Graph Dynamics:
Learning and Representation

by

Andre Figueiredo Ribeiro

B.S. Computer Science
Univ. Fed. de Minas Gerais, Brazil, 2000

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

Massachusetts Institute of Technology

February 2006

Signature of Author: _____
Program in Media Arts and Sciences
September 28, 2005

Certified by: _____
Deb K. Roy
Associate Professor, Program in Media Arts and Sciences
Thesis Supervisor

Accepted by: _____
Andrew B. Lippman
Chairman, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Graph Dynamics:
Learning and Representation

by

Andre F. Ribeiro

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
on September 1st, 2005 in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Graphs are often used in artificial intelligence as means for symbolic knowledge representation. A graph is nothing more than a collection of symbols connected to each other in some fashion. For example, in computer vision a graph with five nodes and some edges can represent a table – where nodes correspond to particular shape descriptors for legs and a top, and edges to particular spatial relations. As a framework for representation, graphs invite us to simplify and view the world as objects of pure structure whose properties are fixed in time, while the phenomena they are supposed to model are actually often changing. A node alone cannot represent a table leg, for example, because a table leg is not *one* structure (it can have many different shapes, colors, or it can be seen in many different settings, lighting conditions, etc.) Theories of knowledge representation have in general concentrated on the stability of symbols – on the fact that people often use properties that remain unchanged across different contexts to represent an object (in vision, these properties are called invariants). However, on closer inspection, objects are variable as well as stable. How are we to understand such problems? How is that assembling a large collection of changing components into a system results in something that is an altogether stable collection of parts? The work here presents one approach that we came to encompass by the phrase "graph dynamics". Roughly speaking, dynamical systems are systems with states that evolve over time according to some lawful "motion". In graph dynamics, states are graphical structures, corresponding to different hypothesis for representation, and motion is the correction or repair of an antecedent structure. The adapted structure is an end product on a path of test and repair. In this way, a graph is not an exact record of the environment but a malleable construct that is gradually tightened to fit the form it is to reproduce.

In particular, we explore the concept of attractors for the graph dynamical system. In dynamical systems theory, attractor states are states into which the system settles with the passage of time, and in graph dynamics they correspond to graphical states with many repairs (states that can cope with many different contingencies). In parallel with introducing the basic mathematical framework for graph dynamics, we define a game for its control, its attractor states and a method to find the attractors. From these insights, we work out two new algorithms, one for Bayesian network discovery and one for active learning, which in combination we use to undertake the object recognition problem in computer vision. To conclude, we report competitive results in standard and custom-made object recognition datasets.

Thesis Supervisor: Deb Roy
Title: Associate Professor of Media Arts and Sciences

Thesis Committee

Thesis Supervisor: _____

Deb Roy
Associate Professor of Media Arts and Sciences
MIT Media Laboratory

Thesis Reader: _____

Whitman Richards
Professor of Media Arts and Sciences
MIT Media Laboratory

Thesis Reader: _____ 8/30/05

Robert Goldstone
Professor of Psychology
Indiana University

Table of Contents

Table of Figures

1. Introduction

1.1 Adaptation and Networks

A self-regulatory network is a network that can recover automatically after the occurrence of momentary errors. The underlying notion is that of a system that can be started in any given state and still converge to a desired state. Embedded in a complex environment, it is hard for a system to specify in advance all the conditions it should operate under. The self-regulatory property handles such a complicated situation by learning to cope with arbitrary risky initial states, from which the system becomes apt to recover and return to a desired state. For example, it is important that the control system of an airplane to self-regulate. The plane may experience a fault and will have to recover autonomously. If the plane experiences a fault because of a temporary electrical power problem, it may fault for a short time, but as long as it self-regulates, it will return to an adequate overall state.

Studies of networks are useful at several different disciplines. For example, they have been used to describe the structure of biochemichal networks in biology, neural networks in neurosciences, food webs in ecology, networks of social interaction in sociology, computer networks in engineering. A graph is a useful representation of a network, where the graph's nodes represent the components of the network (species, neurons, agents, etc.), and its links their mutual interactions.

Usually one studies the relationship between the structure of the system and its behavior (e.g., how the structure of the internet influences the dissemination of information). At this scope the network and its components are typically taken to be static; the prime concern is the dynamics of other variables on a network in respect to the fixed structure.

However, one should also be interested in how networks themselves change with time. Biochemical, neural, ecological, social networks are not static. They evolve and change constantly, strategically responding to their unsteady surroundings. Understanding the processes and mechanisms involved in the development of complex networks is a big challenge. Unlike the simple network models that have now become fashionable in physics [1], to study such networks we must take learning or adaptation processes into account.

In order to address such questions in a mathematical model, one is naturally led to dynamical systems in which the graph that captures the network is also a variable. Here we present a model with such structure. The analysis is facilitated by the development of some new tools in graph, learning and game theory. Together, the model and these tools address possible mechanisms by which networks change, adapt and organize.

1.2 Graphs, Dynamical Systems and Games

1.2.1 Graphs

Self-regulatory networks have a common underlying structure. First, nodes are different states of the phenomena they model, interact with or represent. Second, some nodes - called attractors - are preferred. To make sense of this structure, it is convenient to think of an agent embedded in a particular node that actively tries to stay in those preferred states. We think of attractors as ideal states for this agent and other nodes as states representing errors, faults or contingencies; meanwhile we can think of edges as actions, tests or repairs. The agent can make observations regarding the behavior of its current location and infer likely faults; it can repair the system with actions, observe its new behavior and infer again the likelihood of faults, until the system has reached a desired state. Typically, attractor states have many cycles returning to them, meaning that they can cope with many different errors.

