set of boundary cells of the manifold, or boundary complex.

As the central step to the MBAM approach, models are interpreted as multidimensional manifolds in data space. A model is associated with a manifold by varying each parameter through all physically allowable values (even those not experimentally achievable). The result of this process, the model manifold, is a Riemannian manifold, i.e. a type of smooth hypersurface on which derivatives can be taken and distances and angles can be measured [9]. Importantly, models have limits on the range of predictions that result in bounded (rather than infinite) manifolds [4]. This fortunate consequence implies that the model manifold is often topologically equivalent (specifically, diffeomorphic—see glossary) to a polytope (the generalization of a polyhedron in higher dimensions—see glossary or Ref. [12]). In more general terms, the manifold fills some high-dimensional volume of data space, demarcated by its boundary. The differential structures of the boundary naturally divides it into a hierarchy of smooth cells.



(a)



(b)

Figure 1.1 A parametrized model with N structurally identifiable parameters (equivalently, linearly independent parameter directions) corresponds to an N-dimensional manifold in data space [4]. In this example N is 2, and the manifold is 2-dimensional. The manifold is obtained by varying the parameters through all possible allowed values. There are limits to the predictions a model can give. These limits translate to a boundary on the manifold which is divided into boundary cells. Although the manifold is homeomorphic to a disk, it is diffeomorphic to a triangle. Diffeomorphisms preserve the boundary cell structure. The manifold gives insights into the model that are otherwise not obvious. This information is gleaned through study of the manifold's set of boundary cells, or boundary complex.

As an example, consider the two-dimensional manifold shown in Figure 1.1. This manifold corresponds to the two-parameter model $y(t,\boldsymbol{\theta}) = e^{\theta_1 t} + e^{\theta_2 t}$, where θ_1 and θ_1 are the parameters. Now, imagine that we are given data points for three times. This corresponds to a vector in data space [the red cross in Fig. 1.1(a)]. Now, for a particular choice of $\boldsymbol{\theta}$ (say $\boldsymbol{\theta_0}$), the three values $y_1 = y(t_1,\boldsymbol{\theta_0})$, $y_2 = y(t_2,\boldsymbol{\theta_0})$, and $y_3 = y(t_3,\boldsymbol{\theta_0})$ also define a vector in data space. In the figure are shown the best fit (red), fit A (green triangle), and fit B (black triangle). In fact, varying the two parameters through all possible values gives a surface in data space—the manifold. Each of the three boundary cells corresponds to a limiting approximation in the parameters. (In general, these parameters are combinations—sums, products, ratios, etc.—of the original parameters, and hence are referred to as new parameters when the distinction is important). The boundary cells are topological invariants, where topology refers to *differential* topology. In other words, the boundary cell structure is preserved by diffeomorphisms, as shown in Figure 1.1(b).

Like a polytope, the boundary complex of the model manifold has a hierarchical structure. The structure is hierarchical in the sense that some boundary cells (e.g. a two-dimensional face) *contain* other boundary cells (e.g. edges). Containing boundary cells are higher in the hierarchy than those they contain. The hierarchy is determined by two features of the manifold's differential topology: dimension and adjacency. The dimension of the manifold is given by the number of structurally identifiable parameters (or equivalently, the number of linearly independent parameter directions) [4]. Likewise, the dimension of any element of the boundary complex is given by the number of linearly independent parameter directions that span said boundary. The adjacency of boundary cells is the central organizing principle of the hierarchy. Two N-dimensional boundary cells are adjacent if they share an $(N-1)$-dimensional boundary. An element of the boundary complex is grouped together with those adjacent to it. The adjacency relationships among the boundary cells of the model need to be understood because these connections imply similar relationships among extreme behaviors of the model and simplified models.

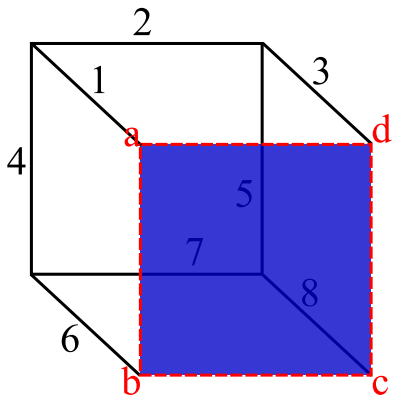


Figure 1.2 A three-dimensional cube has boundary cells that exhibit a hierarchical structure. The highest-dimensional boundary cells are the two-dimensional faces. A particular two-dimensional face (blue) is adjacent to four one-dimensional boundary cells (red dashed edges) and four zero-dimensional boundary cells (vertices $a-d$). It is not adjacent to 8 edges (black) because these edges do not form part of its boundary. The blue face and all cells adjacent to it (the four red edges and vertices $a-d$) form a branch of a hierarchy (like a branch of a tree), where the highest-dimensional cell is at the top, followed by those of successively lower dimensions.

Take for example the three-dimensional cube shown in Figure 1.2. Its boundary cells are two-dimensional faces, each of which is adjacent to the cube. Each face (e.g., the blue one), in turn, has four one-dimensional boundary cells (red dashed edges) to which it is adjacent, as well as four zero-dimensional boundary cells (vertices $a-d$). There are eight edges (labeled in black) to which the blue face is not adjacent. These edges do not form part of the boundary of the face. The blue face and all elements adjacent to it form a branch of the hierarchy (think of a branch of a tree), with the blue face succeeding all lower-dimensional elements in the branch. In this way, each group of adjacent elements form a branch of the hierarchy, with the rank of an element in the hierarchy being determined by its dimension. These same relationships are present in the family of possible simplifications of the model generated by MBAM.

Each boundary cell of the model manifold corresponds to a simplified model whose extreme behaviors are the vertices of the boundary [3, 4]. An N-dimensional boundary has N linearly

independent parameter directions, and gives a model with N structurally identifiable parameters [4]. This model is found by taking certain parameters to limiting values in a way that leads across the manifold to the boundary, eventually eliminating all parameters whose directions of movement are not included in the boundary. In particular, a zero-dimensional boundary (vertex) gives a zero-parameter model, or collective macroscopic behavior, and corresponds to taking limits of parameters to reach that vertex [4]. An arbitrary boundary in general contains, lower-dimensional boundary cells with fewer vertices.

Taking the correct limits to arrive at one of these boundary cells yields a simpler model that achieves fewer extreme behaviors. The vertices contained in a boundary cell reflect all possible extreme behaviors that can be achieved by its corresponding model. Thus, the differential topology of a manifold's boundary complex (hereafter manifold topology) reveals all possible extreme behaviors the full model can achieve, as well as families of models that relate and describe them. With the power of this information in mind, I present a method for characterizing the manifold topology.

1.4 Overview

In the remainder of this paper, I solve the problem of characterizing a model's manifold topology in order to apply MBAM. The solution is not completely general. However, there exists a wide class of models for which minimal information is necessary to reconstruct the entire manifold topology. Sections 2.1 to 2.3 rigorously define this class as Superficially Determined Lattice (SDL) models. Section 2.4 interprets SDL's in the context of MBAM. Within the SDL class, the problem of discovering the complete manifold topology lends itself to an algorithmic solution. Section 2.5 devises this solution from the definition and proven characteristics of an SDL model. To conclude, Chapter 3 demonstrates applications of the solution to some statistical mechanics cases.

Defining the SDL class reduces the problem of manifold topology reconstruction to one of obtaining the Superficial Adjacency Information (SAI)—the aforementioned minimal necessary information. This makes the problem approachable. Otherwise, solution methods amount to searching the manifold for boundary cells along geodesics. The number of possible boundary elements grows combinatorially with dimension. Thus, because of the growing computational cost, it is practically impossible to discover the complete manifold topology via geodesic search. The success and validity of the algorithmic solution hinges on the SAI being necessary and sufficient. Therefore, I take great care to precisely define the conditions for which this is so. The proofs themselves require mathematical concepts likely unfamiliar to a physicist and which therefore require some fairly detailed explanation. All of this rigor proves its worth, however, by yielding the solution naturally.

The solution constitutes two algorithms; the first reconstructs the complete manifold topology, and the second constructs minimal models given desired extreme behaviors. Section 2.5 presents each algorithm and discusses their merits and shortcomings. Importantly, the second algorithm compliments the principal weakness of the first. Together, they are a tool set for gaining insights into complex statistical mechanics models.

The concluding chapter presents an application of each algorithm to statistical mechanics. These examples are not new results in statistical mechanics. Their primary purpose is simply to demonstrate the use and utility of the algorithms. Nevertheless, with each example I discuss wider possible applications not limited to statistical mechanics. My goal is to convey the versatility and power of these tools for exploring models. My hope is that the reader will realize how to use them in their own work.

Chapter 2

Mathematics and Algorithms

2.1 Introducing the Problem

For certain statistical mechanics of models (specifically, exponential families), an equivalence allows the use of what is called a convex hull as a starting point for ascertaining a model's manifold topology. In order to illustrate the idea of a convex hull, the derivation of the convex hull for an exponential model is shown in this section. The equivalence between the convex hull associated with a model and the model's manifold are also illustrated. For computational reasons, working with the convex hull is preferable to working with the model manifold directly. However, the convex hull immediately yields only some of the information about the manifold. Therefore, the problem to be solved is to deduce the remaining manifold topology from that given by the convex hull.

In statistical mechanics, an exponential model may be associated with a convex hull, a term which will be clarified presently. These exponential models are of the form that the probability of finding the system in a configuration determined by the random variables $\boldsymbol{s}$ is

$$P(\boldsymbol{s}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_k \theta_k \Pi_k(\boldsymbol{s})\right). \tag{2.1}$$

Note that $\boldsymbol{s}$ and $\boldsymbol{\theta}$ are vectors—each component of $\boldsymbol{s}$ is a random variable and each component of $\boldsymbol{\theta}$

is a parameter. Here, $Z(\boldsymbol{\theta})$ is the normalization (the partition function), the $\{k\}$ label parameters, and each Π_k is a function of the random variables $\boldsymbol{s}$, which determine the configuration of the system. If we use i to label a configuration, this equation can be rewritten as

$$P_i = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_k \theta_k \Pi_{ik}\right). \tag{2.2}$$

Now, Π_{ik} can be interpreted as a matrix, where each row corresponds to a choice of parameter values for a particular configuration (a choice of $\boldsymbol{s}$). (In contrast, a column of Π_{ik} corresponds to a choice of weights for a particular parameter for *all* configurations.) Think of the rows of the Π matrix as vectors in parameter space. Then the matrix defines a cloud of points in parameter space. Finally, the convex hull is (in rigorous terms) the boundary of the minimal polytope that contains all the points such that no straight line connecting any two points ever leaves the boundary. For a more visual illustration of the convex hull, see the example in Figure 2.1.

In Figure 2.1 is an illustration of the equivalence between the convex hull, the model manifold, and a polytope (a triangle). The convex hull [blue part of (b)] is the boundary surrounding the points of the Π matrix. The boundary of the model manifold in (a) is diffeomorphic to the convex hull; both are diffeomorphic to the boundary of a triangle [10]. For a rigorous definition of diffeomorphism, please see the glossary. For the purposes of this thesis, a diffeomorphism is a transformation that preserves the number of faces, their respective dimensions, and their adjacency relationships with one another. Within the context of MBAM, diffeomorphism is equivalence. The equivalences shown are general to all models in the exponential family [10]. The characteristics of a boundary cell of the manifold that have meaning for the complex model—dimension and adjacency to other boundary cells—are invariant to diffeomorphic changes. Thus, identifying the full structure of the convex hull is equivalent to identifying the full manifold topology.

The convex hull is preferable for computational methods because its faces (of all dimensions) are flat. In the previous sentence, flat means that an n-dimensional face does not curve into the $(n+1)$st dimension. In general, a complex model has many parameters, which (recall from Sec.

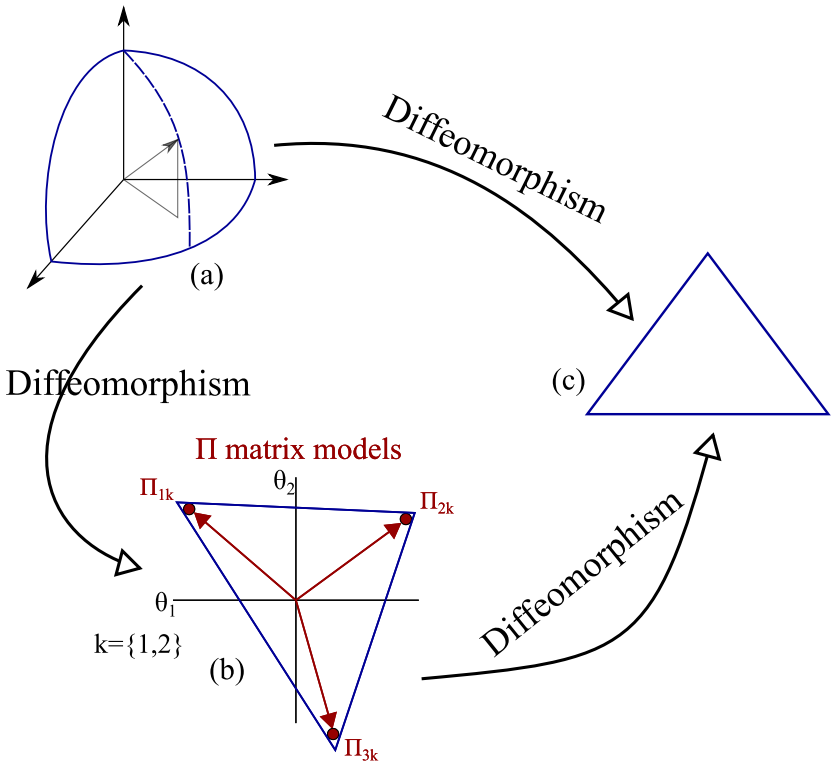


Figure 2.1 The boundary of the model manifold in (a) is diffeomorphic (see glossary) to the convex hull [blue part of (b)] of the rows of the Π matrix (points in parameter space) [10]. In this case Π is 3×2, because there are two parameters (θ_1 and θ_2) and three configurations, giving three points (rows of Π) in parameter space. In general, the Π matrix can be derived from an exponential model as shown. The convex hull is the boundary of the minimal polytope that contains all the points such that no straight line connecting any two points ever leaves the convex hull. This model manifold boundary and this convex hull are diffeomorphic to any triangle's boundary. In general, any model from the exponential family exhibits these diffeomorphism between the manifold boundary, the convex hull, and a polytope boundary [4]. The features of the manifold boundary that have meaning for the model are invariant to diffeomorphic transformations. Therefore, for the purposes of MBAM the convex hull of the Π matrix is equivalent to the boundary of the model manifold.

1.3) translates to a high-dimensional manifold. Therefore, investigation of the resulting boundary complex is only practical numerically.

Many suitable convex hull algorithms exist, but each only gives superficial information. In topological terms, these programs yield the highest-dimensional boundary cells, each with a list of its adjacent vertices (i.e., the vertices each face contains). I term this information the Superficial Adjacency Information (SAI), as it specifies how the highest-dimensional elements of a boundary complex are topologically adjacent to the zero-dimensional ones. To apply MBAM, we must reconstruct the entire manifold topology, including the intermediate-dimensional boundary elements. Thus, a convex hull algorithm is only a first step.

The pertinent question is this: when is the SAI sufficient to completely reconstruct the manifold topology? The remainder of this chapter answers this question by defining the Superficially Determined Lattice (SDL) class. However, to understand this answer and the reconstruction algorithm that follows, we need a basic mathematical framework.

2.2 Mathematical Framework

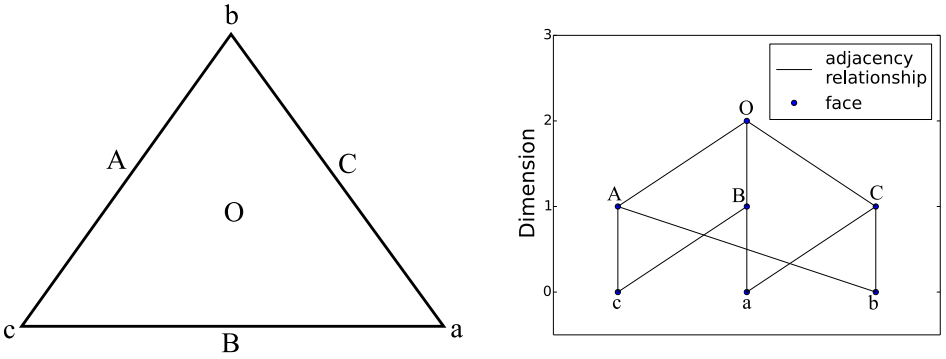
The differential topology of the boundary complex (i.e., the manifold topology) captures the characteristics of the model essential for MBAM. For a more detailed justification of this statement, see Sections 1.3 and 2.4. Although beyond the scope of this section the term manifold topology suffices when referring to the differential topology of the boundary complex, this section requires more precise language. Specifically, the language of partially ordered sets, algebraic lattices, and abstract polytopes naturally describes the differential topological structures that arise as boundary complexes of model manifolds. To facilitate understanding of the remaining terms, I first present the Hasse diagram, a directed graph useful for visualizing partially ordered sets. Definitions of the essential mathematical terms follow. Finally, I define several new concepts that characterize

the differential topologies completely determined by their respective SAI. This section culminates with the definition of a Superficially Determined Lattice.

The clearest tool for representing these differential topological structures, mentally or visually, is the Hasse diagram (see Figure 2.2(b)). All of the differential topological information is contained in the Hasse diagram by the nodes, which represent boundary cells, and the connecting lines, which represent adjacency relationships. A higher-dimensional element is connected by these lines to all of the lower-dimensional elements it contains. In Figure 2.2 a triangle is shown next to its Hasse diagram representation. The vertical axis represents the dimension of boundary elements. The horizontal axis has no meaning, and exists only to visually separate the nodes (blue circles) that represent the boundary elements. Each boundary element of the triangle is labeled in both diagrams; the element O represents the entire triangle. The Hasse diagram representation has the particular advantage of extending naturally to any dimension. Additionally, it provides natural visualization of some of the proofs of the theorems in Sec. 2.3.

Now that we have a mental framework in place, the next order of business is to define the mathematical terms essential to the remaining discussion. A *partially-ordered set (poset)* $\mathscr{P}$ is a set with a partial ordering relation defined on its elements [11]. Any two elements $a \in \mathscr{P}$ and $b \in \mathscr{P}$ are related in one of three ways: a succeeds b ($a \succ b$), a precedes b ($a \prec b$), or a is incomparable to b ($a \nsim b$). In this sense, a finite poset may have maximal (minimal) elements if no other element in the set succeeds (precedes) them. If there is one unique maximal (minimal) element, it is the *greatest* (*least*) *element*. Further, a subset of elements $P \subset \mathscr{P}$ can have a set of upper (lower) bounds U (L)—all elements in U (L) succeed (precede) all elements in P. In particular, there can be a *least upper bound* or *supremum*, $x_S \in U$, (correspondingly *greatest lower bound* or *infimum*, $x_I \in L$,) which precedes (succeeds) all of the other upper (lower) bounds [11]. That is,

$$x_S \prec u \; \forall \; u \in U \; s.t. \; u \neq x_S \tag{2.3}$$



(a) (b)

Figure 2.2 A basic example of the Hasse diagram of a triangle (b), next to the triangle it represents (a). For clarity, differential topological structures of boundary complexes are represented as Hasse diagrams throughout this paper. The vertical axis is the dimension of boundary elements. The horizontal axis has no meaning, and exists only to visually separate the nodes (blue circles) that represent the boundary elements. Each boundary element of the triangle is labeled along with its corresponding element in the Hasse diagram. The element O represents the entire triangle.

and

$$x_I \prec l \; \forall \; l \in L \; s.t. \; l \neq x_I. \tag{2.4}$$

A poset for which any two of its elements have a supremum and infimum is called an *algebraic lattice* (hereafter lattice) [11].

Another type of poset is the *abstract polytope* (often simply polytope). For a complete definition of abstract polytope, the reader is welcome to see the glossary. As it turns out, an exploration of the definition is not necessary for the following discussion. The features of polytopes essential for the understanding needed here are, for the most part, derived properties that are fortunately more intuitive than the definition itself. With this in mind, I shall state briefly that a polytope P of rank N is a ranked poset of faces [12]. This rank can in many circumstances be called the dimension of the polytope. Technically, an abstract polytope has a rank but no dimension—a *realization* of a polytope (an embedding of the faces in a coordinate space) has dimension. Further, the dimension

is equal to the rank when the realization is faithful (see Ref. [12]). In the cases addressed in this thesis, a faithful realization exists, and the rank of the abstract polytope can therefore be thought of as its dimension. This implies that each face of the polytope also has a dimension. In particular, there are zero-dimensional faces (vertices), one-dimensional faces (edges), two-dimensional faces, etc. All of this work is to say that a polytope of rank N can be thought of as the N-dimensional generalization of a polyhedron.

In most cases, the differential topology of the model manifold is both a polytope and a lattice. There are some exceptions, such as lattices that are not polytopes, and manifolds that are neither polytopes nor lattices—this paper does not treat these cases. Our research group speculates that they are few in the context of modeling complex systems. The reason for introducing the term polytope above is for the natural terminology it carries, such as vertices, edges, faces, and dimension of faces. This last property is particularly useful, as the boundary cells of a manifold naturally have an associated dimension. The elements of a lattice, in contrast, have no dimension unless a rank function is additionally given.

An example illustrating several of these mathematical terms is given in Figure 2.3, which shows the Hasse diagram of a square pyramid. As mentioned, lattices can be given a rank function to associate a dimension with their elements. Conversely, the differential topology of a polytope often has an algebraic lattice structure, as is the case with this square pyramid. The axes in Fig. 2.3 are as in Fig. 2.2. It is now clear that a line connecting two elements denotes that the upper element succeeds the lower one. The greatest element (green) is clearly identifiable at the top of the diagram, and in the language of polytopes represents the entire pyramid. Edge 1 (red) is the supremum of the vertices a and b ($\sup(\{a,b\})= 1$). Likewise, edge 2 is the infimum of the two-dimensional faces A and B ($\inf(\{A,B\})= 2$). The least face (not shown), which has dimension -1, precedes all of the zero-dimensional faces, ensures that any two faces have an infimum. For simplicity of discussion, the least face is omitted throughout this paper.

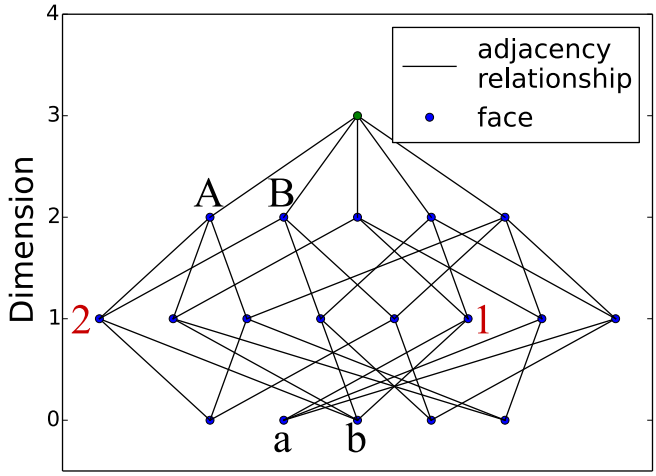


Figure 2.3 Hasse diagram of a square pyramid (a lattice). The greatest element (green) ensures that any two elements have a supremum. Element 1 (red) is the supremum (least upper bound) of the elements a and b. Element 2 (red) is the infimum (greatest lower bound) of the elements A and B. The line connecting A to 2 implies that 2 precedes A. Not shown is the lattice feature of the least face.

Finally, to be determined by their SAI, lattices must have several additional properties that are as yet undefined in lattice theory. The definitions of these properties along with those of several related concepts are presented here. In the following definitions, X is a lattice where each chain (think of a path between two elements on the Hasse diagram) is finite. Let $m(X)$ and $M(X)$ be the sets of minimal and maximal elements of X, respectively. Let $W = X - (m(X) \cup M(X))$. Let $\mathcal{P}(S)$ be the power set of S. X is *supremum-filled* if for all non-minimal x in X, there exist $a,\ b \in X$ such that $a \nsim b$ and the supremum of $\{a,b\}$ is x. Similarly, X is *infimum-filled* if for all non-maximal x in X, there exist $a,b \in X$ such that $a \nsim b$ and the infimum of $\{a,b\}$ is x. Next, define a vertex function $V_X : X \to \mathcal{P}(m(X))$ by $y \mapsto \{x \in m(X) \mid x \prec y\}$. Likewise, define a maximal face function $F_X : X \to \mathcal{P}(M(X))$ by $y \mapsto \{x \in M(X) \mid y \succ x\}$. With these definitions in place, the question of when the SAI is sufficient to reconstruct the manifold topology can now be answered.

A finite lattice W can be reconstructed from the SAI if it is supremum- and infimum-filled.

In this paper, such a lattice is termed a Superficially Determined Lattice (SDL). Similarly, SDL model refers to a model whose boundary complex meets the SDL criteria. As Section 2.3 shows, it is equivalent to say that a lattice W with injective functions V_W and F_W is an SDL. Though the question now has an answer, it is largely unjustified. The proofs of these statements constitutes a significant portion of my work. An outline of these proofs and the supporting logic is given in the next section.

2.3 Theorems

This section presents several theorems with the aim of proving that the SAI is sufficient to reconstruct an SDL. Though the proofs are not terribly cumbersome, they do interrupt the flow of logic. Consequently, I have placed them in an appendix (see Appendix A). Note that, because of the symmetry properties of finite lattices, each theorem has a dual theorem whose proof is practically identical once the lattice (think of the Hasse diagram) is inverted. The first three theorems build on one another, and so are discussed together. Theorem 4 is a summary of the these first three theorems, and is consequently followed by an elucidation of how the statements in Theorems 1-4 support the statement that an SDL can be reconstructed from the SAI. Finally, Theorem 5 is presented, although its significance is clarified later. In what follows, X and W are as in the previous section.

Theorem 1. For a finite-chained, supremum-filled poset, each element is the supremum of two immediately preceding elements.

Theorem 2. W is supremum-filled if and only if each element x of W is the supremum of $V_W(x)$ (i.e., $\sup(V_W(x)) = x$).

Theorem 3. W is supremum-filled if and only if V_W is injective. Note here that the dual theorem is this: W is infimum-filled if and only if F_W is injective (*not* surjective). Think carefully

about inverting the Hasse diagram and the definitions of infimum-filled and F_W to see this.

Theorems 1-3 outline a proof by construction that each element x of a supremum-filled lattice W is uniquely identified by the minimal elements $V_W(x)$. Recall that suprema are unique. Thus, Theorem 1 means that any element x can be identified uniquely as the supremum of two immediately preceding elements a and b. Each of a and b in turn can be uniquely identified by two elements that precede them—let's say p and q immediately precede a and likewise r and s immediately precede b. By implication, x is the supremum of $\{p,q,r,s\}$ and can thus be uniquely identified by these elements. By advancing down successive levels of the lattice (think of the Hasse diagram again), x can eventually be identified by some set of the minimal elements—in fact, all of the minimal elements that precede x. This is essentially what Theorem 2 states. Theorem 3 nearly amounts to a restatement of Theorem 2 in the language of functions, but additionally states that $V_W(x)$ is unique, i.e., each element of the lattice can be *uniquely* identified by a set of minimal elements. However, the minimal elements are only a portion of the SAI. The following example illustrates why knowing the minimal elements alone is not sufficient.

Consider a lattice W as above that is also a polytope. The elements are therefore faces, and the minimal elements are vertices (zero-dimensional faces). Suppose W has n vertices ($|m(W)| = n$), each of which is given an integer index. Each of the $n!$ combinations of vertices constitutes a possible additional face of W. However, this cannot be a valid procedure for identifying all faces. Consider the case where $n = 4$. This is true of both a square and a tetrahedron. If this procedure of including each combination of vertices as a distinct face were valid, the square and tetrahedron would have the same number of faces. Yet they clearly do not; a square has nine (four vertices, four edges, and the two-dimensional face), and a tetrahedron has 17 (four vertices, six edges, four two-dimensional faces, and the three-dimensional face). So, how can a set of vertices that does not correspond to an actual face of the polytope be excluded? Also, how can an algorithm ensure that all faces are found? The answer lies in leveraging the properties of a supremum- and infimum-filled

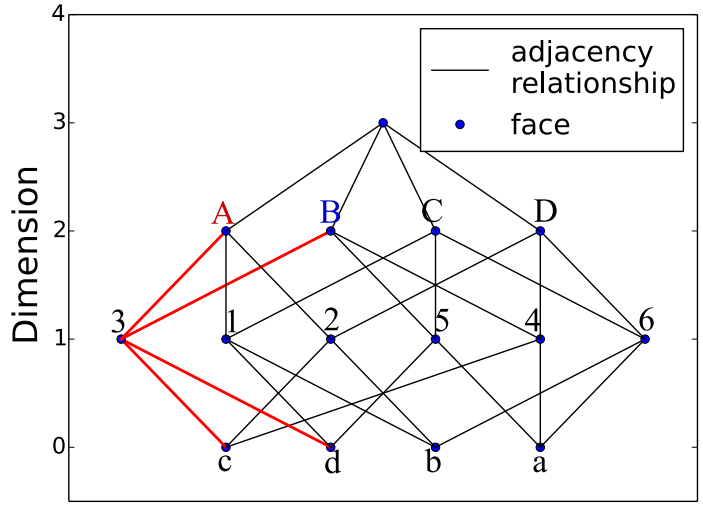
lattice.

In order to identify all elements of an SDL, both the supremum-filled and infimum-filled properties must be exploited. Theorem 4 (The Equivalence Theorem) makes the definitions of these two properties more usable. Because both properties are needed, The Equivalence Theorem not only summarizes the previous results about supremum-filled lattices, but also explicitly states the dual results regarding infimum-filled lattices.

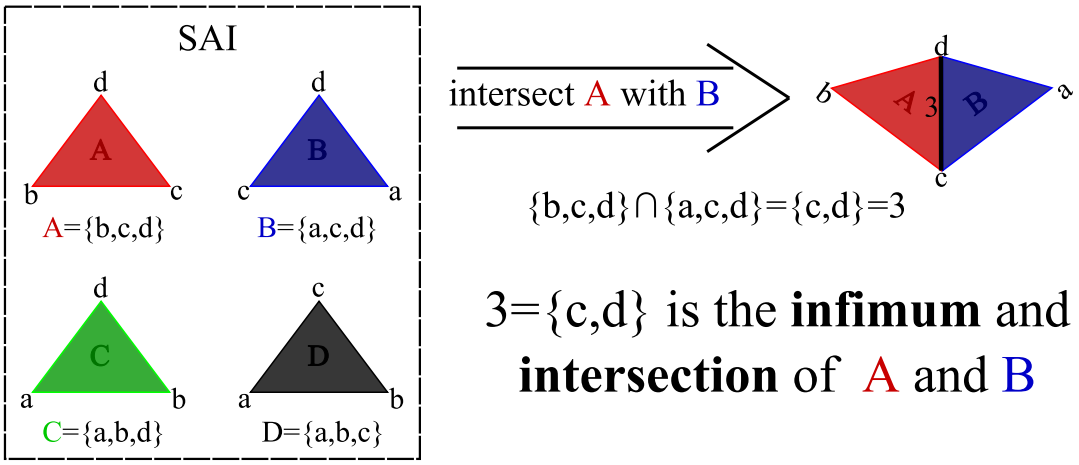
Theorem 4—The Equivalence Theorem. In summary, W is supremum-filled if and only if V_W is injective if and only if for all $x \in W$, $\sup(V_W(x)) = x$. Likewise, W is infimum-filled if and only if F_W is injective if and only if for all $x \in W$, $\inf(F_W(x)) = x$.

From the SAI, all intermediate elements of an SDL can be found by searching for infima. Because the SDL is infimum-filled, every element $x \in W$ (besides the maximal ones) is the infimum of some set of the maximal elements. Therefore, taking all combinations of maximal elements and finding the infimum of each of those sets guarantees that every element will be found—it is a brute force search for every possible infimum. However, it is possible for the same element x to be identified as the infimum of two different sets of maximal faces (say $F_1 \subset F_W(x)$ and $F_2 \subset F_W(x)$). In such cases, it is always true that $F_1 \subset F_2$ or $F_2 \subset F_1$. This does not violate Theorem 4; x is still uniquely identified as $\inf(F_W(x))$, the infimum of *all* of the maximal faces that succeed x. It is just also possible that x is the infimum of subsets of $F_W(x)$. Thus the question arises: in such a case, how can an algorithm recognize that two elements are actually the same element? This problem is resolved using the supremum-filled property.

Using the SAI and the supremum-filled property, each infimum can be uniquely labeled by minimal elements. To see how this works, suppose as before that each minimal element is given an integer index. Now using the SAI, identify (i.e., label) each maximal element F_i by $V_W(F_i)$, the set of *indices* of all minimal elements that precede it. As the upcoming example shows, finding the infima f of a set of maximal elements $\{F_i\}, i \in \mathbb{I}$ amounts to taking the set intersection of all the



(a)



(b)

Figure 2.4 The Hasse diagram (a) and the SAI (b) for an SDL diffeomorphic to the boundary of a tetrahedron. The SAI consists of each of the four two-dimensional triangular faces ($A-D$) and, for each face, the vertices contained by the face (e.g., A contains $b-d$). From the Hasse diagram, it is clear that $\inf(A,B)=3$, which can be represented by $\{c,d\}$ (see Theorem 3). Thus, the infimum of A and B is the set intersection of their respective sets of vertices ($\inf(\{A,B\})=\{b,c,d\}\cap\{a,c,d\}=\{c,d\}$). This is also the geometric intersection of A and B.