Figure 1 depicts a visual notation for self-regulatory networks. Graphs are sometimes used to encode information about the values of state variables of a system at a time t given information about the variables at time $t-1$. An edge between two nodes indicates that the variables are somehow correlated. More precisely, the agent associates various gains and losses with being in particular states, and we double-circle the states he desires to stay in. This agent is capable of observing local aspects of the system state and taking action (i.e., choosing an edge) based on its observations. Since the outgoing edges are the decision space for the agent, we can call them decision edges and, visually, we dot them. Meanwhile, the other edges depict causal relationships.

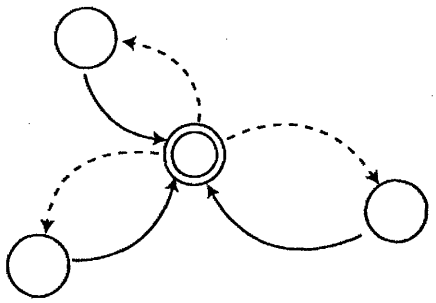


Figure 1: Self-regulatory network visual notation.

Consider the example of the simplest self-regulatory system: the thermostat. Thermostats are used in many environments to keep their temperature in a desired level. The apparatus that is normally used to maintain such a system in equilibrium consists of a heating element connected to a thermostat. The thermostat consists of two parallel metallic plates: one is fixed and the other one can move. If the temperature falls below some threshold, the two plates make contact. This closes an electric contact, which activates the heating element, so that the temperature is raised again. If the temperature becomes high enough, the bi-metallic plate bends back again and the contact is broken. This stops the heating activity so that the temperature can decline again. This is a classic example of a feedback system [2]. And it can be represented with an self-regulatory network. We have indeed a system with two states: plates in contact, and plates not in contact. The first state of the thermostat corresponds to the world-state: "The temperature is too low", the second one to the world-state: "The temperature is high enough". In that sense the thermostat can be said to have an elementary knowledge of the world around itself, or, just the same, to represent it in some way. But it also knows about the system itself, namely what is the ideal temperature for the system to maintain in equilibrium. The position of the plate then is an embodiment of the knowledge about the present interaction between system and environment, which is arrived at through an elementary form of perception. So, the states of the thermostat represent the changes in the external environment (temperature fluctuations), as well as the ideal state of the system (its equilibrium temperature).

Sometimes it is useful to distinguish two aspects of the thermostat operation. The first is perception: a change in the outside temperature causes a change in the bending of the plate. After which comes the second, an action phase: the thermostat heats the air. After a certain interval, the environment temperature increases above the threshold, and a new perception-action sequence is triggered. So, the process works in a circular fashion. This is called a feedback loop: the external consequences (temperature change) of the

system's action is fed back into the system and re-evaluated so that a new, corrective action can be initiated.

However, there is also a feedforward aspect in the process, a predictive aspect that in our notation correspond to decision edges. It means that the action is initiated not only by outside reactions which determine the direction in which the previous action should be changed, but also by the anticipation that a particular action will bring the system closer to its preferred state. In general, a self-regulatory system will use feedback as well as feedforward mechanisms in order to deal with changes in the outside environment. In the thermostat example the feedforward mechanism is very primitive: the only anticipation made is that if the temperature is below the threshold, then the activation of the heating element will bring the system closer to its desired behavior. This is because the representation used by the system has only two states (above and below some threshold) and its only internal knowledge is that a certain action will transform one state into the other one. Unlike in this example, there is typically indeterminacy among the different states the system might be in, and action and feedback can also inform in what state the system is.

1.2.2 Dynamical Systems

Graphs are, by definition, static structures. To study the dynamics of graphs - that is, how graphs change and how we can describe those changes - we must first position ourselves differently in relation to what graphs are. We must look at graphs not as static depictions of information but byproducts of a process, a process obtained by iteratively applying some given operators on simpler graphs. The graph in *Figure 2(a)* can, for example, be thought both as a set of vertices $V=\{v_1,v_2,v_3\}$ and edges $E=\{e_1,e_2\}$ and a dynamical system consisting of an initial graph $X=\{x_1\}$ (in this case the singleton graph) and an operator $T=\{t_1\}$ (translation to the right). A graph dynamical system (X, Γ) is a graph X together with mappings Γ: $X \rightarrow X$, recursively defined by $\Gamma^1 = \Gamma$ and $\Gamma^n = \Gamma(\Gamma^{n-1}(X))$ for $n \geq 2$. A given graph X is fixed, or invariant, under some given operator Γ if X and $\Gamma(X)$ are isomorphic.

A graph dynamical system has also its own graph representation, illustrated by the graph in *Figure 2(b)* and reminiscent of our notation for self-regulatory systems. A walk (or circuit) on this graph is some path of edges that brings you back to the node you started from, and corresponds to some sequence of mappings that leave the original graph invariant, i.e. an identity. So, for example, we can generate (construct) the graph in *(a)* by going two times around the only cycle in the graph *(b)*, see *(c)*.

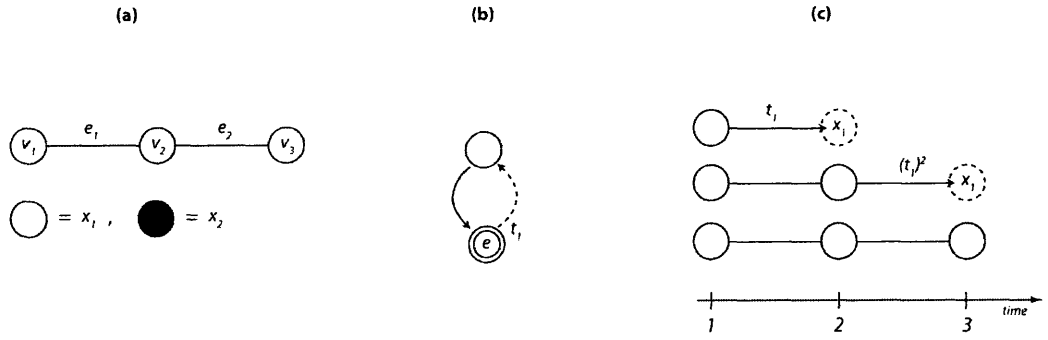


Figure 2: Simple graph dynamics example.