$V_W(F_i)$, i.e., $V_W(f) = \bigcap_{i\in\mathbb{I}} V_W(F_i)$. When lattice elements correspond to faces of a polytope (and hence minimal elements become vertices), infima also correspond to the geometrical intersection of the faces. In fact, the geometrical intersection of a set of faces corresponds exactly to the set of vertices shared by the faces.

Consider an SDL that is diffeomorphic to the boundary of a tetrahedron. Then, as shown in Figure 2.4(b), the SAI consists of each of the four two-dimensional triangular faces ($A-D$) and, for each face, the vertices contained by the face (e.g., A contains $b-d$). From the Hasse diagram [Fig. 2.4(a)], it is clear that $\inf(\{A,B\}) = 3$. Further, by Theorem 3, we can label the lattice element 3 by its minimal elements, $V_W(3) = \{c,d\}$. Yet it is clear from Fig. 2.4(b) that this set of vertices also corresponds to the geometric intersection of A and B (in this case, a line). This gives us a visual representation of the process of searching for infima. The search involves choosing different combinations of maximal faces, intersecting two, three, or (in higher dimensions) more of them to find all the elements of the polytope/lattice. While the geometrical interpretation only holds for polytopes, the statements regarding set intersections of minimal elements hold true for all SDL's. In other words, each set of minimal elements that is shared by more than one maximal face constitutes an element of the lattice. The Reconstruction Algorithm finds for these sets of minimal elements.

Minimal element labels are used to discard superfluous elements. Elements found by infimum search can be superfluous in two ways: they can be duplicates as outlined above, or they can in fact not be elements of the lattice. In the first case, the check is simple: as a new element g is found, check if $V_W(g)$ is identical to any of $\{V_W(f_i)\}$, where $\{f_i\}$ are the infima already found. The second superfluous case requires more explanation. Although in general, any subset of elements of a lattice has an infimum in the lattice (this is a defining property of lattices), in an SDL the least face (the single minimal element that precedes all others) is removed. Therefore, there is no guarantee that *every* subset of the lattice has an infimum. For some sets of maximal elements, the infimum would

be this least face, but it is removed from consideration. For such sets of maximal elements there is no minimal element common to all maximal elements in the set ($\bigcap_{i\in\mathbb{I}} V_W(F_i) = \emptyset$). At this point, an example of each these two superfluous cases is in order.

A square pyramid provides a simple example of the first case mentioned above. Consider intersecting three of the triangular faces ($A - C$), and then all four of them ($A - D$), as shown in Figures 2.5(a) and 2.5(b). Both of these combinations of maximal elements lead to the same infimum (the top vertex e). Because this structure is an SDL, it is by definition supremum-filled. Therefore, we can exchange each of the maximal elements (faces) for minimal element labels: $A \to \{c,d,e\}$, $B \to \{a,c,e\}$, $C \to \{b,d,e\}$, and $D \to \{a,b,e\}$. Observe how the minimum element labels show that these two infima are the same.

$$\inf(\{A,B,C\}) = \{c,d,e\} \cap \{a,c,e\} \cap \{b,d,e\} = e$$

and

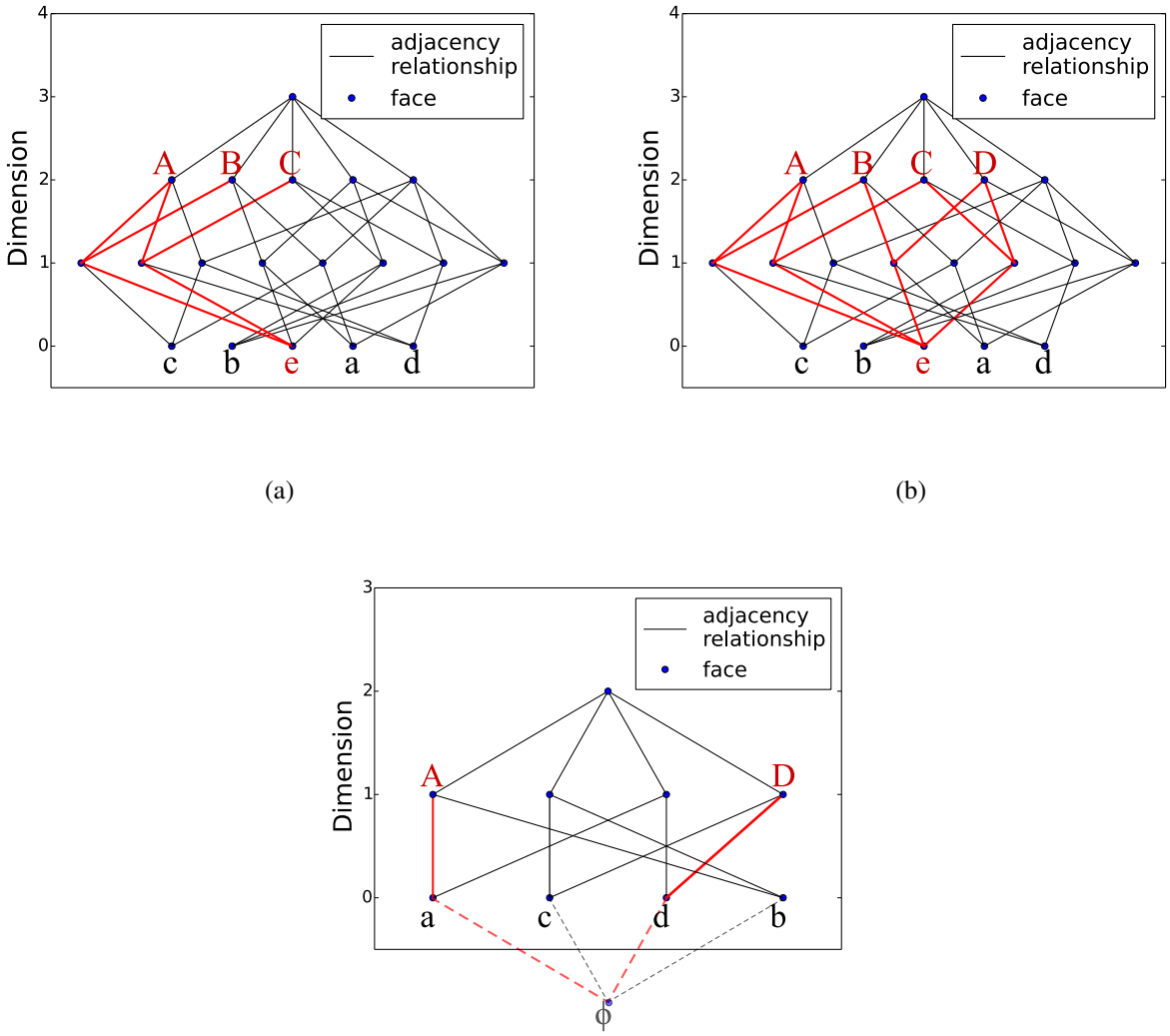
$$\inf(\{A,B,C,D\}) = \{c,d,e\} \cap \{a,c,e\} \cap \{b,d,e\} \cap \{a,b,e\} = e$$

Thus, by using the minimal element labels, the algorithm realizes that these two elements are duplicates, and combines them.

The second example of superfluous elements is shown in Figure 2.5(c). In this example, the boundary complex is topologically a square. Maximal faces are therefore one-dimensional edges. The edges A and B are on opposite side of the square, and therefore share no vertices (they have no infimum). Their infimum would have been the least face, but it has been removed from the SDL, as mentioned in Sec. 2.2. Yet this is no problem. Again using minimal element labels, we have $A \to \{a,b\}$ and $D \to \{c,d\}$. Thus, $\inf(\{A,D\}) = \{a,b\} \cap \{c,d\} = \emptyset$. All such cases can be recognized when the set intersection yields the empty set.

Finally, I include a result here whose importance will be clarified in the next section.

Theorem 5—The Minimal Model Theorem. Let W be infimum-filled. Given $\{a_i\}_{i\in\mathbb{I}} \subset W$



(a)

(b)

(c)

Figure 2.5 Figures (a) and (b), both Hasse diagrams of a square pyramid, show how the infimum of different sets of elements can be identical. This is clear from using minimal element labels. Exchanging each maximal face for its set of minimal elements gives $\inf(\{A,B,C\}) = \{c,d,e\} \cap \{a,c,e\} \cap \{b,d,e\} = e$ and $\inf(\{A,B,C,D\}) = \{c,d,e\} \cap \{a,c,e\} \cap \{b,d,e\} \cap \{a,b,e\} = e$. Figure (c) shows how the infimum of two maximal faces can lead to the least face, which every SDL lacks. The minimal element labels again reveal this: $\inf(\{A,D\}) = \{a,b\} \cap \{c,d\} = \emptyset$.

(where $\mathbb{I}$ is a counting set),

$$\text{if } x = \sup(\{a_i\}), \text{ then } x = \inf(\bigcap_{i\in\mathbb{I}} F_W(a_i)).$$

In summary, the SAI is sufficient to find and uniquely identify all elements of a supremum- and infimum-filled lattice. The fact that the lattice is infimum-filled implies that every element can be discovered as the infimum of some set of the maximal elements, which are given in the SAI. The fact that the lattice is supremum-filled allows each element to be uniquely labeled by the minimal elements preceding it. From the SAI, each maximal element can be thus labeled. An infimum of a set of maximal faces inherits its label from the maximal faces. This allows for identification of superfluous elements—duplicates and sets of maximal faces whose infimum would have to be the least face. At this point, the natural next step is to devise an algorithm for finding all of the elements of an SDL from the SAI. However, as Theorem 5 hints, there is more to the story. The interpretations of these theorems in the context of MBAM motivate the development of an additional algorithm. Therefore, the meanings of these results for models and model reduction are presented next.

2.4 Connections to MBAM

This section interprets Theorems 1-5 in the context of MBAM. The hope is to illustrate the significance and power of these results. Though SDL's are not necessarily polytopes, solely for the sake of simplifying language I shall assume moving forward that the boundary complex in question is an abstract polytope. I stress that this assumption does not simplify the mathematics. In fact, the results and eventual solutions (next section) apply to SDL's in general—that is, they apply to any SDL, polytope or not. In particular, by making this assumption, I am avoiding the (explanatory—not mathematical) difficulty of defining a rank function for every SDL in order to speak of dimension of elements. In general, the statements made in this section can be converted

to equivalent statements for lattices by replacing corresponding terms (e.g., vertices becomes minimal elements, as long as the unique least face is already removed from consideration—even here, the lattice language is cumbersome). With this in mind, the meaning of several polytope parts and properties are given in the language of MBAM, followed by the meanings of Theorem 4 (The Equivalence Theorem) and Theorem 5 (The Minimal Model Theorem).

Given a polytope arising from a model manifold, each face of the polytope corresponds to a simplified model. It is simplified in the sense that it necessarily has fewer parameters than the original model. In fact, the dimension of the face is exactly the number of structurally identifiable parameters needed for the simplified model [4]. Moreover, this model leads to a family of models with fewer parameters—those that correspond to the lower-dimensional boundary cells contained in the face. In particular, vertices (having zero dimension) represent zero-parameter models. These are extreme behaviors that a model can achieve. The vertices of a given face denote all possible extreme behaviors achievable by its model. In this way, each face of the polytope (and thus the whole polytope) represents a family of models, each with as many parameters as the dimension of their corresponding face.

Importantly, no pair of these models (or faces) are identical—even those of the same dimension. Consider two faces Y_1 and Y_2, both of dimension d. Though both corresponding models y_1 and y_2 have d parameters, the particular parameters will not be the same. The limits taken to arrive at the Y_1 boundary cell of the manifold are different than those necessary to arrive at the Y_2 boundary cell. Therefore, each model has different parameters removed from the fully-parametrized model. However, a better proof of this statement relies on Theorem 4, which has a very intuitive interpretation when applied to polytopes.

In the context of polytopes, the first statement in Theorem 4 says that each face in a polytope (that is an SDL) is uniquely identified by the vertices it contains—no two faces share the exact same set of vertices. (Likewise, the second statement in Theorem 4 says that each face is uniquely

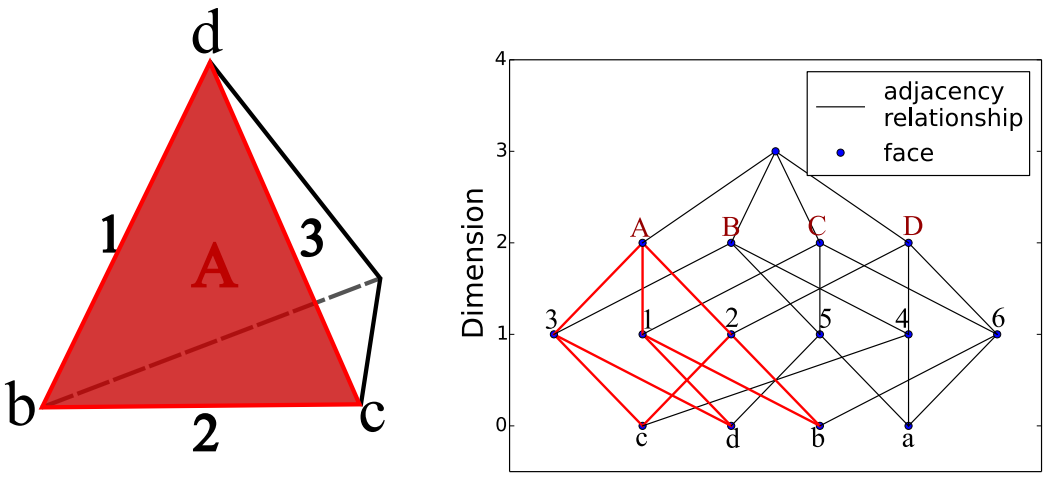
identified by the set of maximal faces to which it belongs; the first statement is the more useful here.) For our example, Y_1 must have at least one vertex not contained by Y_2, or they would be the same face. Thus, y_1 much achieve at least one extreme behavior that y_2 cannot, and the models are distinct. In the next section, this understanding of Theorem 4 will become the basis of the Reconstruction Algorithm.

The significance of Theorem 5 is also more easily illustrated in the language of polytopes. Theorem 5 (The Minimal Model Theorem) essentially says that each face corresponds to the simplest model that achieves all of the extreme behaviors corresponding to the vertices contained in the face. Note that this theorem also says that each face is the supremum of its set of vertices. The Minimal Model Theorem is powerful: if I want the minimal model that gives a certain set of extreme behaviors, I simply choose the model given by the lowest-dimensional face containing all of the corresponding vertices. I develop an algorithm for such a procedure in the next section. This algorithm may lead to discovery of new extreme behaviors—vertices contained in the face but not part of the original set of desired extreme behaviors. This idea is further explored in section 3.3.

2.5 Devising the Algorithms

As outlined in Sec. 2.3, the fact that V_W is injective for an SDL suggests that a natural identifier for each face is the (unique) set of vertices it contains. Thus, each vertex is represented computationally by an index, and each face by the set of indices corresponding to the vertices it contains. For the sake of brevity, I will term the set of indices of the vertices preceding a face f the vertex representation of f.

The fact that an SDL is infimum-filled suggests a combinatorial search for infima. Theorem 4 says that every face besides the maximal faces is an infimum of some subset of the maximal faces. Each maximal face F_i is represented by $V_W(F_i)$ (its vertex representation). Again as outlined in Sec.



(a) (b)

Figure 2.6 A possible boundary complex (a tetrahedron), shown as a two-dimensional projection (a) and a Hasse diagram (b). In general, the boundary cells of the model manifold are sub-manifolds that correspond to reduced models. An $(N-1)$-dimensional boundary cell corresponds to a reduced model with $N-1$ parameters. Likewise, the dimension of any boundary cell gives the number of structurally identifiable parameters in the model it represents. In particular, a 0-dimensional boundary cell (a vertex) gives a 0-parameter model—a phase or extreme behavior. Here, the 2-dimensional boundary cell A corresponds to a sub-manifold (a triangle), whose Hasse diagram (a subset of the full Hasse diagram) is highlighted in red. This corresponds to a 2-parameter model that, by taking different limits, can be reduced to the three distinct 1-parameter models 1, 2, and 3. The model represented by A achieves the extreme behaviors a, c, and d, but can not achieve the behavior b. This is characteristic of the general case: the reduced models achieve fewer extreme behaviors, but are simpler and easier to interpret.

2.3, for each possible combination of maximal faces, the vertex representation of their infimum is found by taking the set intersection of all the vertex representations. That is,

$$V_W(f) = \bigcap_{i \in \mathbb{I}} V_W(F_i), \tag{2.5}$$

where f is the infimum. Finding the infima of all possible combinations of the maximal faces guarantees that every face is found.

For an SDL, it is guaranteed by property of lattices that every combination of maximal faces has as an infimum that either 1) is a face of the polytope or 2) would have been the least face, which has been removed from the lattice. In the latter case, the vertex representation is the empty set because the set of maximal faces has no common vertex, as outlined in Sec. 2.3, specifically in Fig. 2.5(b).

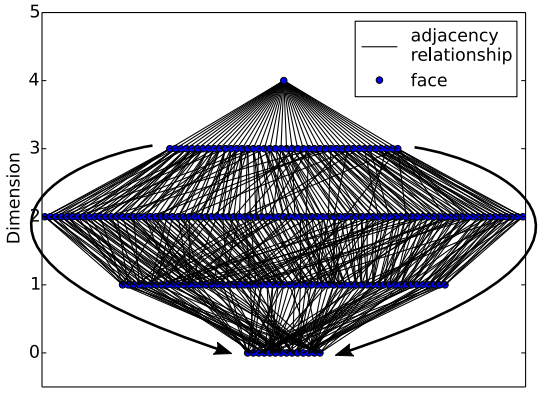
In order to save memory and speed up operations, a binary sparse matrix representation is used. Although it does not have the visual clarity of a Hasse diagram, this representation contains all of the same information and is vastly preferable in a computational context. In this representation, each face f is a column of length N_V, where N_V is the total number of vertices $\{v_i\}$. Then, if N_f is the total number of faces, the binary matrix representation is a $N_V \times N_f$ Boolean matrix allf with entries

$$\text{allf}_{ij} = \begin{cases} \text{true} & \text{if } v_i \prec f_j \\ \text{false} & \text{otherwise} \end{cases}$$

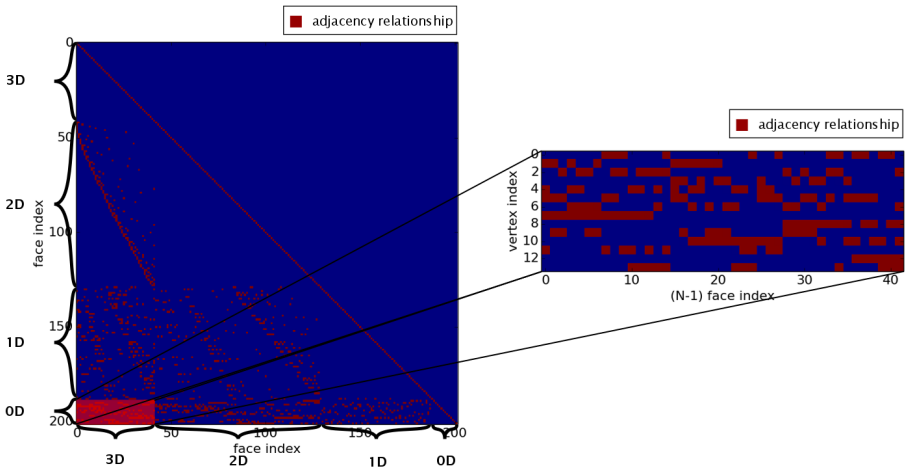
.

I now present the main boundary complex reconstruction algorithm. The algorithm "pares down" the number of combinations it checks by avoiding recursion once a combination leads to the least face. In general, the number of elements in a boundary complex grows combinatorially with dimension. Yet, if the boundary complex is a Superficially Determined Lattice, the SAI is sufficient to infer all the additional elements between the highest- and lowest-dimensional elements, and the connections among them. As Figure 2.7 illustrates, the Reconstruction Algorithm obtains a wealth

of information, given only the SAI.



(a)



(b)

Figure 2.7 In general, the number of possible elements in a boundary complex grows combinatorially with dimension. Yet, if the boundary complex is a Superficially Determined Lattice, the SAI is sufficient to infer all the additional elements between the highest- and lowest-dimensional elements, and the connections among them. Using the Reconstruction Algorithm, a wealth of information can be obtained given only minimal information (the SAI) initially. The SAI contains the information about the adjacency relationships between the highest-dimensional boundary cells (in this case, three-dimensional) and the lowest-dimensional ones (the zero-dimensional ones).

Algorithm 1 —Reconstruction Algorithm: The Reconstruction Algorithm infers the intermediate elements of the differential topology of a boundary complex from the SAI. Each boundary element is represented by a Boolean vector indicating the vertices it contains. Iterating through combinations of maximal elements, the algorithm finds new elements by intersecting their sets of vertices. Note in particular the *if* statement preceding the recursion. With this statement, the algorithm pares down the number of combinations it checks by avoiding recursion once a combination yields an empty intersection (meaning the maximal faces intersected share no vertices). New elements are stored as columns of a sparse Boolean matrix $allf$.

1: $N_V =$ number of minimal elements in W.

2: $N_F =$ number of maximal elements in W.

3: $\mathcal{S}$ is a $N_V \times N_F$ Boolean matrix with the elements

$$\mathcal{S}_{ij} = \begin{cases} \text{true} & \text{if } V_i \prec F_j \\ \text{false} & \text{otherwise} \end{cases}$$

4: N is the maximum length of a chain in W (see note 3).

5: comb = Boolean vector of N_F falses.

6: nstart = 1

7: allf = copy($\mathcal{S}$)

8: **procedure** FINDALLINFIMA($\mathcal{S}$, N,N_V,N_F, comb, nstart, allf)

9: **if** sum(comb) $> N$ **then**

10: **return** allf

11: **end if**

12: **for** n between nstart and N_F **do**

13: nextcomb=copy(comb)

14: nextcomb[n]=true

15: cols = $\{\text{k} \mid \text{nextcomb[k]} = \text{true}\}$

Algorithm 1 —Reconstruction Algorithm (continued)

f = a length N_V Boolean vector such that

$$f_i = \begin{cases} \text{true} & \text{if } \mathcal{S}_{ij} == \text{true } \forall\ j \in \text{cols} \\ \text{false} & \text{otherwise} \end{cases}$$

if $\exists\ i \mid f_i$ = true **then**
    rows,columns=size(allf)
    $j = 1$
    new = true
    while ($j \leq$ columns) and new **do**
        if allf[all , j]=f **then**
            new=false
        end if $j = j + 1$
    end while
    if new **then**
        add f as a new column of allf
    end if
    allf=FINDALLINFIMA($\mathcal{S}$, N, N_V, N_F, nextcomb, n+1, allf)
  end if
 end for
return allf
end procedure

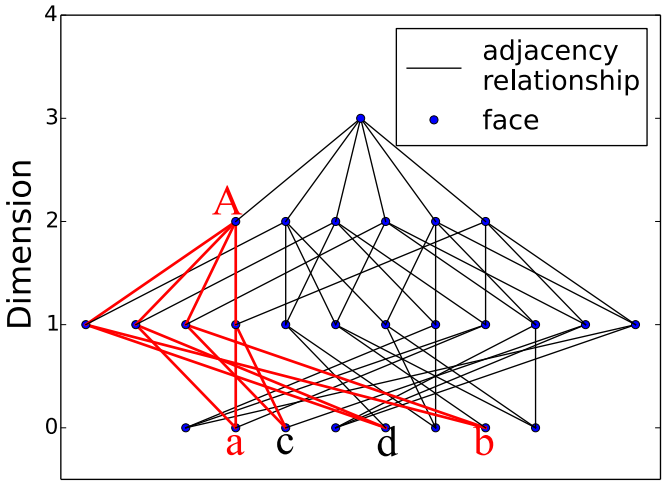


Figure 2.8 Given two vertices *a* and *b* (which correspond to desired extreme behaviors), the Minimal Model Algorithm finds their supremum (*A*) and reconstructs the intermediate topology (highlighted in red). In this case, the minimal model has a manifold that is topologically equivalent to a square. No 1-parameter model exists that interpolates between the two extreme behaviors.

In order to leverage the power of the Minimal Model Theorem, I devised an algorithm for finding the supremum of a set of vertices, and inferring the topology of all the elements preceding this supremum. The algorithm essentially yields minimal models given desired extreme behaviors. The Hasse diagram in Figure 2.8 illustrates an example in which the desired extreme behaviors correspond to the vertices *a* and *b*. The Minimal Model Algorithm finds their supremum (*A*) and reconstructs the intermediate topology (highlighted in red). In this case, the minimal model has a manifold that is topologically equivalent to a square. No 1-parameter model exists that interpolates between the two extreme behaviors. Note that, although the entire Hasse diagram is shown, the algorithm avoids reconstruction of the entire boundary complex. The Minimal Model Algorithm searches for only those faces and adjacency relationships connected in red. Therefore, this method can be employed even for problems that are too large (combinatorially) to reconstruct the entire manifold topology.

Algorithm 2 —Minimal Model Algorithm: Given a set of vertices, the Minimal Model Algorithm finds their supremum and, in a similar manner to the Reconstruction Algorithm, infers the intermediate topological structure between the supremum and the vertices. Vertices correspond to extreme behaviors. Therefore, given a set of desired extreme behaviors, the algorithm essentially yields the minimal (minimum-parameter) model that achieves all of those extreme behaviors. By only discovering the topology of a sub-manifold, the Minimal Model Algorithm avoids reconstruction of the entire boundary complex. This algorithm is therefore useful in cases that the problem is too large computationally (*i.e.* the boundary complex has too many maximal elements to check all the necessary combinations).

1: $N_V =$ number of minimal elements in W.

2: $N_F =$ number of maximal elements in W.

3: $\mathcal{S}$ is a $N_V \times N_F$ Boolean matrix with the elements

$$\mathcal{S}_{ij} = \begin{cases} \text{true} & \text{if } V_i \prec F_j \\ \text{false} & \text{otherwise} \end{cases}$$

4: vs = Vector of the (integer) indices of the desired vertices

5: **procedure** FINDPRECEDING($\mathcal{S}$, N_V,N_F, vs)

6: comb = Boolean column vector of N_F falses.

7: vset= Boolean column vector of N_V falses.

8: vset[vs] = true

9: **for** i between 1 and N_F **do**

10: **if** (Sam[all, i] $\wedge$ vset)==vset **then**

11: comb[i] = true

12: **end if**

13: **end for**

Algorithm 2 —Minimal Model Algorithm (continued)

minitreepossible = true

if sum(comb)==0 **then**

minitreepossible = false

BaseFs = vector of integers from 1 to N_F

NbFs = N_F

else

BaseFs = indices of true values in comb

NbFs = length(BaseFs)

end if

return comb, BaseFs, NbFs, vset, minitreepossible

end procedure

$N_V =$ number of minimal elements in W.

$N_F =$ number of maximal elements in W.

$\mathcal{S}$ is a $N_V \times N_F$ Boolean matrix with the elements

$$\mathcal{S}_{ij} = \begin{cases} \text{true} & \text{if } V_i \prec F_j \\ \text{false} & \text{otherwise} \end{cases}$$

BaseFs = indices of maximal elements succeeding f (output of FINDPRECEDING)

NbFs = length of BaseFs (output of FINDPRECEDING)

vset = Boolean column vector of N_V elements (output of FINDPRECEDING)

comb = Boolean column vector of N_F elements (output of FINDPRECEDING)

nstart = 1

allf = copy($\mathcal{S}$)

procedure FINDMINIINFIMA($\mathcal{S}$, N_V,N_F, BaseFs, NbFs, vset, comb, nstart, allf)

f = Boolean column vector of N_V falses

Algorithm 2 —Minimal Model Algorithm (continued)

if sum(f $\wedge$ vset)==1 **then**

    return allf

end if

for i between nstart and N_F and i not in BaseFs **do**

    nextcomb = copy(comb)

    nextcomb[i]=true

    cols = indices of true values in nextcomb

    f = a length N_V Boolean vector such that

$$f_i = \begin{cases} \text{true} & \text{if } \mathcal{S}_{ij} == \text{true } \forall\ j \in \text{cols} \\ \text{false} & \text{otherwise} \end{cases}$$

    if $\exists\ i \mid f_i$ = true **then**

        rows,columns=size(allf)

        j=1

        new=true

        while (j$\leq$ columns) and new **do**

            if allf[all, j]==f **then**

                new=false

            end if

            j=j+1

        end while

        if new **then**

            add f as a new column of allf

        end if

        allf=FINDMINIINFIMA($\mathcal{S}$, N, N_V, N_F, BaseFs, NbFs, vset, nextcomb, i+1, allf)

Algorithm 2 —Minimal Model Algorithm (continued)

```
58:          end if
59:       end for
60:      return allf
61:  end procedure
```

Chapter 3

Model Simplification and Interpretation

This chapter demonstrates the utility of the Minimal Model Algorithm (Section 3.3) and the Reconstruction Algorithm (Section 3.2) in the Manifold Boundary Approximation Method (MBAM) with some statistical mechanics examples. First, now that the SDL is rigorously defined, examples of models in the SDL class are given. Next, two applications of the algorithms are discussed. Finally, I finish with a summary of the tools yielded by the algorithms for complex model analysis and further avenues of possible application.

3.1 Some SDL Models

Many types of models fall in the SDL class—to which the Reconstruction and Minimal Model Algorithms are applicable. In statistical mechanics, all exponential families are SDL models. This includes all models of the form given in Equation (2.2). Fortunately, most statistical mechanics models are in this family—noteworthy examples include restricted Boltzmann machines, Markov random fields, and cluster expansions.

There exist SDL models outside of statistical mechanics. An example in biology is the full enzyme substrate model presented by Transtrum *et al.* [4]. The ubiquity of SDL models is a matter

of further investigation—investigation which includes both the number of fields beyond statistical mechanics in which they are found and especially the frequency with which a model falls in the SDL class. Our research group is pursuing this avenue; we speculate that some (possibly large) subset of biological and neural network models are SDL models. We hope that such is the case, as the following applications would be greatly useful, especially in neuroscience.

3.2 Phase Diagrams

The Reconstruction Algorithm yields a global picture of how extreme behaviors of a model are related to one another. In fact, knowing the differential topology of the boundary complex—the information contained in a Hasse diagram—and its MBAM interpretation, we can construct a type of phase diagram for each model. The meaning of phase here is not so important—think of it as referring to extreme behavior. It is the phrase *phase diagram* that is illuminating. Just as a phase diagram illustrates how closely behaviors are related—by a phase transition or through some higher-order coexistence point—the global understanding gained through the Reconstruction Algorithm allows us to deduce analogous relationships among extreme behaviors of a complex model.

Consider as an example this model of a binary alloy: a two-parameter cluster expansion for atoms on an fcc (crystal—not algebraic) lattice,

$$H = -J_1 \sum_{n.n.} s_i s_j - J_2 \sum_{n.n.n.} s_i s_j. \tag{3.1}$$

Here "n.n." and "n.n.n." denote nearest neighbors and next-nearest neighbors (respectively), and the $\{s_i\}$ are the spin magnetic moments of the atoms. The two interaction parameters are J_1, the interaction strength between the nearest neighbors, and J_2, that between the next-nearest neighbors. The five ground states of this model are known, and so are their mutual relationships. It is therefore possible to construct a phase diagram as I have done below (see Figure 3.1(a)). The bulk phases

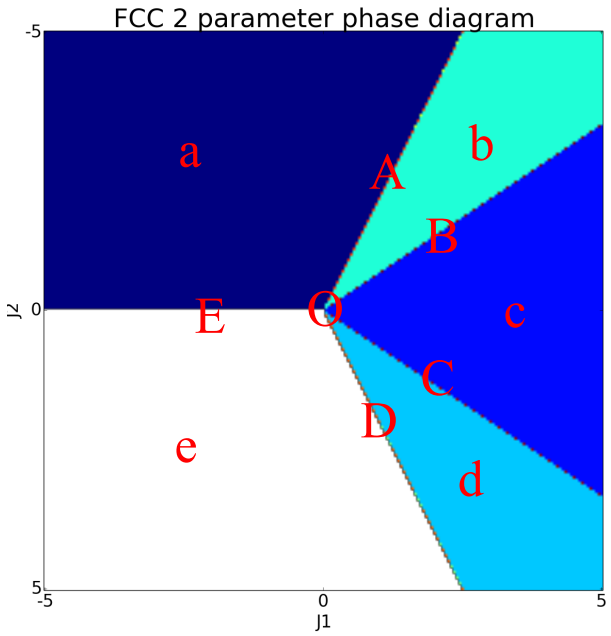
or ground states (shades of blue/white—lowercase letters a-e) are separated by phase transitions (lines—capital letters A-E), which meet at the coexistence point O. By tuning J_1 or J_2 (moving along the axes), any of the phases can be achieved.

Although for this model this information is known, in the general case these relationships among extrema are not known *a priori*. Moreover, they become harder to deduce as models becomes more complex (e.g., as they gain parameters). Fortunately, the Reconstruction Algorithm can yield this information directly, as this example illustrates.