To carry this simple example further, consider the three graphs in *Figure 3(a)*. They are examples of "line" graphs, with occasional noise or distortion. The dynamical representation appears on *(b),* while the transformations referred in the figure are illustrated in (c). We see that there are two levels. One is a "point" graph, consisting of three nodes. This graph is an identity element for the next level, which is a "line" graph. The decision and causal edges are test-repair pairs that can be used to adapt the network representation to the target environment. In this example, the line graph can test for another point graph directly to the right (by choosing transformation t_1), while the point graph can test two different positions (by choosing transformation t_2 or t_3). The two individual graphs capture the kind of distortion that a graph can undergo at its spatial level. For example, for the graph on the top in *Figure 3(a)* the re-construction process goes on as follows: start with a node, test and verify identity point (at point level), test and verify transformation t_1 (at line level), iterate. However, for the bottom graph the process must go differently, at $t=4$ the test for the identity (at point level) is not verified and transformation t_2 is used as a repair action. We say then that the graph has adapted to its environment. That the latter construction had to go further into the graph structure tell us that it has more distortion.

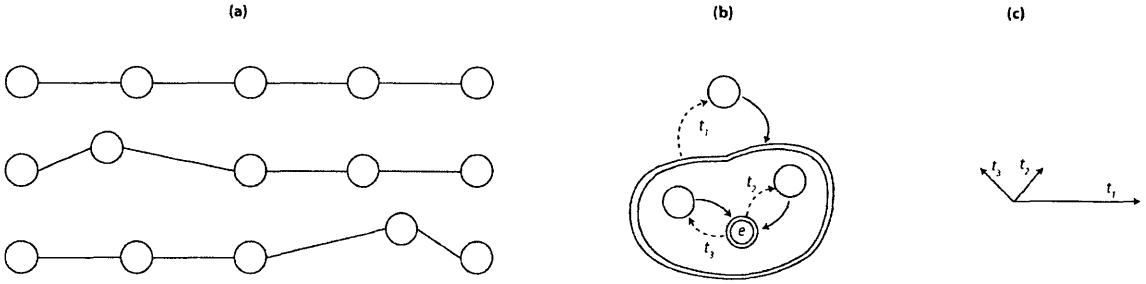


Figure 3: Hierarchy.

More precisely, we have defined a graph as a set of mappings. The concept of a mathematical group is intimately related to the concept of mapping, and groups can be thought of as a set of mappings[1]. Furthermore, groups naturally describe the suggested hierarchical structure. However, in this thesis we will not explore this somewhat obvious relationship. Here, we will focus on the computational side of the theory. The interested readers can look at [3].

1.2.3 Games on Graphs

Unlike the simple examples on the last sections, we will typically have a large collection of graphs (each corresponding to a different agent) and it is unnecessary or it is impractical to construct all of them separately. Also, attractor states will typically not consist of single nodes but of more complex graphical structures. In a typical feedback fashion, each agent should instead choose discriminative graphical constructions, letting error inform which is the actual graph.

This corresponds to the situation where separate regulatory systems are interdependent. In this arrangement, different agents have different gains and losses, and are responsive to the actions of the other agents. For an example, consider the case of two thermostats that are active simultaneously in many homes: a refrigerator and a furnace. The refrigerator keeps the temperature inside it at a desired level, through a feedback loop. The furnace does the same thing, but with respect to the temperature inside the house. Each of the agents has an impact on the other. The output function of the refrigerator cools the air inside the refrigerator by transferring heat from inside to outside. When the refrigerator's compressor is running, it's not just cooling the refrigerator but heating the room. In the same way, the furnace is

influencing the refrigerator. Thus the furnace's action makes the refrigerator work harder to keep its temperature down.

The processes illustrated in the last sections can be thought of as a game against nature: if nature chooses a node not in the attractor-set (or "occupies" that node), the player is prepared to counteract and bring the system back to its desired state. There is, however, the more complex case where the opponent is also rational, and can take (often conflicting) actions to his own benefit. There might be, for example, two agents (Player I and II) embedded in the graph, occupying different nodes. Each player chooses in turn a vertex adjacent to the current vertex (an edge). Let G be the graph of *Figure 4*, whose vertex set V is partitioned into sets V_1 and V_2 and the edges connect vertices belonging to different sets. We use white circles for the positions of Player I and black ones for those of Player II. Thus $V_1=\{1,3\}$ and $V_2=\{2\}$. And we consider that Player I plays on vertices in V_2 and Player II on vertices in V_1 (the position they do not own). Decision edges are shown in relation to Player II. A play is a path in G. If we start the system in a vertex that is in V_1, then Player II plays and otherwise Player I plays. In this example, Player I wins the game by always choosing node 3, because the resulting path will never pass by *1*. For a vertex set V, the attractor of V for Player I is the set of vertices from which Player I can induce a visit in V. The complement is a set that is a trap for Player I.

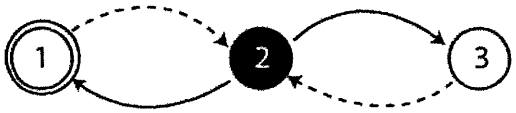


Figure 4: Simple game on a graph.

The game defines the different levels of the self-regulatory network, also called the local and global levels. The local level defines controllable actions, actions allowing the player to influence the possible outcomes of the overall game. This distinction is another fingerprint of a self-regulatory network, since they are necessarily spatially distributed. So, for example, in biochemical network of cells, although individual cells directly influence only their closest neighbors, they are globally functional. In a neural network, individuals can emit activation tokens only to close neighbors, etc.

---

[1] A group binary operation can be viewed as a mapping, since every ordered pair of elements w_1 and w_2 of a group

1.3 Application

The contribution of this thesis is three-fold. At the artificial intelligence level, we explore one approach to knowledge representation in terms of self-regulatory networks. In the more specific level of machine learning, we explore the use of game theory in classification. And in conclusion, we show how we can represent shapes in computer vision with the first and learn the corresponding representations with the second.

The learning problem is: given different sets of examplar graphs (each set corresponding to a different representational hypotheses), how to learn the corresponding self-regulatory network and how to use it to verify the different hypotheses? This is the problem that will occupy most of this thesis.

In some straightforward sense, there is a (often unnamed) relationship between dynamical systems theory and subfields of machine learning, especially time series analysis. The connection is in the common goal of understanding systems that change over time. The difference is in that the first focuses on describing systems' patterns of behavior while the second in identifying them. Beyond the mathematical analysis of algorithms (proving convergence results and such), more intricate concepts from dynamical systems theory - such as the concept of attractors - haven't been fully studied in relation to learning yet. The main difficulty lies in formulating these concepts computationally, in special formulating versions that can cope with noise and be used in real-world applications. In that direction, we have found useful to look at the problem of model inference and experimental design as a game played on a graph.

The traditional problem of ensembling classifiers can be seen as decision making when one faces an intelligent counterpart, a counterpart that is influenced not only by the static properties of the world (or the data) but also the decisions of other classifiers. In the view that we take here, the actions or tests of one classifier and the actions of another form a dynamical system, whose behavior we study using game theory. The main difference between the classical and game-theoretical approaches to optimization lies in the different concepts of what is a solution. Under the game theoretical view, a solution is a process of negotiating the probabilities and values of losses and gains until some stable, system-wide middle ground is found. We have assessed that many researchers today agree that the game theoretical view is a very attractive one, although it is still not clear how to put its concepts to good, practical use.

---

there corresponds a unique element w_3 of the group such that $(\Gamma_1, \Gamma_2) \rightarrow \Gamma_3$.

Game theory is currently used in machine learning in its normal, one-round or static form. For example, when we must design classifiers to perform over a set of prior probability densities and we use a minimax criteria so that the worst overall risk is minimized [5]. Within the machine learning scope, the main contribution of this thesis is to, conversely, attempt to formulate and study a "game of classification" in its sequential or dynamic form – where players make choices over time.

As validation, we will show that the view developed here can be directly used to address the object recognition problem in computer vision and that it leads to an alternative to current state-of-the-art systems. Namely, the problem we will engage is: given a familiar object (or class of objects), we humans have little difficulty in recognizing it irrespective of its particular instance, position, size, surroundings, illumination, etc. How do we achieve this, and, how can we get a computer to do the same? We are unaware of any use of game theory in object recognition.

2. Background

Our main interests are on machine learning and computer vision, and although we start this chapter with a few comments that situate our view on the larger context of the cognitive sciences, we quickly turn to summarizing recent related work on object representation and recognition, and then to the problem of learning, by reviewing some fundamental work in machine learning.

2.1 Cognitive Sciences

The use of networks as a framework for knowledge representation is widespread [4], we have argued for an alternative view, of representations as self-regulatory (adaptative) networks. Many areas of cognitive science seem to have stepped into a common obstacle, a definite conflict between the "variability and stability" of representations, and serious problems that appear in current research can be seen as instantiations of this same fundamental problem: "objects cannot map directly into a set taxonomy because the objects of mental life are not themselves stable entities... Mental events are naturally adapted to context ... thoughts about frogs in restaurants differ appropriately from thoughts about frogs in ponds because the perceived object and the remembered knowledge are made in and from the context of the moment" [6]. The issue has been raised in psychophysical approaches to perception [7], learning [10] and recognition [11]. It appears in semantics (the "semantic indeterminacy problem") [9]. And reappear in the study of cognitive processes and memory [8]. This is also related to the shaping influence of short-term memory and task dependent information [12] (in reading, detection of scene changes, attentional blink, repetition blindness and "inatentional amnesia"). The central question raised is: how can categories be locally variant but globally invariant? In general, we will argue for the notion that representational units are not to be seen as canonical surrogates of an external world, but part of a process of conceptual accommodation to situations in which they become useful. In this view, representations are more like reenactments than re-presentations.

2.2 Attributed Graphs

The use of graphs in Computer Vision is as old as the field. Attributed Relational Graphs (ARGs) are perhaps the most popular structural descriptions of visual patterns [13][14]. The ARGs' nodes correspond to unary features in the image (such as preprocessed line segments, regions, etc.), while the edges correspond to binary features (such as distance, angle, length ratios, etc. between unary features). This line of work has focused on matching algorithms, comparing the scene ARG to a stored graph of an

object as means of recognition. Deterministic linear least squares [16], graph eigenspectra [15], probabilistic optimization models (including probabilistic relaxation labeling) and hopfield networks have been used [17][18]. Inexact graph matching and multi-subgraph matching have been studied by Bunke and coworkers [14]. Generally, this approach requires two pieces: a shape descriptor and a shape matching algorithm.

A more recent and successful instance of this paradigm comes from the work of Belongie et al. [19]. The "shape context" is a new descriptor consisting of a point (edge element) log-polar histogram localized in different positions of a shape. Graph matching algorithms are then used to find correspondences between these features in an image and a model.

2.3 Classification Trees

A decision tree [20] is a classifier with a tree structure where each node is either a leaf node, indicating a class of instances, or a decision node that specifies some test to be carried out on a single feature value, with one branch for each possible outcome of the test. A decision tree can be used to classify an instance by starting at the root of the tree and moving through it until reaching a leaf node, which provides the classification of the instance.

In short, tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a threshold level or a regression) in each one. They are conceptually simple yet powerful. Typically to learn a classification three, one start with an exhaustive set of features in low-dimensional versions of the image set, and choose a subset and an ordering of these features capable of quickly decreasing uncertainty over the classification decision. In vision, classification trees have been used with many different features and node splitting and purity criteria [21].