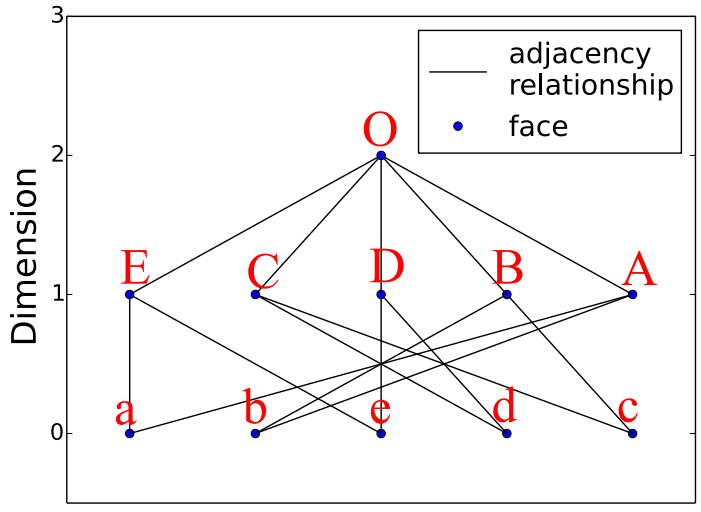
The model manifold for the cluster expansion (3.1) is diffeomorphic to a pentagon, whose Hasse diagram can be seen in Figure 3.1(b). Each of the faces (in this case, vertices, edges, and the pentagon as a whole) have been labeled according to the corresponding element of the phase diagram. Each vertex of the pentagon corresponds to one of the ground states (phases). Similarly, each one-dimensional face (edge) corresponds to a one-parameter model that interpolates between two phases—a phase transition. Finally, the two-dimensional face O corresponds to the full (two-parameter) model, which interpolates between all five phases.

Now consider for example, fixing $J_1 = 0$ and adjusting J_2 only. This is now a one-parameter model that moves between the phases a and e, namely the phase transition E. On the Hasse diagram, the edge that connects the vertices a and e is E. In other words, the Hasse diagram reveals that there exists (in some physically interesting limiting approximation) a one-parameter model that interpolates between a and e. Likewise, consider moving from a to d on the phase diagram. Exactly three (non-repetitive) paths exist: clockwise (to e through E, then to d through D), counterclockwise (to b through A, then c through B, then d through C), or through the coexistence point O. The reader may check that the same is true of the Hasse diagram. In fact, all of the qualitative information in the phase diagram can be deduced directly from the Hasse diagram, which in turn is given by the Reconstruction Algorithm.

The Reconstruction Algorithm can be employed to classify the extreme behaviors of myriad



(a)



(b)

Figure 3.1 A phase diagram 3.1(a) and a Hasse diagram 3.1(b) for a 2-parameter cluster expansion (nearest neighbors and next-nearest neighbors) on an fcc crystal lattice. On the phase diagram, the coexistence point is labeled *O*. Each phase transition (a boundary that interpolates between two phases or extreme behaviors) is labeled with a capital letter (*A* through *E*). Each phase is labeled with a lowercase letter (*a* through *e*). The corresponding boundary elements of the model manifold are labeled on the Hasse diagram. Note how all the information in the phase diagram is also present in the Hasse diagram.

complex systems. As long as the model is an SDL model, the algorithm yields a complete picture of the manifold topology, which can be used as shown to discover relationships among extreme behaviors. Imagine the power of this knowledge in systems biology or neuroscience. Those extreme behaviors that are adjacent on such a phase diagram are only separated by the adjustment of one parameter. Thus, if only some of the extreme behaviors are desirable, it is possible that only a subset of the parameters is necessary to interpolate between them. This is the subject of the next section.

3.3 Minimal Models

The Minimal Model Algorithm yields minimum-parameter models to achieve a desired set of extreme behaviors. The basics rudiments of this algorithm as well as its power have already been discussed in Sections 2.4 and 2.5. These points nevertheless merit repeating, further discussion, and most importantly clear illustration. I shall accomplish this by applying the algorithm to a model whose manifold topology is that of a four-dimensional hypercube (a cube generalized to four dimensions—see Figure 3.2).

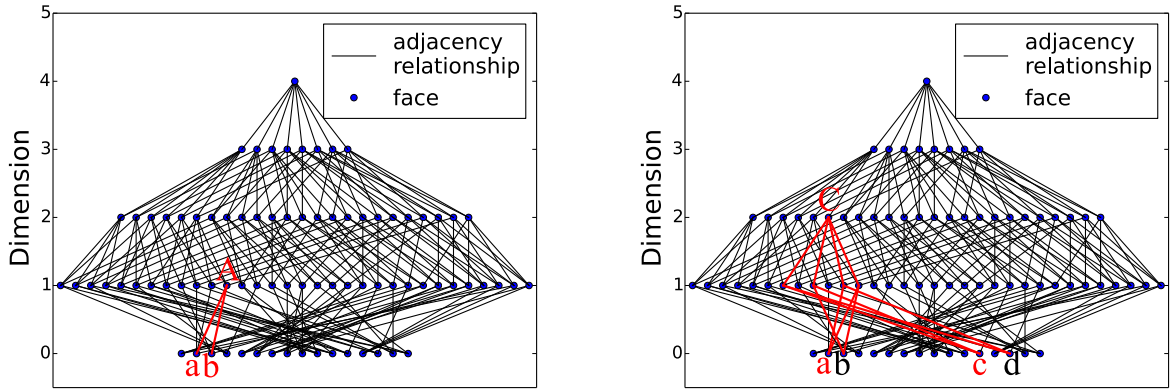
Consider a set of extreme behaviors that we, the designers of a reduced model, wish to achieve using a complex model. Recall that these extreme behaviors correspond to a set of vertices of the boundary complex of the model manifold (our hypercube). In fact, because this complex model is (by assumption) an SDL model, our set of vertices is guaranteed to have a unique supremum—a unique face that contains all the vertices and has a lower dimension than any other face that contains them all. In principle, we are done. The supremum is actually some boundary cell of the model manifold that corresponds to the exact reduced model we want. There are a few more practical steps, however.

We next transform the boundary cell into parameter space, and find the parameter directions

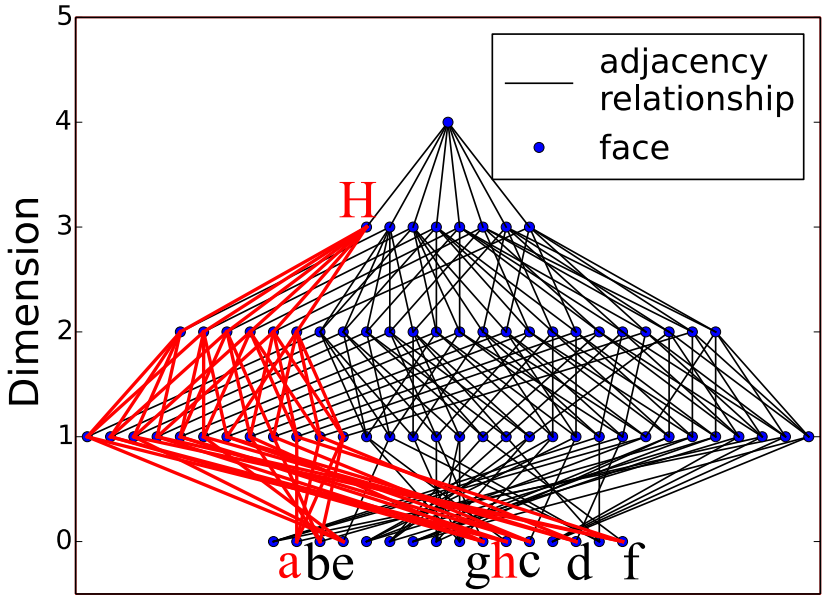
that span it. Recall from Section 1.3 that these *new* parameters are in general non-linear combinations of the model's original parameters (ratios, products, etc.). Further, the basis for the boundary cell (call it the *final* parameter basis) may include some linear combinations of the new parameter directions. Nevertheless, the final parameter basis can be found. These final parameters are those needed for our minimal model—any more and the model would be overly complicated, any fewer and it would not achieve all of the desired behaviors. In practice, however, there are more details that in fact give additional insight upon investigation. To this end, let us assume that there are two desired behaviors.

We know that these extreme behaviors correspond to two vertices and that their supremum exists, but finding it is another matter. As discussed in Section 2.5, the Minimal Model Algorithm leverages the power of the Minimal Model Theorem to find a supremum and *all* elements that precede the supremum. With our example, our two vertices *could* be connected by a simple line segment (an edge), in which case they are the only two elements that precede their supremum. Suppose, however, that this is not so. The vertices could be on opposite corners of a two-dimensional square face of the hypercube, or across the longest diagonal of one of the three-dimensional cubes that make up the boundary. Figure 3.2 illustrates each of the three possibilities. The desired extreme behaviors (red) are connected (red-highlighted lines) through their supremum, in some cases to other extreme behaviors [as in Figs. 3.2(b)) and 3.2(c)].

In these latter cases, the face that contains the two given vertices (the supremum) contains additional vertices (two more in the case of the square, six for the cube). These correspond to additional extreme behaviors that our minimal model can achieve, yet that we have not considered. They may even be as-of-yet unobserved behaviors of that system. Conversely, consider the case that one of our vertices corresponds to an observed ground state of a system, and the other corresponds to a hypothesized or theoretical extreme behavior not yet achieved experimentally. We now have the simplest possible model that relates the two extreme behaviors—we can tune the parameters to



(a) (b)



(c)

Figure 3.2 Hasse diagrams representing various minimal models, where the full model manifold is topologically a hypercube. Top left (a): the minimal model for vertices a and b is a line segment (A), meaning there is a 1-parameter model that interpolates between the two extreme behaviors represented by a and b. Top right (b): The minimal model for vertices a and c is (topologically) a square (C). Thus, no interpolating 1-parameter exists, but a 2-parameter model connects these extreme behaviors, along with two others (represented by vertices b and d). Bottom (c): The minimal model for vertices a and h is (topologically) a cube (H). Thus, no 1- or 2-parameter model exists that connects these extreme behaviors, but they can be connected through a 3-parameter model which also connects six other extreme behaviors ($b-g$).

take the system from the observed state to the new state.

Recall that the Minimal Model Algorithm also avoids reconstruction of the entire manifold topology. Thus, for problems that are too combinatorially large, this algorithm can still be used to reconstruct pieces of the manifold topology and thus yield models that relate important extreme behaviors. In this regard, this algorithm is even more widely applicable than the Reconstruction Algorithm.

3.4 Conclusion

Information topology is a useful tool for understanding complex models. A model defines a mapping between parameters and predictions, and is therefore naturally understood as a manifold. Because there is a limit on the range of possible predictions, models are most often bounded. The boundary is commonly divided into boundary cells, the union of which cells is called the boundary complex. These boundary cells correspond to physically interesting limiting approximations in the model. Together, they provide a global characterization of the parameter space. Further, the boundary cells are a feature of the differential topology, meaning they are invariant to diffeomorphic changes. By ascertaining the topology of the boundary complex, information can be extracted about the boundary cells and thus about the underlying model.

Many models admit the reconstruction of the boundary complex from minimal information. This minimal information is termed in this paper the Superficial Adjacency Information (SAI). It constitutes the adjacency relationships between the highest- and lowest-dimensional boundary cells of the manifold. For statistical mechanics models in the exponential family, the convex hull of the Π matrix can be used to obtain the SAI. In Chapter 2 (specifically Sections 2.2 and 2.3), the necessary and sufficient conditions that a boundary complex must meet in order to be reconstructed completely from the SAI are defined. Most statistical mechanics models and possibly a much wider

range of models meet these conditions. Such a model is here termed a Superficially Determined Lattice (SDL) model.

Given one of these models, the Reconstruction Algorithm generates a family of models—simplified to varying degrees—by reconstructing the boundary complex of the model manifold. The dimension of each boundary cell corresponds to the number of structurally identifiable parameters in the corresponding model. In particular, zero-dimensional boundary cells correspond to extreme behaviors that the model can achieve. Thus, the complete picture of the boundary complex gives all possible extreme model behaviors and families of models to relate and describe them.

The Minimal Model Algorithm characterizes the relationship between extreme behaviors and constructs minimal models to connect those behaviors. Given a set of desired extreme behaviors, the algorithm finds the model with fewest parameters that achieves all of the extreme behaviors. The Minimal Model Algorithm has the advantage that it can be applied to problems too combinatorially large for the Reconstruction Algorithm. It does not need to reconstruct the entire boundary complex. Application of this algorithm could lead to the discovery of new extreme behaviors.

3.5 Further Work

As mentioned in Section 3.1, our research group is currently engaged in an investigation of additional model types (outside of statistical mechanics) that fall in the SDL class. We suspect that, in the majority of the cases to which we desire to apply the solution, the model is an SDL model.

In the future, the Reconstruction and Minimal Model Algorithms can be applied to construct alloy phase diagrams and to classify behavioral regimes of non-statistical mechanics models. Akin to the example given in Sec. 3.2, phase diagrams can be constructed for more complex, more interesting, and/or less-well-understood alloys. Likewise, the two algorithms can be used to give either a partial or complete picture of the manifold topology of other models. Specifically in systems bi-

ology and neuroscience, zero-parameter models correspond to behavioral regimes. The algorithms can be applied to classify these behavioral regimes and investigate their relationships through minimal models. This may lead to the discovery of as-yet-unobserved behaviors. Together, these algorithms are essential and powerful tools for harnessing the power of the Manifold Boundary Approximation Method formalism.

Appendix A

Proofs

As noted in Section 2.3, because of the symmetry of posets and lattices, every lemma and theorem in this section has an equally true dual theorem whose proof is practically identical. Therefore, the reader should be prepared to follow not only each statement presented, but the dual statements as well. The duals of select lemmas and theorems will be especially important for the proofs of Theorems 4 and 5. In the following discussion, X is a poset where each chain is finite.

Definition 1. We say X is *supremum-filled* if for all non-minimal x in X, there exist $a,b \in X$ such that a is incomparable to b ($a \nsim b$) and the supremum of $\{a,b\}$ is x [$\sup(\{a,b\} = x)$].

Definition 2. Similarly, we say X is *infimum-filled* if for all non-maximal x in X, there exist $a,b \in X$ such that $a \nsim b$ and the infimum of $\{a,b\}$ is x [$\inf(\{a,b\} = x)$].

Definition 3. Given some non-minimal $x \in X$, let $\text{Base}(x) = \{y \in X \mid y \text{ is an immediate predecessor of x}\}$. Similarly, given some non-maximal $x \in X$, let $\text{Cover}(x) = \{z \in X \mid z \text{ is an immediate successor of x}\}$.

Theorem 1. X is supremum-filled if and only if for all non-minimal $x \in X$, there exist $a,b \in \text{Base}(x)$ such that $\sup(\{a,b\}) = x$. *Proof.* $\Longleftarrow$) Trivial.

$\Longrightarrow$) Given $x \in X$ non-minimal, choose $c,d \in X$ such that $c \nsim d$ and $\sup(\{c,d\}) = x$. Note that since each chain in X in finite, there exists some $a,b \in \text{Base}(x)$ such that $a \succeq c, b \succeq d$. Given any

upper bound y of $\{a,b\}$, $y \succeq a$ and $y \succeq b$, which implies $y \succeq c$ and $y \succeq d$. Since $\sup(\{c,d\}) = x$, we have that $y \succeq x$. So, $\sup(\{a,b\}) = x$.

Corollary. If X is supremum-filled , then for all non-minimal $x \in X$, $\sup(\text{Base}(x)) = x$.

Proof. Given $x \in X$ non-minimal, by the previous theorem we choose $a,b \in \text{Base}(x)$ such that $\sup(\{a,b\}) = x$. Any upper bound y of $\text{Base}(x)$ is also an upper bound of $\{a,b\}$, hence the desired conclusion. ■

Definition 4. Recall from Section 2.2 that given any poset X with finite chains, $m(X)$ is the set of minimal elements of X and $M(X)$ is the set of maximal elements of X.

Definition 5. Recall also from Section 2.2 that given X as in Definition 4, the vertex function $V_X : X \to \mathcal{P}(m(X))$ is defined by $y \mapsto \{x \in m(X) \mid x \prec y\}$. Likewise, the maximal face $F_X : X \to \mathcal{P}(M(X))$ is defined by $y \mapsto \{x \in M(X) \mid y \succ x\}$.

From this point on X refers to a lattice (rather than a poset) where all chains are finite. Let $W = X - [m(X) \cup M(X)]$. We will see below that this gives the equivalences of Theorem 4. That is,

$$W \text{ is supremum-filled} \iff V_W \text{ is injective} \iff \text{for all } x \in W, \sup(V_W(x)) = x$$

and

$$W \text{ is infimum-filled} \iff F_W \text{ is injective} \iff \text{for all } x \in W, \inf(F_W(x)) = x.$$

The proofs now are presented in sequence to prove Theorem 2, then Theorem 3, and finally arrive at the proof of Theorem 4. The proofs are divided into lemmas so as to be taken in steps, with the numbering of each lemma corresponding to the theorem it helps to prove. Thus, we begin with Lemma 2.1.

Lemma 2.1. If X is nonempty, then X has a greatest and a least element.

Proof. We show that X has a greatest element. The proof that X has a least element is entirely analogous.

Since each chain in X is finite, we can choose some maximal element $x \in X$. Given any element

$y \in X$, the set $\{x,y\}$ has a supremum in X because X is a lattice. Therefore, let $z = \sup(\{x,y\})$. Note that $z \succeq x$ and, since x is maximal, it follows that $z = x$. Thus, $y \preceq x$, and x is indeed the greatest element of X. ■

Lemma 2.2 W is supremum-filled if and only if for each $x \in W$, there exists some $S \subset m(W)$ such that $\sup(S) = x$.