A serious problem with this approach is that it looses sight of the concept of a shape. Objects are described by a bag of cues, which have no geometrical interpretation. Because of that, these models are more useful for object detection rather than recognition. They can be used to recognize instances of objects (an instance of a book, of a mug, etc.) but suggest no clear mechanism of abstraction across these instances, which is off course the crux of any notion of representation. Also, it is not clear how to extend this approach gracefully for classification over several classes, which is essential to object recognition (and move beyond the simpler foreground/background binary classification, as they are more commonly used today).

2.4 Graphical Models

Dynamical systems are used to model physical phenomena that changes over time [23]. This section provides a brief characterization of dynamical systems in terms of graphical models. In special, we investigate discrete-time, finite-state, stochastic models.

A dynamical system related to a specific physical phenomenon consists of a set of all 'states' of the system together with laws that determine the time evolution of states (also called its law of motion). A state is a picture of the phenomenon at a given moment. The state space of a dynamical system is the set of all potential states of the system, and changes in the system are transitions between these states. We begin by assuming that X is a variable representing the state of the dynamical system, and that the state space can be decomposed into a set of state variables $\{ X_1, X_2, ..., X_m \}$.

A way of representing a finite-state dynamical system is with its state-transition diagram. *Figure 5(a)* illustrates the state-transition diagram for a dynamical system with three states named *1* through *3*. Nodes represent different states and edges represent possible transitions between them. In diagrams with stochastic transitions, the out-going edges are labeled with probabilities such that the sum of the labels on the out-going edges sum to one.

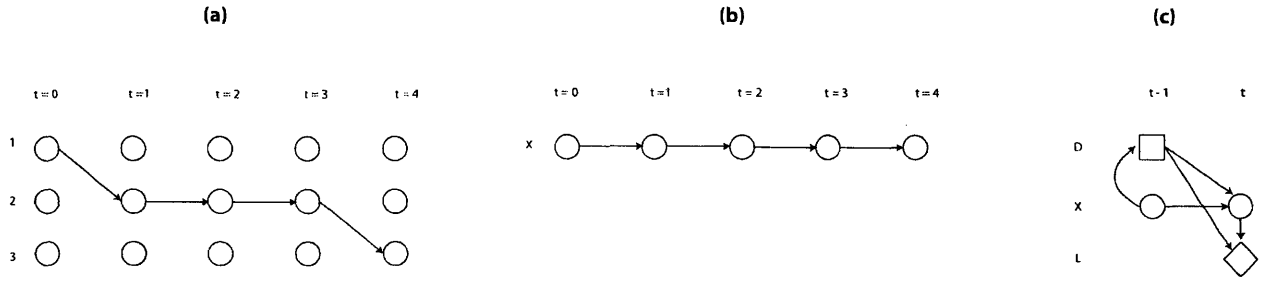


Figure 5: Graphical models.

A graphical model [22] is a directed acyclic graph, where nodes amount to variables and edges to functional dependencies. An edge from X_1 to X_2 designates that X_2 is functionally dependent on X_1. In a graphical model, X is a random function usually defined by a conditional probability distribution $P(X \mid Incoming(X))$ where $Incoming(X)$ indicate the nodes with edges ending in X in the graph. The dependency

is defined by a density function. In this way, graphical models correspond to Bayesian networks. A Bayesian network describes efficiently an m-dimensional joint probability distribution $P(X_1,...,X_m)$, which can be factored as a product of lower-dimensional conditional and marginal distributions $\prod_{i=1}^{M} P(X_i | Incoming(X_i))$. If X_i has no incoming edges, its marginal distribution is $P(X_i)$. *Figure 5(b)* represents the information in *(a)* in terms of a graphical model by collapsing the possible states at time t into a variable X_t. The variable X_t indicate the state of the system at time t that may depend on the state of earlier times. In many cases, the current state tells us all we need to know in order to compute the distribution governing the next state. In cases in which the current state provides a sufficient summary of the past for the purpose of predicting future, states are said to be Markovian.

If the stochastic process governing transitions is the same at all times, a graphical model can easily represent a dynamical system. Such process is said to be stationary. We can represent a stationary process as a graphical model consisting of two nodes representing the state at time $t-1$ and t. A special case of Bayesian network can represent a dynamical system, it consists of a grid of variables with size $T \times M$. Columns are called levels. The variable in the tth level represent $X_{i,t}$ for $1 \leq i \leq M$. Edges between variables in the same level indicate that the variables are correlated. The first level represents the initial-state distribution while the later stages represent the state-transition distribution.

It is possible to catalog approaches to learning graphical models in respect to the assumptions they make. Generally, methods assume that the samples and parameters are independent, the complete set of variables and their respective values are known, the structure of the graph is known, and only a subset of the variables are observable. In this thesis, we will focus on the case where the variables and graph structure are unknown. The computational picture for the existing learning methods is diverse and often convoluted. There are however good, popular surveys [24][25].

2.5 Decision Processes

It is useful to distinguish state variables in a dynamical system by the different roles they may play in the system, such as observations and actions. It is useful then to think of an agent that is capable of observing the system and take actions based on both its observations and the gains it associates with the different states. The graphical model in *Figure 5(c)* is an example of what is often referred to as an influence diagram [26]. While observations are drawn with the typical circles, the actions are represented by decision nodes and drawn with boxes. They are frequently deterministic functions of previous observations. Losses and gains are represented with value functions which are real-valued functions of

state variables. Value functions are distinguished graphically as diamonds. The value associated with a particular state at a particular time is the sum of the value functions applied to the values of the corresponding state variables. The graphical model in *Figure 5(c)* depicts a two-level stationary network representing an agent embedded in a dynamical system whose actions are dependent only on the current observation and whose value function depends only on X_t and D.

Influence diagrams are used to specify decision problems in which the goal is to choose a plan for acting that maximizes a known value (or loss) function. For example, the influence diagram in *Figure 5(c)* identify the problem of mapping observables into actions (such mapping is called a policy) that maximizes the expected cumulative rewards over a finite number of stages. The corresponding decision problem is often called a partially observable Markov decision process, a generalization of a completely observable Markov decision process in which the decision making agent can directly observe the state of the system at each stage in the process.

Although they have been typically studied separately (perhaps with the exception of [27]), the two problems - that of learning a model and a policy – are clearly related. In this thesis we address the two problems with a common framework, constructed with concepts from dynamical system theory. In particular, the concept of a graphical attractor will prove useful. Attractors are states the system often returns to, and finding such sets will help understanding the behavior of dynamical system.

3. A Game on Graphs

Decision theory studies the ways in which a decision maker may choose among a set of alternatives in light of their possible consequences. Decision theory can be seen as a theory of one-person games, or a game of a single player against nature. In the more complex cases where different decision makers interact (i.e., their preferences cannot be captured by one monolithic objective function), it corresponds to game theory. Both decision [4][28] and game theory [4][29] have stimulated a wide range of techniques in artificial intelligence.

In this section, we systematically study a game on a graph. This is a game in which each player chooses sequentially the position he wants to play next. The result of the play is a sequence of edges $e_1e_2e_3$.... This chapter contains the analytical motivation for the entire thesis, as well as the outline of the method. While it's easy to imagine several different methods for learning and using a graph dynamical system, this chapter will tell us which ones will work and why.

3.1 The Field

Consider a fixed graph $G=(V, E, X)$, consisting of a set of nodes V, and directed edges between the nodes E. Nodes are labeled, taking values in the set X. We also refer to labels as configurations, interchangeably. An edge is an ordered pair (u,v), where $u,v \in V$ - indicating that is possible to transform u into v; the set of edges is thus the set of similarity mapping on the nodes. In the simplest case, a graph may have labels taken from the set $X=\{1,0\}$ indicating when a node is "activated", while links with an identity transformation indicate when nodes are mutually activated. We write uEv to designate this relation. There is a path of length k between two nodes if uE^kv. In addition to the graph, we have a time coordinate which proceeds in integer ticks: $t \in N$. We need also a way to refer to the combination of a position and a time; we will write it using a parameter pair $X(v,t)$, respectively. When only one argument is used, we are referring to the position alone, $X(v)$.

At each node, we have a random variable $\boldsymbol{X}(v,t)$, taking values on X; this is the random field. We will write $X(v,t)$ for both $X(v,t)$ and $\boldsymbol{X}(v,t)$, letting context determine to which we refer. Accordingly, edges are operators on the simplex $X \times X$. Let $\aleph$ be the maximum area of influence of the node v. Now define the local neighborhood of the node v as the set of all nodes whose field could be influenced by the field at v (including v); likewise the global neighborhood is all the nodes whose fields could not be directly

influenced by the field at v. We can think of the relationship between local neighborhoods at different nodes in the graph as yet another, further spatially extended, graph labeled by the local neighborhood configurations. In this case, the label set is an "alphabet" of local graphs. The subscript k in $X^k(v,t)$ will indicate which such level we are referring to. Thus, $X^3(v,t)$ is the global neighborhood of $X^2(v,t)$, which in turn is $X^1(v,t)$'s. *Figure 6(b)* is a schematic illustration of the local and global (doted) neighborhood ($\aleph^1 = \aleph^2 = 1$).

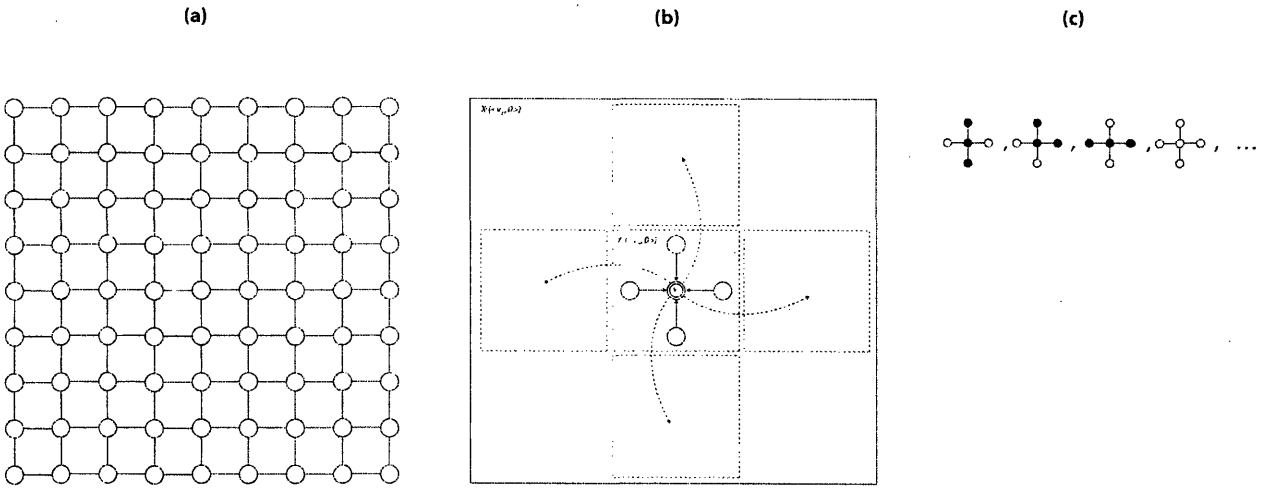


Figure 6: Field.

More precisely, the local and global neighborhoods are defined recursively by

$$X^{i+1}(v,t) = \bigcup_{0 < j < \aleph} \{X^i(u,t+j) \mid \vee u E^j v\},$$

and $X^0=\{0,1\}$. *Figure 6(c)* shows a few local configurations for $k=1$ and $\aleph = 1$.