Proof. $\Longleftarrow$) Let $x \in W$ be non-minimal. Then $\text{Base}(x)$ is non-empty. We first show that $|\text{Base}(x)| > 1$.

Assume by way of contradiction that $\text{Base}(x) = \{y\}$ for some y. Then, choose some $S \subset m(W)$ such that $\sup(S) = x$. Note that $S \subset V_W(x)$, and that $V_W(x) = V_W(y)$. Since y is an upper bound of $V_W(y)$, y is also an upper bound of S, contradicting the fact that $\sup(S) = x$. Thus, $\text{Base}(x)$ has at least two elements.

Choose $z_1, z_2 \in \text{Base}(x)$. Since X is a lattice, $\sup(\{z_1,z_2\})$ exists, and it is clear that this supremum must be x. Thus, W is supremum-filled.

$\Longrightarrow$) Given $x \in W$ non-minimal, we first show that if $\sup(\{y_i\})_{i\in\mathbb{I}}$ (where $\mathbb{I}$ is a counting set) and $y_j = \sup(\{z_1,z_2\})$ for some j, then $x = \sup(\{y_i\}_{i\neq j} \cup \{z_1,z_2\})$. This is immediate, since x is clearly an upper bound of this set and given any other upper bound y of this set, $y \succeq z_1, z_2 \implies y \succeq y_i$ for all $i \in \mathbb{I} \implies y \succeq x$.

Now, since W is supremum-filled , we can write $x = \sup(\{y_1,y_2\})$ for some y_1, y_2. Thus, by the above, we can apply the supremum-filled property to y_1 and y_2 individually. If y_1 and y_2 are non-minimal, we can apply the supremum-filled property to each element in $\text{Base}(y_1)$ and $\text{Base}(y_2)$ and we will have $x = \sup(\{z_1,\ldots,z_k\})$ (where $\{z_1,\ldots,z_k\} = \text{Base}(y_1) \cup \text{Base}(y_2)$ for some integer k). If these elements are also non-minimal, we can repeat the procedure. In fact, if we assume there is a uniform bound on the length of chains generated by this process, applying the supremum-filled property to the $\{z_i\}$ a finite number of times gives a set of minimal elements whose supremum is x. It thus remains to show that there is a uniform bound on the length of these chains. This is shown

in Lemma 2.2.1 below. ■

Lemma 2.2.1 Let Y be a poset with a greatest element (which exists by Lemma 2.1) where all chains are finite. If Base(y) is finite for all $y \in Y$, then there is a uniform bound on the length of chains in Y.

Proof. Assume to the contrary there is no uniform bound on the length of chains in Y. We show this gives an infinite chain in Y.

Let y_1 be the greatest element of Y. For each $y \in \text{Base}(y_1)$, let $l(y)$ be the supremum of the lengths of chains in Y with y as the greatest element in the chain. If each $l(y)$ is finite, then $max_{y \in \text{Base}(y_1)}(\{l(y)\})$ is a uniform bound on the length of chains in Y. Since there is no uniform bound, we choose some $z \in \text{Base}(y_1)$ such that $l(z) = \infty$, and let $y_2 = z$.

Similarly, since there is no upper bound to the length of chains beginning at y_2, we know that there is some $z' \in \text{Base}(y_2)$ such that $l(z') = \infty$. Set $y_3 = z'$ and repeat this process, letting y_3 take the place of y_2 and so on to get an infinite chain in Y. ■

Theorem 2. W is supremum-filled if and only if for all $x \in W$, $\sup(V_W(x)) = x$.

Proof. We show that given $x \in W$, if there exists some $S \subset m(W)$ such that $\sup(S) = x$, then $\sup(V_W(x)) = x$.

Note that $S \subset V_W(x)$, and that x is an upper bound of $V_W(x)$. Given any other upper bound y of $V_W(x)$, y is also an upper bound of S, and so $y \succeq x$. The claim now follows from Lemma 2.2 above.

■

Theorem 3. W is supremum-filled if and only if V_W is injective.

Proof. $\Longleftarrow$) Let $x \in W$ be non-minimal. Then Base(x) is non-empty. Note that Base(x) has at least two elements, since if Base$(x) = \{y\}$, we would have $V_W(x) = V_W(y)$, a contradiction. In an exactly identical fashion to the proof of the backwards implication ($\Longleftarrow$) of Lemma 2.2, we choose $z_1, z_2 \in \text{Base}(x)$ and note that their supremum is x.

$\Longrightarrow$) Choose $x, y \in W$ such that $V_W(x) = V_W(y)$. Applying Theorem 2, we see that $x =$

$\sup(V_W(x)) = \sup(V_W(y)) = y$. Therefore, V_W is injective. ■

As stated in Section 2.3, taking Theorems 2 and 3 together along with their duals, we have Theorem 4.

Theorem 4—The Equivalence Theorem. In summary, W is supremum-filled if and only if V_W is injective if and only if for all $x \in W$, $\sup(V_W(x)) = x$. Likewise, W is infimum-filled if and only if F_W is injective if and only if for all $x \in W$, $\inf(F_W(x)) = x$.

The remaining proof of Theorem 5 requires some elements of the proofs of Lemma 2.2, Theorem 2, and one additional lemma.

Lemma 5.1 Given $\{a_i\}_{i\in\mathbb{I}} \subset W$ (where $\mathbb{I}$ is a counting set), if $\sup(\{a_i\}) = x$, then $F_W(x) = \bigcap_{i\in\mathbb{I}} F_W(a_i)$.

Proof. It can be seen that $F_W(x) \subset \bigcap_{i\in\mathbb{I}} F_W(a_i)$, since for any $p \in F_W(x)$, $p \succeq x \implies p \succeq a_i$ for all a_i. Conversely, given $q \in \bigcap_{i\in\mathbb{I}} F_W(a_i)$, note that q is an upper bound for $\{a_i\}$, so $q \succeq x$ and $q \in F_W(x)$. Thus, $F_W(x) = \bigcap_{i\in\mathbb{I}} F_W(a_i)$. ■

Theorem 5—The Minimal Model Theorem. Let W be infimum-filled. Given $\{a_i\}_{i\in\mathbb{I}} \subset W$, if $\sup(\{a_i\}) = x$, then $x = \inf(\bigcap_{i\in\mathbb{I}} F_W(a_i))$.

Proof. By the dual of Lemma 2.2, we choose some $S \subset M(W)$ such that $\inf(S) = x$. Now, recall the following was shown in the proof of Theorem 2: given $x \in W$, if there exists some $S \subset m(W)$ such that $\sup(S) = x$, then $\sup(V_W(x)) = x$. By symmetry, the equally true dual statement is this: if there exists some $S \subset M(W)$ such that $\inf(S) = x$, then $\inf(F_W(x)) = x$. We have just shown by the dual of Lemma 2.2 that there does exist such a subset $S \subset M(W)$. Therefore, we have $\inf(F_W(x)) = x$.

Now, because $\sup(\{a_i\}) = x$, Lemma 5.1 allows us to replace $F_W(x)$ with $\bigcap_{i\in\mathbb{I}} F_W(a_i)$. Thus, we have $x = \inf(F_W(x)) = \inf(\bigcap_{i\in\mathbb{I}} F_W(a_i))$, the desired result. ■

Appendix B

Glossary

Differentiable Manifold: (Quoted from Ref. [9], page 2.) A *differentiable manifold* of dimension n is a set M and a family of injective mappings $x_\alpha : U_\alpha \subset \mathbb{R}^n \to M$ of open sets U_α of $\mathbb{R}^n$ into M such that:

(1) $\bigcup_\alpha x_\alpha(U_\alpha) = M$

(2) for any pair α, β, with $x_\alpha(U_\alpha) \cap x_\beta(U_\beta) = A \neq \emptyset$, the sets $x_\alpha^{-1}(A)$ and $x_\beta^{-1}(A)$ are open sets in $\mathbb{R}^n$ and the mappings $x_\beta^{-1} \circ x_\alpha$ are differentiable.

(3) The family $\{(U_\alpha, x_\alpha)\}$ is maximal relative to the conditions (1) and (2).

The pair (U_α, x_α) (or the mapping x_α) with $p \in x_\alpha(U_\alpha)$ is called a *parameterization* (or *system of coordinates*) of M at p; $x_\alpha(U_\alpha)$ is then called a *coordinate neighborhood* at p. A family $\{(U_\alpha, x_\alpha)\}$ satisfying (1) and (2) is called a *differentiable structure* on M.

Tangent Space: (Taken from Ref. [9], pages 7-8.) Let M be a differentiable manifold. A differentiable function $\alpha : (-\varepsilon, \varepsilon) \to M$ is called a (differentiable) *curve* in M. Suppose that $\alpha(0) = p \in M$, and let $\mathcal{D}$ be the set of functions on M that are differentiable at p. The *tangent vector to the curve*

α at $t = 0$ is a function $\alpha'(0) : \mathcal{D} \to \mathbb{R}$ given by

$$\alpha'(0)f = \left.\frac{d(f \circ \alpha)}{dt}\right|_{t=0}, \quad f \in \mathcal{D}.$$

A *tangent vector at p* is the tangent vector at $t = 0$ of some curve $\alpha : (-\varepsilon, \varepsilon) \to M$ with $\alpha(0) = p$. The set of all tangent vectors to M at p will be indicated by T_pM. The author of Ref. [9] (do Carmo) shows on pages 7 and 8 that the set T_pM, with the usual operations of functions, forms a vector space of dimension n, called the *tangent space*.

Riemannian Manifold: (Taken from Ref. [9], page 38.) A *Riemannian manifold* is a differentiable manifold with a given Riemannian metric (see next definition).

Riemannian Metric: (Taken from Ref. [9], page 38) A *Riemannian metric* (or *Riemannian structure*) on a differentiable manifold M is a correspondence which associates to each point p of M an inner product $\langle\ ,\ \rangle_p$ (that is, a symmetric, bilinear, positive-definite form) on the tangent space T_pM, which varies differentiably in the following sense: If $x : U \subset \mathbb{R}^n \to M$ is a system of coordinates around p, with $x(x_1, x_2, \ldots, x_n) = q \in x(U)$ and $\frac{\partial}{\partial x_i}(q) = dx_q(0, \ldots, 1, \ldots, 0)$, then $\langle \frac{\partial}{\partial x_i}(q), \frac{\partial}{\partial x_j}(q) \rangle_q = g_{ij}(x_1, \ldots, x_n)$ is a differentiable function on U.

Diffeomorphism: (Quoted from Ref. [9], page 10.) Let M_1 and M_2 be differentiable manifolds. A mapping $\varphi : M_1 \to M_2$ is a *diffeomorphism* if it is differentiable, bijective, and its inverse φ^{-1} is differentiable. [In this case, M_1 and M_2 are said to be *diffeomorphic*.] φ is said to be a *local diffeomorphism* at $p \in M$ if there exist neighborhoods U of p and V of $\varphi(p)$ such that $\varphi : U \to V$ is a diffeomorphism.

The notion of diffeomorphism is the natural idea of equivalence between differentiable manifolds. It is an immediate consequence of the chain rule that if $\varphi : M_1 \to M_2$ is a diffeomorphism, then $d\varphi_p : T_pM_1 \to T_{\varphi(p)}M_2$ is an isomorphism for all $p \in M_1$; in particular, the dimensions of M_1 and M_2 are equal.

Lattice: (Taken from Ref. [11].) A poset L with a partial ordering relation denoted by $\preceq$ is a *lattice* if $\sup(\{a, b\})$ and $\inf(\{a, b\})$ exist for all $a, b \in L$.

Abstract Polytope: (Quoted from Ref. [12].) An *abstract polytope* $\mathcal{P}$ *of (finite) rank* $n(\geq -1)$, or, more briefly, an *abstract n-polytope*, is a partially ordered set (or *poset* for short) with properties (P1),...,(P4) below. The elements of $\mathcal{P}$ are called the *faces* of $\mathcal{P}$. Sometimes we use the term *face-set* of $\mathcal{P}$ to denote the underlying set of $\mathcal{P}$ (without reference to the partial order).

Two faces F and G of $\mathcal{P}$ are said to be *incident* if $F \preceq G$ or $F \succeq G$. A *chain* of $\mathcal{P}$ is a totally ordered subset of $\mathcal{P}$. A chain has *length* i (≥ -1) if it contains exactly $i+1$ faces. Note that, by definition, the empty set is a chain (of length -1). The maximal chains are called the *flags* of $\mathcal{P}$. We denote the set of all flags of *mathscrP* by $\mathcal{F}(\mathcal{P})$. It is clear that each chain is contained in a flag of $\mathcal{P}$.

We begin with the first two defining properties for abstract polytopes, of which (P1) is purely technical.

(P1) $\mathcal{P}$ contains a least face and a greatest face; they are denoted by F_{-1} and F_n, respectively.

(P2) Each flag of $\mathcal{P}$ has length $n+1$ (that is, contains exactly $n+2$ faces including F_{-1} and F_n).

For any two faces F and G of $\mathcal{P}$ with $F \preceq G$, we call

$$G/F := \{H | H \in \mathcal{P}, F \preceq H \preceq G\}$$

a *section* of $\mathcal{P}$. In general, there is little possibility of confusion if we identify a face F and the section F/F_{-1}; the context should make clear if F is considered as an element of $\mathcal{P}$ or as a section of $\mathcal{P}$. Note that each section itself is a poset with properties (P1) and (P2), with the rank chosen appropriately; in fact, the subsequent conditions ensure that each section will indeed be an abstract polytope of this rank. Each section of $\mathcal{P}$ distinct from $\mathcal{P}$ itself is called a *proper* section of $\mathcal{P}$.

The properties (P1) and (P2) imply that $\mathcal{P}$ has a natural *rank function*, which we denote by "rank". More precisely, if F is a face of $\mathcal{P}$ and the rank of F/F_{-1} is i, then we set rank $F := i$ and call F a *face of* $\mathcal{P}$ *of rank* i, or, more briefly, an *i-face* of $\mathcal{P}$. It follows that rank $F_{-1} = -1$ and rank $F_n = n$, so that F_{-1} and F_n are the only faces of $\mathcal{P}$ of these ranks. The faces F_{-1} and F_n are called the *improper* faces of $\mathcal{P}$; all other faces of $\mathcal{P}$ are *proper*. By our convention, $\mathcal{P}$ and its n-face F_n are

identified. We also write rank $\mathcal{P} := n$ to indicate that $\mathcal{P}$ has rank n. Finally, we denote by $\mathcal{P}_i$ the set of all i-faces of $\mathcal{P}$, for $i = -1, 0, \ldots, n$.

...

We shall use the term *k-section* to mean a section of $\mathcal{P}$ of rank k.

...

Our next defining property concerns the connectedness of $\mathcal{P}$. A poset of $\mathcal{P}$ of rank n with properties (P1) and (P2) is called *connected* if either $n \leq 1$, or $n \geq 2$ and for any two proper faces F and G of $\mathcal{P}$ there exists a finite sequence of proper faces $F = H_0, H_1, \ldots, H_{k-1}, H_k = G$ of $\mathcal{P}$ such that H_{i-1} and H_i are incident for $i = 1, \ldots, k$. We say that $\mathcal{P}$ is *strongly connected* if each section of $\mathcal{P}$ (including $\mathcal{P}$ itself) is connected. Note that, in general, connectedness of each proper section of $\mathcal{P}$ does not imply connectedness of $\mathcal{P}$ itself. Our next defining property is

(P3) $\mathcal{P}$ is strongly connected.

...

Our last defining property is a certain homogeneity requirement for the sections of rank 1. Roughly speaking, this property says that the poset is basically "real". It is this property which is responsible for the close connexion with traditional polytope theory.

(P4) For each $i = 0, 1, \ldots, n-1$, if F and G are incident faces of $\mathcal{P}$, of ranks $i-1$ and $i+1$, respectively, then there are precisely *two* i-faces H of $\mathcal{P}$ such that $F \prec H \prec G$.

For more general kinds of posets or geometries, the homogeneity parameter 2 in (P4) must be replaced by other values (if it exists at all). However, for abstract polytopes it is crucial that this value is 2. Note that (P4) can be rephrased by saying that all 1-sections of $\mathcal{P}$ are of diamond shape[.]

Appendix C

Julia Code

```
__precompile__()
module AMatrix

import Logging
#= Throughout this module, W refers to a graded poset that is a supremum-filled lattice. In mose
cases, W is described by the boundary complex of a polytope (the N-dimensional generalization of a
polyhedron).
=#
function SuperficialFromNormals(nlist::Array{Float64,2},blist::Array{Float64,2},
logger=Logging.Logger("DefaultLogger",level=Logging.INFO),singtol=1e10)
    #=
    Arguments:
    --nlist: normals as unit column vectors where each column is a vector.
    --blist: the offsets as column vectors.
    --logger: a logger
    From the normals and offsets, finds the N-1 faces (facets) and vertices and constructs a
    Superficial Adjacency Matrix
    (SAM -- see below).
    Returns:
    --vertices (Array{Int64,1}): an Array of the integer IDs of all of the vertices of the convex
    polytope.
    --vcoords (Array{Float64,2}): an Array of all of the vertices of the convex polytope as column
    vectors.
    --Sam (SparseMatrixCSC{Bool,Int64}): a Boolean sparse matrix with Sam[i,j] = true if the
```

```
    minimal element/vertex with index
    i precedes the jth maximal face (N-1 face). This is the minimal information needed to
    reconstruct models whose manifold boundary complexes are equivalent to supremum filled
    algebraic lattices.
    --N (Int64): the number of N-1 faces (columns).
    --M (Int64): the number of vertices (rows).
    =#
@time begin
N,M=size(nlist) #the normals have the same dimension as the convex polytope.
# the number of columns (normals) reflects the number of N-1 cells.

vertices=Int64[] #initialize vertices (which will just contain IDs, not points
vcoords=Array(Float64,N,0)
#initialize vcoords (which will contain the coordinates of the vertices).

#=We'll need to manipulate these in the following loops. They will be used in the construction of
the Sam (see julia sparse matrix documentation. The following is the short version: I are row
indices of nonzero values, J are column indices, and Vals are the values at those indices (which in
this case are all Boolean trues). They are coordinated thus: The kth entries of I and J determine
row, column index pair of Sam (Sam[I[k],J[k]], for which Vals[k] is the value. Therefore, by
constructing these Arrays, we're effectively constructing Sam.
=#
I=Int64[]
J=Int64[]
Vals=Bool[]

    counter=0
    counters=Int64[]

    for idxs in combinations(1:M,N)
        counter+=1
        p=counter

        #we build the matrices to solve for the vertex
        normals=nlist[:,idxs]
        Ne=transpose(normals)
        offsets=blist[:,idxs]
        Be=Float64[dot(normals[:,i],offsets[:,i]) for i in 1:length(idxs)]
```

```
if cond(Ne)>singtol

    Logging.debug(logger," Condition number of at least one normal matrix exceeds $singtol.
    Normal matrix may actually be singular, and was only calculated as nonsingular because
    of numerical error. Consider lowering singular tolerance (keyword argument: singtol).
    Vertices found may exceed actual number of vertices.")
else

    v=Ne\Be #assuming that Ne is invertible, we calculate the vertex.

    #=The following is to check if we have now found a vertex that lies outside the
    hyperplane defined by a normal and offset not specified by idxs=#
    U=setdiff(1:M,idxs)
    i=1
    vinX=true
    while (i <= length(U) && vinX)

        if dot(nlist[:,U[i]],(v-blist[:,U[i]]))>0
            #dot(n,v) > b
            vinX = false
            #Logging.info(logger,"vertex outside other faces")
        end #if
        i+=1
    end #while

    #=if vinX is still true at this point, we are sure that this is a vertex of the convex
    polytope. We add it to the list of vertices and update Sam (the superficial adjacency
    matrix) accordingly.=#

    if(vinX)

        #=if a vertex is contained in more than N N-1 cells, we must check if we've already
        found it, and use its ID as the vertexid.
        =#

        r,c = size(vcoords)
        isnew = true
```

```
                i = 1
                while isnew && i <= c
                    if norm(vcoords[:,i] - v) < 1e-10
                        vertexid = i
                        isnew = false
                    end #if
                    i += 1
                end #while

                if isnew
                    #we haven't already found this vertex.
                    #add the vertex to our list of vertices as an ID.
                    push!(counters,counter)
                    Logging.debug(logger,counter)
                    Logging.debug(logger,idxs)
                    vertexid=length(vertices)+1
                    push!(vertices,vertexid)
                    vcoords=hcat(vcoords,v)
                end #if

                #=Also adjust the superficial adjacency matrix to reflect that the N-1 cells
                corresponding to the indices in idxs contain this vertex.
                =#
                append!(J,idxs)#each column denoted by idxs needs to be adjusted.
                #=in each of those columns, we need to change the row corresponding to vertexid
                (rows represent vertices).=#

                append!(I,vertexid*ones(Int,length(idxs)))

                #the value at each of those positions needs to be 1.
                        append!(Vals,ones(Bool,length(idxs)))

            end #adding vertices if

        end #big if-else
    end #for

#=Finally, we construct Sam (see above for definition/explanation).  The last argument (see
```

```
documentation) is to specify to take the max if we have duplicates (to ensure that we always have
trues and falses -- a boolean matrix).=#

Sam=sparse(I,J,Vals,length(vertices),M,max)

end #begin

return vertices, vcoords, Sam, N, M, counters

end #function

function findAllInfima(Sam::SparseMatrixCSC{Bool,Int64},N::Int64,Nv::Int64,Nf::Int64,
comb::Array{Bool,1},istart::Int64,allf=copy(Sam);
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --Sam: a Nv by Nf Boolean sparse matrix with Sam[i,j] = true if the minimal element/vertex with
    index i precedes the jth maximal face (N-1 face). (Output from SuperficialFromPoints or
    SuperficialFromNormals).
    --N: the length of the longest chain in W, where W is the boundary complex of a polytope and N
    is the dimension of the polytope.
    --Nv: The number of minimal elements (in most cases vertices) of W.
    --Nf: The number of maximal (N-1 dimensional) faces of W.
    --comb: Boolean vector Nf falses. On later iterations in the recursion, true values correspond
    to the maximal faces to be
    intersected.
    --istart: 1. This is the index at which comb is next to be modified, and it changes
    progressively from 1 to Nf during the recursion.
    --allf: a Nv by nf (nf >= Nf) Boolean sparse matrix like Sam. However,  the columns with index
    greater than Nf correspond to new elements of W found by intersecting maximal elements.
    --logger: a logger.
    Finds the intermediate elements of W by intersecting its maximal elements. This is done by
    taking all combinations of the columns of Sam and taking the Boolean and of each combination.
    Returns:
    --allf (SparseMatrixCSC{Bool,Int64}): a Nv by nf Boolean sparse matrix, where nf is the number
    of all elements in W. allf[i,j] = true if the vertex with index i precedes the jth element
    (that represented by column j).
    =#
```

```
    if sum(comb)==1
        Logging.info(logger,indmax(comb))
    end #if

    if sum(comb) > N
        return allf
    end #if
#    f=spzeros(Bool,Nv,1)
    for i=istart:Nf

        nextcomb=copy(comb)
        nextcomb[i]=true
        cols=find(nextcomb)
        arr=SparseMatrixCSC[Sam[:,j] for j in cols]
        f=map(&,arr...)
        if any(f)
            #=it's possible that this intersection of sets of vertices is empty because it's not
            the infimum of the chosen N-1 faces. But if we say an infimum must exist
            (Theorem 3),then we have to allow inclusion of the least face=#
            r,c=size(allf)
            j=1
            new=true
            while(j<=c && new) #this loop checks if we have a duplicate
                if allf[:,j]==f
                    new=false
                    Logging.debug(logger,"Found Twice")
                    Logging.debug(logger,find(f))
                end #if
                j+=1
            end #while
            if new #if we do not have a duplicate
                allf=hcat(allf,f)
            end #if
            allf=findAllInfima(Sam,N,Nv,Nf,nextcomb,i+1,allf,logger=logger)
        end #if
    end #for

    return allf
```

```
end #function

function facesToFam(allf::SparseMatrixCSC{Bool,Int64})
    #=
    Arguments:
    --allf: a Nf by nf Boolean sparse matrix, where nf is the number of all elements in W.
    allf[i,j] = true if the vertex with index i precedes the jth element (that represented by
    column j). From allf, builds a nf by nf Boolean sparse matrix, Fam, with a row and column
    corresponding to each element of W. Fam[i,j] = true for all i==j or if the element with index
    i precedes the jth element (that represented by column j).
    Returns:
    --Fam (SparseMatrixCSC{Bool,Int64}): a nf by nf Boolean sparse matrix, where nf is the total
    number of elements in W.  Fam[i,j] = true for all i==j or if the element with index i precedes
    the jth element (that represented by column j).
    =#

    Nv,Nf=size(allf)
    Fam=spzeros(Bool,Nf,Nf)
    for i in 1:Nf
        for j in 1:Nf
            if allf[:,i]==(allf[:,i]&allf[:,j]) #Boolean equivalent for issubset
                Fam[i,j]=true
            end #if
        end #for
    end #for

    return Fam
end #function


function Fvector(sFam::SparseMatrixCSC{Bool,Int64},N::Int64;
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --sFam: a sorted version of a Fam, which is a nf by nf Boolean sparse matrix, where nf is the
    total number of elements in W. sFam[i,j] = true for all i==j or if the element with index i
    precedes the jth element (that represented by column j). The fact that it is sorted means that
    elements of higher dimension (equivalently, elements with longer paths through immediately
```

```
preceding elements to minimal elements (vertices)) have lower column and row indices.
Particularly, the N-1 dimensional elements (those of highest dimension, or equivalently, those
with paths of length N) have the first block of indices, followed by those of dimension N-2
(paths of length N-1), etc. From sFam, finds the fvector, which is a vector of N integers,
where the first entry is the number of N-1 dimensional elements (equivalently, the number of
elements with paths of length N), the second entry is the number of N-2 dimensional elements,
and so on.
NOTE:
The fact that sFam is already sorted is essential for this function.  This function is
intended for use in miniSamToFam.
Returns:
--fvector (Array{Int64,1}): a vector of N integers, where the first entry is the number of N-1
dimensional elements (equivalently, the number of elements with paths of length N), the second
entry is the number of N-2 dimensional elements, and so on.
=#
nf,mf=size(sFam)
i=1
fvector=zeros(Int64,N) #We know the fvector will have N elements.
for d=1:N #This loops calculates one entry of the fvector on each iteration.
    j=1
    Logging.debug(logger,"d:")
    Logging.debug(logger,d)
    Logging.debug(logger,"i:")
    Logging.debug(logger,i)
    #This loop compares a block of sFam to the j by j identity, and increases j until they no
    longer match.
    while ((i+j-1)<=nf &&  sFam[i:(i+j-1),i:(i+j-1)]==speye(Bool,j,j))
        j+=1
        Logging.debug(logger,"j:")
        Logging.debug(logger,j)
        if !((i+j-1)<=nf)
            Logging.debug(logger,"First test failed.")
        end
        if ((i+j-1)<=nf)
            if !(sFam[i:(i+j-1),i:(i+j-1)]==speye(Bool,j,j))
                Logging.debug(logger,"Second test failed.")
            end
        end
```

```
        end #while
        j+=-1
        i+=j
        fvector[d]=j
    end #for

    return fvector
end #function

function sortFam!(Fam::SparseMatrixCSC{Bool,Int64},N::Int64,
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --Fam: a nf by nf Boolean sparse matrix, where nf is the total number of elements in W.
    Fam[i,j] = true for all i==j or if the element with index i precedes the jth element (that
    represented by column j).
    --N: the length of the longest chain in W, where W is the boundary complex of a polytope and N
    is the dimension of the polytope.
    --logger: a logger.
    Sorts Fam (by changing Fam itself) so that elements of higher dimension (equivalently,
    elements with longer paths through immediately preceding elements to minimal elements
    (vertices)) have lower column and row indices. Particularly, the N-1 dimensional elements
    (those of highest dimension, or equivalently, those with paths of length N) have the first
    block of indices, followed by those of dimension N-2 (paths of length N-1), etc. Also finds
    the fvector (see below) by taking advantage of the information found during the sorting
    process.
    Returns:
    --fvector (Array{Int64,1}): a vector of N integers, where the first entry is the number of N-1
    dimensional elements (equivalently, the number of elements with paths of length N), the second
    entry is the number of N-2 dimensional elements, and so on.
    =#
    Nf,nf=size(Fam)
    fvector=zeros(Int64,N)
    sorted=0
    tosort=sorted+1
    for d in 1:(N-1)
        moved=0
        tomove=moved+1
```

```
        for i in tosort:Nf
            j=find(Fam[i,tosort:end])+sorted
            if length(j)==1
                if j[1]==i#=this means it precedes
                    nothing in this block.=#
                    tc=Fam[:,tomove+sorted]
                    Fam[:,tomove+sorted]=Fam[:,i]
                    Fam[:,i]=tc
                    tr=Fam[tomove+sorted,:]
                    Fam[tomove+sorted,:]=Fam[i,:]
                    Fam[i,:]=tr
                    moved+=1
                    tomove+=1
                end #inside if
            end #outside if
        end #for
        sorted+=moved
        tosort=sorted+1
        fvector[d]=moved
    end #for
    fvector[N]=Nf-sorted
    #=
    for d in 1:N
        fn=1
        F=sum(fvector)
        while ((F+fn)<=Nf && Fam[(F+1):(F+fn),(F+1):(F+fn)]==speye(Bool,fn,fn))
            fn+=1
        end #while
        fvector[d]=(fn-1)
    end #for
    =#

    return fvector
end #function

function sortFam(Fam::SparseMatrixCSC{Bool,Int64},N::Int64;
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
```

```
Arguments:
--Fam: a nf by nf Boolean sparse matrix, where nf is the total number of elements in W.
Fam[i,j] = true for all i==j or if the element with index i precedes the jth element (that
represented by column j).
--N: the length of the longest chain in W, where W is the boundary complex of a polytope and
N is the dimension of the polytope.
--logger: a logger.
From Fam, creates a sorted version (see sFam below) Also finds the fvector (see below) by
taking advantage of the information found during the sorting process.
Returns:
--sFam (SparseMatrixCSC{Bool,Int64}): a sorted version of a Fam, which is a nf by nf Boolean
sparse matrix, where nf is the total number of elements in W. sFam[i,j] = true for all i==j or
if the element with index i precedes the jth element (that represented by column j). The fact
that it is sorted means that elements of higher dimension (equivalently, elements with
longer paths through immediately preceding elements to minimal elements (vertices)) have lower
column and row indices. Particularly, the N-1 dimensional elements (those of highest
dimension, or equivalently, those with paths of length N) have the first block of indices,
followed by those of dimension N-2 (paths of length N-1), etc.
--fvector (Array{Int64,1}): a vector of N integers, where the first entry is the number of N-1
dimensional elements (equivalently, the number of elements with paths of length N), the second
entry is the number of N-2 dimensional elements, and so on.
=#
Nf,nf=size(Fam)
sFam=copy(Fam)
fvector=zeros(Int64,N)
sorted=0
tosort=sorted+1
for d in 1:(N-1)
    moved=0
    tomove=moved+1
    for i in tosort:Nf
        j=find(sFam[i,tosort:end])+sorted
        if length(j)==1
            if j[1]==i#=this means it precedes
                nothing in this block.=#
                tc=sFam[:,tomove+sorted]
                sFam[:,tomove+sorted]=sFam[:,i]
                sFam[:,i]=tc
```

```
                    tr=sFam[tomove+sorted,:]
                    sFam[tomove+sorted,:]=sFam[i,:]
                    sFam[i,:]=tr
                    moved+=1
                    tomove+=1
                end #inside if
            end #outside if
        end #for
        sorted+=moved
        tosort=sorted+1
        #Logging.debug(logger,"fvector:")
        #Logging.debug(logger,fvector)
        #Logging.debug(logger,"N:")
        # Logging.debug(logger,N)
        # Logging.debug(logger,"d:")
        # Logging.debug(logger,d)
        fvector[d]=moved
    end #for
    fvector[N]=Nf-sorted
    #=
    for d in 1:N
        fn=1
        F=sum(fvector)
        while ((F+fn)<=Nf && sFam[(F+1):(F+fn),(F+1):(F+fn)]==speye(Bool,fn,fn))
            fn+=1
        end #while
        fvector[d]=(fn-1)
    end #for
    =#
    return sFam,fvector
end #function

function SamToFam(Sam::SparseMatrixCSC{Bool,Int64},N::Int64;
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --Sam: a Nv by Nf Boolean sparse matrix with Sam[i,j] = true if the minimal element (vertex)
    with index i precedes the jth maximal face (N-1 face). (Output from SuperficialFromPoints or
```

```
    SuperficialFromNormals). Nv is the number of minimal elements (in most cases vertices) of W.
    Nf is the number of maximal elements (N-1 faces) of W.
    --N: the length of the longest chain in W, where W is the boundary complex of a polytope and N
    is the dimension of the polytope.
    --logger: a logger.
    From N and the information in Sam (the minimal information necessary), constructs Fam, sFam,
    and fvector (see below) using findAllInfima, facesToFam, and sortFam.
    Returns:
    --Fam (SparseMatrixCSC{Bool,Int64}): a nf by nf Boolean sparse matrix, where nf is the total
    number of elements in W.  Fam[i,j] = true for all i==j or if the element with index i precedes
    the jth element (that represented by column j).
    --sFam (SparseMatrixCSC{Bool,Int64}): a sorted version of a Fam, which is a nf by nf Boolean
    sparse matrix, where nf is the total number of elements in W. sFam[i,j] = true for all i==j or
    if the element with index i precedss the jth element (that represented by column j). The fact
    that it is sorted means that elements of higher dimension (equivalently, elements with longer
    paths through immediately preceding elements to minimal elements (vertices)) have lower column
    and row indices. Particularly, the N-1 dimensional elements (those of highest dimension, or
    equivalently, those with paths of length N) have the first block of indices, followed by those
    of dimension N-2 (paths of length N-1), etc.
    --fvector (Array{Int64,1}): a vector of N integers, where the first entry is the number of N-1
    dimensional elements (equivalently, the number of elements with paths of length N), the second
    entry is the number of N-2 dimensional elements, and so on.
    =#
    Nv,Nf=size(Sam)
    comb=zeros(Bool,Nf) #the starting combination for findAllInfima
    allf=findAllInfima(Sam,N,Nv,Nf,comb,1,logger=logger)
    Fam=facesToFam(allf)
    sFam,fvector=sortFam(Fam,N,logger=logger)

    return Fam,sFam,fvector
end #function

function findPrec(Sam::SparseMatrixCSC{Bool,Int64},Nv::Int64,Nf::Int64,vs::Array{Int64,1};
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --Sam: a Nv by Nf Boolean sparse matrix with Sam[i,j] = true if the minimal
    element/vertex with index i precedes the jth maximal face (N-1 face).
```

```
(Output from SuperficialFromPoints or SuperficialFromNormals).
--Nv: The number of minimal elements (in most cases vertices) of W.
--Nf: The number of maximal (N-1 dimensional) faces of W.
--vs: The indices of the minimal elements (vertices) for which we desire to find the supremum
(in the poset sense, the least element of W that succeeds all of the vertices in question).
--logger: a logger.
This function is intended for use in miniSamToFam, as a preliminary step to findMiniInfima. It
essentially finds all the maximal elements (N-1 faces) that succeed all of the minimal
elements (vertices) with indexes in vs. From this information, it constructs the various
arguments needed by findMiniInfima (see findMiniInfima documentation)
Returns:
--comb (Array{Bool,2}): a Boolean column vector of length Nf. comb[i]=true if the ith maximal
element (N-1 face) is preceded by all of the minimal elements (vertices) with an index in vs.
--BaseFs (Array{Int64,1}): an integer vector containing the indices of all the maximal
elements (N-1 faces) which are preceded by all of the minimal elements (vertices) with an
index in vs.
--NbFs (Int64): the length of BaseFs.
--vset (Array{Bool,2}): a Boolean column vector of length Nv. vset[i]=true if i is in vs.
--minitreepossible (Bool): This value is false if there is no maximal element (N-1 face) that
is preceded by all of the minimal elements (vertices) with an index in vs. This means that the
supremum of these minimal elements is the "greatest face", or equivalently, that no
sub-lattice of W contains all the minimal elements (vertices). Otherwise, this value is true.
=#
comb=zeros(Bool,Nf,1)
vset=zeros(Bool,Nv,1)
vset[vs]=true
for i=1:Nf
    if (Sam[:,i]&vset)==vset
        comb[i]=true
    end #if
end #for
minitreepossible=true
if sum(comb)==0
    minitreepossible=false
    BaseFs=Int64[]
else
    BaseFs=find(comb)
end #if-else
```

```
    NbFs=length(BaseFs)
    return comb,BaseFs,NbFs,vset,minitreepossible
end #function

function findMiniInfima(Sam::SparseMatrixCSC{Bool,Int64},Nv::Int64,Nf::Int64,
BaseFs::Array{Int64,1},NbFs::Int64,vset::Array{Bool,2},comb::Array{Bool,2},
istart::Int64,allf=copy(Sam);logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --Sam: a Nv by Nf Boolean sparse matrix with Sam[i,j] = true if the minimal element (vertex)
    with index i precedes the jth maximal face (N-1 face). (Output from SuperficialFromPoints or
    SuperficialFromNormals).
    --Nv: The number of minimal elements (in most cases vertices) of W.
    --Nf: The number of maximal (N-1 dimensional) faces of W.
    --BaseFs: an integer vector containing the indices of all the maximal elements (N-1 faces)
    which are preceded by all of the minimal elements (vertices) whose indices correspond to true
    values in vset.
    --NbFs: the length of BaseFs.
    --vset: a Boolean column vector of length Nv. vset[i]=true if i is an index of the of the
    minimal elements (vertices) for which we desire to find the supremum (in the poset sense, the
    least element of W that succeeds all of the vertices in question).
    --comb: a Boolean column vector of length Nf. comb[i]=true if the ith maximal element (N-1
    face) is preceded by all of the minimal elements (vertices) whose indices correspond to true
    values in vset.
    --istart: 1. This is the index at which comb is next to be modified, and it changes
    progressively from 1 to Nf during the recursion.
    --allf: a Nv by nf (nf >= Nf) Boolean sparse matrix like Sam. However,  the columns with index
    greater than Nf correspond to new elements of W found by intersecting maximal elements.
    --logger: a logger.
    Finds all elements of W that precede the supremum of the minimal elements (vertices) in
    question. Like findAllInfima, it does so by intersecting all of the maximal elements with
    indices in BaseFs. Where W is the boundary complex of a polytope, the supremum is the
    lowest-dimensional face that contains all of the vertices in question.
    Returns:
    --allf (SparseMatrixCSC{Bool,Int64}): a Nv by mf Boolean sparse matrix, where mf is the number
    of elements in W that precede the supremum of the minimal elements (vertices) in question,
    plus one for the supremum itself, plus all Nf of the maximal elements (N-1 faces). allf[i,j] =
    true if the vertex with index i precedes the jth element (that represented by column j).
```

```
    =#
    if sum(comb)==NbFs
        Logging.info(logger,indmax(comb))
    end #if
    f=spzeros(Bool,Nv,1)
    if sum(f&vset)==1
        return allf
    end #if

    for i=setdiff(istart:Nf,BaseFs)
        nextcomb=copy(comb)
        nextcomb[i]=true
        cols=find(nextcomb)
        arr=SparseMatrixCSC[Sam[:,j] for j in cols]
        f=map(&,arr...)
        if any(f)
            #=it's possible that this intersection of sets of vertices is empty because it's not
            the infimum of the chosen N-1 faces. But if we say an infimum must exist
            (Theorem 3), then we have to allow inclusion of the least face=#
            r,c=size(allf)
            j=1
            new=true
            while(j<=c && new) #this loop checks if we have a duplicate
                if allf[:,j]==f
                    new=false
                end #if
                j+=1
            end #while
            if new #if we do not have a duplicate
                allf=hcat(allf,f)
            end #if
            allf=findMiniInfima(Sam,Nv,Nf,BaseFs,NbFs,vset,nextcomb,i+1,allf,logger=logger)
        end #if
    end #for

    return allf
end #function
function presort!(allf::SparseMatrixCSC{Bool,Int64},Nf::Int64;
```

```
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --allf: a Nv by mf Boolean sparse matrix, where mf is the number of elements in W that precede
    the supremum of the minimal elements (vertices) in question, plus one for the supremum itself,
    plus all Nf of the maximal elements (N-1 faces). allf[i,j] = true if the vertex with index i
    precedes the jth element (that represented by column j).
    --Nf: The number of maximal (N-1 dimensional) faces of W.
    --logger: a logger.
    This function is intended for use in findFlag, which in turn is intended for use in
    miniSamToFam. Despite this, it is not necessary that allf be an output of findMiniInfima. It
    also works on the output of findAllInfima. The function sorts the columns of allf such that
    those elements which succeed the most minimal elements (vertices) come first, followed by
    those with the second-most minimal elements (vertices), etc. (Essentially, the columns with
    the most true values are put first, followed by those with the second-highest number of true
    values, etc.) This preliminary step is essential for findFlag to function correctly.
    Returns:
    --allf (SparseMatrixCSC{Bool,Int64}): this is the same allf that came in as an argument, but
    sorted as described above.
    --worked (Bool): A bool that reveals if the sort worked properly. It is true if a second
    attempt at sorting results in no columns needing to be moved.
    =#
    nv,mf=size(allf)
    colsums=sum(allf[:,Nf+1:end],1)
    sortPerm=sortperm(collect(colsums),rev=true)
    allf[:,Nf+1:end]=allf[:,Nf+sortPerm]
    checkPerm=sortperm(collect(sum(allf[:,Nf+1:end],1)),rev=true)
    worked=false
    if checkPerm==collect(1:(mf-Nf))
        worked=true
    end #if
    return allf,worked

end #function

function findFlag(allf::SparseMatrixCSC{Bool,Int64},Nf::Int64;minitree=false,BaseFs=collect(1),
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
```

```
Arguments:
--allf: a Nv by mf Boolean sparse matrix, where mf is the number of elements in W that precede
the supremum of the minimal elements (vertices) in question, plus one for the supremum itself,
plus all Nf of the maximal elements (N-1 faces). allf[i,j] = true if the vertex with index i
precedes the jth element (that represented by column j).
--Nf: The number of maximal (N-1 dimensional) faces of W.
--minitree: a Boolean value; a keyword argument that allows the user to specify if allf
includes all the elements of W (corresponding to minitree=false) or if it is as described
above (the output of findMiniInfima, and corresponding to minitree=true).
--BaseFs: if minitree=true, this is an integer vector containing the indices of all the
maximal elements (N-1 faces) which are preceded by all of the minimal elements (vertices) with
an index in vs, the input of miniSamToFam. If minitree=false, BaseFs is just an integer vector
with one element, 1. In this case, it does not matter which maximal element (N-1 face) we use,
so we use the first one.
--logger: a logger.
This function is intended for use in miniSamToFam. Despite this, it is not necessary that allf
be an output of findMiniInfima. It also works on the output of findAllInfima. The function
finds a flag, or a path among the elements of allf, from a maximal element, through
immediately preceding elements, to a minimal element. More importantly for out purposes, it
finds the length of that path, which corresponds to the rank of the graded poset, if we impose
a rank function on our supremum-filled lattice to make it a graded poset. Where W corresponds
to the boundary complex of the polytope, this is the dimension of the lowest-dimensional face
that contains all the vertices in vs (the supremum). In the case that allf contains all the
elements of W (minitree=false), this is the same dimension returned by SuperficialFromPoints,
that of the entire polytope.
Returns:
--Flag: a Nv by N Boolean sparse matrix that contains the elements that form a flag; a path
from a maximal element to a minimal element through immediately preceding elements. Each
column corresponds to one of these elements. Flag[i,j] = true if the minimal element (vertex)
with index i precedes the jth element in the flag (that represented by column j).
--N: the length of the longest chain in the sub-lattice of W formed by the elements of allf.
Where W is the boundary complex of a polytope, N is the dimension of the
lowest-dimensional face that contains all the vertices in vs (the supremum).
=#
allf,worked=presort!(allf,Nf,logger=logger)
if !worked
    Logging.warn(logger,"allf not properly sorted. Flag and N may be incorrect.")
end #if
```

```
F=allf[:,BaseFs[1]]
Flag=allf[:,BaseFs[1]]
Mf,mf=size(allf)
j=Nf+1
Logging.debug(logger,"size of allf (row,column):")
Logging.debug(logger,Mf)
Logging.debug(logger,mf)
Logging.debug(logger,"starting j (Nf+1):")
Logging.debug(logger,j)
foundMinElem=false
while(j<=mf && !foundMinElem)
    if allf[:,j]==(allf[:,j]&F) #Boolean equivalent for issubset
        nv,nflag=size(Flag)
        Logging.debug(logger,"j:")
        Logging.debug(logger,j)
        Logging.debug(logger,"nv,nflag:")
        Logging.debug(logger,nv)
        Logging.debug(logger,nflag)
        precedesall=true
        k=2
        while(k<=nflag && precedesall)
            if sum(Flag[:,end])>sum(Flag[:,end-1])
                tc=Flag[:,end-1]
                Flag[:,end-1]=Flag[:,end]
                Flag[:,end]=tc
            end #if
            if !(allf[:,j]==(allf[:,j]&Flag[:,k])) #Boolean equivalent for !issubset
                precedesall=false
            end #if
            k+=1
        end #while
        if precedesall
            Flag=hcat(Flag,allf[:,j])
            Logging.debug(logger,"Adding new element to Flag")
            Logging.debug(logger,"j:")
            Logging.debug(logger,j)
            Logging.debug(logger,"nv,nflag:")
            Logging.debug(logger,nv)
```

```
                Logging.debug(logger,nflag)
                if sum(Flag[:,end])==1
                    foundMinElem=true
                end #if
            end #if
        end #if
        j+=1
    end #while
    nv,nflag=size(Flag)
    N=nflag
    if minitree
        N+=-1
    end #if
    if N==0
        Logging.debug(logger,"N:")
        Logging.debug(logger,N)
    end #if

    return Flag,N
end #function

function miniSamToFam(Sam::SparseMatrixCSC{Bool,Int64},originalN::Int64,vs::Array{Int64,1};
logger=Logging.Logger("DefaultLogger",level=Logging.INFO))
    #=
    Arguments:
    --Sam: a Nv by Nf Boolean sparse matrix with Sam[i,j] = true if the minimal element (vertex)
    with index i precedes the jth maximal face (N-1 face). (Output from SuperficialFromPoints or
    SuperficialFromNormals). Nv is the number of minimal elements (in most cases vertices) of W.
    Nf is number of maximal elements (N-1 faces) of W.
    --originalN: the length of the longest chain in W. Where W is the boundary complex of a
    polytope, N is the dimension of the polytope.
    --vs: The indices of the minimal elements (vertices) for which we desire to find the supremum
    (in the poset sense, the least element of W that succeeds all of the vertices in question).
    --logger: a logger.
    From originalN and the information in Sam (the minimal information necessary), and using
    findPrec, findMiniInfima, findFlag, facesToFam, and Fvector, the function finds all elements
    of W that precede the supremum of the minimal elements (vertices) in question. (In the poset
    sense, the supremum is the least element of W that succeeds all of the vertices in question;
```

```
    those with indices in vs). Where W is the boundary complex of a polytope, the supremum is the
    lowest-dimensional face that contains all of the vertices in question. Also finds the fvector
    (see below).
    Returns:
    --miniSFam (SparseMatrixCSC{Bool,Int64}): a mf by mf Boolean sparse matrix, where mf is the
    number of elements of W that precede the supremum of the minimal elements (vertices), plus the
    supremum itself. Fam[i,j] = true for all i==j or if the element with index i precedes the jth
    element (that represented by column j).
    --fvector (Array{Int64,1}): a vector of N integers (N being the output of findFlag) where the
    first entry is the number of N-1 dimensional elements (equivalently, the number of elements
    with paths of length N), the second entry is the number of N-2 dimensional elements, and so
    on.
    =#
    Nv,Nf=size(Sam)
    comb,BaseFs,NbFs,vset,minitreepossible=findPrec(Sam,Nv,Nf,vs,logger=logger)
    Logging.debug(logger,"minitreepossible:")
    Logging.debug(logger,minitreepossible)
    Logging.debug(logger,"BaseFs:")
    Logging.debug(logger,BaseFs)
    if minitreepossible
        allf=findMiniInfima(Sam,Nv,Nf,BaseFs,NbFs,vset,comb,1,logger=logger)
        Flag,N=findFlag(allf,Nf,minitree=true,BaseFs=BaseFs,logger=logger)
        miniSFam=facesToFam(allf[:,Nf+1:end])
        fvector=Fvector(miniSFam,N,logger=logger)
    else
        Logging.info(logger,"Mini Adjacency Matrix not possible\n(vertices do not connect until
        highest dimension\n i.e. the supremum of
        these vertices is the greatest face)")
        miniSFam=spzeros(Bool,Nf,Nf)
        fvector=zeros(Int64,originalN)
    end #if

    return miniSFam,fvector
end #function
export SuperficialFromNormals, findAllInfima, facesToFam, sortFam!, sortFam, SamToFam, findPrec,
findMiniInfima, findFlag, miniSamToFam, Fvector, presort!

end #module
```