3.2 The Game

Imagine a dynamic system which moves through a sequence of states $v_0,v_1,v_2,\ldots$ at times $0,1,2,\ldots$. We associate the states with different nodes in a graph. We suppose that the motion is controlled by a sequence of actions $e_0,e_1,e_2,\ldots$, which we associate with edges on the same graph.

We now formulate a game as a model for the control of dynamical systems, and its set of winning strategies which requires only a limited memory of the past. We shall begin with deterministic examples which do not involve any unknown parameters in the system's law of motion (in our case, corresponding to a fixed edge set). There is a vast literature on the use of automata to control deterministic systems (for example, in motion control) [30][31][32][33][34][35][36][37]; most of the work however focuses on proving certain combinatorial results and conditions of the fixed system and do not involve any learning. Statistical problems are usually more difficult to formulate and solve, and the determinism in the law of motion is a usual assumption in the study of Markov Decision Processes, for example. Ignorance about parameters in the law of motion can however lead to serious complications in the formulation of decision problems, and we address stochastic motion in turn.

A game starts at a position $v \in V$. Suppose that we are interested in the random variable $X^0(v, 0)$, and each player at that location have an hypothesis-set about its true label. A player wins locally if he predicts the label correctly. For convenience, let us start with labels $X^0=\{0,1\}$ and assume that Players I and II have known guesses for position v, respectively $x_I=1$ and $x_{II}=0$. Let Π be an unobserved random variable denoting the winner and S^0 a function $S^0 : X^0 \rightarrow \{ I, II \}$, where $S(x^0) = \pi$. In general, this assignment can be noisy, yielding a random function $S(x^0)$. The function S^0 partition the labels $X^0(v, 0)$ in two sets s_I and s_{II}. We say that Player I "owns" the labels in s_I, and Player II the labels in s_{II}. A node where $S(x^0) = I$ favor the hypothesis associated with Player I - the case where $x_I(v)$ is the observed label. If this is so, Player II (namely, the one that has lost) chooses the next vertex to play by picking an edge e. This represents an attempt by Player II to "repair" the situation.

In this scenario, at time 0, Player I is the winner with a prior probability density $S_0(x^0)$. In general decisions which control the behavior of the system at times 1,2,,... will lead to a succession of posterior densities $S_1(x^0)$, $S_2(x^0)$, $S_3(x^0)$,..., allowing for any extra information about S^0 to be gathered as time passes, leading to the posterior density function $S_t(x^0)$ which is the estimated state of the system at time t. We say that Player II makes a prediction for the second node $u \in V$ given by its chosen edge $e : x_{II}(v,0) \rightarrow x_{II}(u,1)$. Hopefully, for Player II, the prediction is correct and the extra observations may point at him as the winner, and in this case the control goes back to Player I, and so on.

Given an initial distribution and all edges/transformations, a function S^0 makes the graph G bipartite, i.e. its vertex set V is partitioned into subsets V_1 and V_2, and the edges connect vertices belonging to different sets. Therefore, we consider that I plays on vertices in V_2 and II on vertices in V_1; and that a play is path in G. Thus, if the first node is in V_2, then Player I plays first and until he looses locally; otherwise, Player II

plays first, which defines the notion of a winning strategy for each player as a function from the set of paths into the set V of vertices.

To decide his next position (which edge to choose) a Player must take into account two factors: the possibility of winning there, and the paths (actions) the other player may take from there. Each of the sucessor's labels is a different local observation, and each may motivate the Player at that location to take a different action - that is, to choose a different edge. A deterministic memoryless strategy for Player I is a strategy which depends only on the last node of the path. We will define one such wining strategy, which we name an attractor. For a vertex set $F \subset V$, we define the attractor of F for Player I, denoted $A(F)$, as the set of vertices from which Player I can induce a visit in F. The complement $W(F)$ of $A(F)$ is a set which is a trap for Player I: Player II can force Player I to remain inside $W(F)$.

We now delineate how players may find their attractors. When studying the behavior of the nodes in a set $F \subset V$, it is useful to define a function that measures how long it takes for paths starting at a node $q \notin F$ to reach the set. We call this function the return time function. With discrete time, define the return time function $\sigma: V/F \rightarrow N$ by

$$\sigma(q) = \begin{cases} \min\{t > 0: \text{for some } u \in F \text{ and } q \notin F, qE^t u\} \\ \infty, \text{if } qE^t u \notin F \text{ for all } t > 0 \end{cases}$$

This defines an equivalence class among the nodes in V:

$$v \sim u \longleftrightarrow \sigma(v) = \sigma(u)$$

where $u, v \in V$. This equivalence relation decomposes the nodes V/F into sets:

$$A_i = \sigma: \{t = i\}$$

The return of a vertex $q \in V$ is the smallest integer t such that $q \in A_t$ where A_t is an increasing sequence of subsets of V defined by $A_0 = F$ (some initial subset of nodes) and inductively,

$$A_{t+1} = A_t \cup \{p \in V_2 \mid q \in A_t \text{ for some } (p,q) \in E\}$$
$$\cup \{p \notin V_2 \mid q \in A_t \text{ for every } (p,q) \in E\}.$$

The attractor of A is then $\bigcup_{i \geq 0} A_i$. The strategy of Player I on this set consists in decreasing the return time function. The strategy of Player II consists in keeping off the positions of $A_I(F)$. Starting in the next section, we will examine stochastic versions of the concepts above, and in the next chapter, an algorithm to learn such strategies for our game on graphs.

3.3 Stochastic Motion and Strategy

A decomposition A_i partition the nodes in V into sets with distinct losses (utilities) to the Player – namely, different expected times necessary for the player to win the game. If the player chooses a node in A_1 as successor, for example, he will expect to win the game in one round, and so forth. This section considers a stochastic version of the game and its wining strategy; we start with pure strategies and on the subsequent section we move on to mixed strategies.