Bibliography

[1] A. Seko, K. Shitara, and I. Tanaka, "Efficient determination of alloy ground-state structures," Physical Review B 90 (2014).

[2] G. L. W. Hart, "Where are nature's missing structures?," Nature Materials **6,** 941–945 (2007).

[3] M. K. Transtrum and P. Qiu, "Model reduction by manifold boundaries," Physical review letters **113,** 098701 (2014).

[4] M. K. Transtrum, G. Hart, and P. Qiu, "Information topology identifies emergent model classes," (2014).

[5] E. J. Candes and M. B. Wakin, "An Introduction to Compressive Sampling," IEEE Signal Processing Magazine **V.21,** 21–30 (2008).

[6] D. L. Donoho, "Compressed sensing," IEEE Transactions on Information Theory **52,** 1289–1306 (2006).

[7] M. A. Davenport, "The Fundamentals of Compressive Sensing,".

[8] E. Candes, J. Romberg, and T. Tao, "Stable Signal Recovery from Incomplete and Inaccurate Measurements," (2005).

[9] M. P. do Carmo, *Riemannian geometry* (Birkhuser, Boston, 1992).

[10] M. K. Transtrum, “Manifold boundaries give "gray-box" approximations of complex models,” (2016).

[11] G. A. Gratzer, *General lattice theory* (Elsevier, New York, 1978).

[12] P. McMullen and E. Schulte, *Abstract Regular Polytopes* (Cambridge University Press, Cambridge, 2002), iD: Accession Number: 120688.

Index