The problem for a Player I located in a node v is to select the next node and predict its label. Consider then the Player chooses a node u in v's neighborhood of size $\aleph$, and some particular label $x(u)$ at that location. This choice corresponds to a decision edge and its inverse to a causal edge (with length 1). The causal edge asks the probability of Player I being the winner if this edge is selected.

More precisely, let us assume that a player at $v \in V$ has, at any moment, a probability function and a loss function for each node and label in its neighborhood. These are defined in first instance over a single local label x_I at v by a transition probability distribution $P(X^0(v)=x_I \mid x^I(u))$, which measures the agent's degree of belief that $x^I(u)$ will cause x_I to be the actual label at that location. These scores fall on a scale from zero to one, and they sum to one. Also each label $x^I(u)$ has a loss $L(x^I)$, which measures how unsatisfactory to the player for x_I to be the actual label. These values fall on a linear scale with arbitrary zero and unit. The expected loss, or risk, of a node $u \neq v$ is a function of the form

$$R^0(u) = \sum_{x^u} P(x^0 \mid x^I) L(x^0 \bullet x^I)$$

The key to this definition is the probability $P(\cdot \mid x^I)$, where in traditional decision theory figures an unconditional distribution. Its value reflects a judgment about the neighbor's label potential to causally influence the local event. The quantity $R(u)$ then gauges the extent to which the node can be expected to *bring about* desirable or undesirable local outcomes.

As we have seen, in our game a local outcome is desirable to a player if it belongs to him, and undesirable otherwise. To account for that, we may go on to define the risk also for sets of local labels of which exactly one belong to a player or another. Such sets are partitions of labels, and we say that a partition holds at just those labels which are its members. For a partition Π of local labels, there is alternatively a distribution $P(X^0(v) \in S_\pi^0 \mid x^1(u))$, which we call the return probability for the partition π. For any partition π, we sum scores:

$$P(X^0(v) \in S_\pi) = \sum_{x^0 \in S_\pi} P(x^0 \mid x^1(u))$$

As S_π is a partition of local labels, there is also a partition of labels in neighbor nodes, in terms of the different effects in the local configuration. For that we need to define an equivalence class of labels, formally:

$$x_1^1(u) \sim x_2^1(u) \longleftrightarrow P(X^0 \in S_\pi^0 \mid x_1^1(u)) = P(X^0 \in S_\pi^0 \mid x_2^1(u))$$

We will look at this equivalence relation in detail in the next chapter. For the moment, let the variable S range over these classes of labels in the node u (in which case the $X^* \cdot S$ labels, for fixed X^* and varying S, are a partition of X^*). The rule of addictivity for probability, and for the product of probability and expected loss are:

$$P(X^0) = \sum_S P(X^0 \bullet s),$$

$$P(X^0) L(X^0) = \sum_S P(X^0 \bullet s) L(X^0 \bullet s).$$

The rule of averaging for expected loss of the node u in relation to v is then:

$$R_v(u) = \sum_S P(s \mid u) L(u \bullet s).$$

Thence we can get an alternative definition of expected loss. For any element s, let $[L=l]$ be the labels that holds at just those labels x for which $L(x)$ equals l. Since they are a are a partition:

$$R_v(u) = \sum_S P([L = l] \mid u) l.$$

The partition-set S is a useful tool to study the possible response of a node if selected to be a successor. And the measure $R_v(u)$ basically quantifies the overall risk of existing a causal edge between u and v (that is, a direct return from u). Still, the partition-set does not provide the complete picture of u's influence on v, since it considers only the paths of length one leading back to v. When a local label is observed, the best reaction may be to expect a transition to a different node, and a return from there – a return path of length two and so on. This issue will occupy most of the next chapter. For now, it suffices to mention the stochastic analog of the return function decomposition:

$$\sigma(x(u),\pi) = \begin{cases} \max\{P(X^0 \in S_\pi^0 \mid x(u)E'x(v)) : \textit{for some } v \in V, u \neq v,\ x(v) \in S_\pi, \textit{and } t > 0\} \\ \infty, \textit{if } qE'u \notin F \textit{ for all } t > 0 \end{cases}$$

This function measures the best amount of time for a Player π to allow a random walk starting at a node u with observed label $x(u)$ to run. In parallel with the deterministic wining strategy, the stochastic strategy is given by an increasing sequence of subsets of V defined by $S^0=F$ (some initial subset of nodes), and sketched by the procedure

$$S_{t+1}=S_t \cup \{p \in V_1 \mid \min_p R_{S_t}\}$$
$$\cup \{p \notin V_2, \mid \max_p R_{S_t}\}.$$

This leads to a minimax solution concept for the game. In the remainder of this chapter and thesis, we make this procedure more precise.

3.4 Mixed Strategy

So far we have considered only pure strategies for the game on the graph. A pure strategy defines a specific move, prediction or action that a player will follow in every possible situation in the game. Such moves are not drawn from a distribution, as in the case of mixed strategies where distributions correspond to how frequently each move is to be played. Suppose some local set of labels is chosen, let the variable Π range over candidate probability distributions for this configuration: functions assigning to each x^0 in the partition a number $p(x^0)$ in the interval from zero to one, such that the $p(X^0)$'s sum to one. Let $[P=p]$ be the neighbor labels that holds at just those configurations where the chances of the X^0's are correctly given by the function p.

In the mixed strategies version, we are also interested in the random variable $X^0(v,0)$, but each player at that location have as hypothesis a probability distribution. For convenience, let us assume that Players I and II have a probability density function given by either p_I or p_{II}, where p_I and p_{II} are known functions. Therefore, the game involves for a node v the random variable $X^0(v,0)$ and two alternative hypotheses about its probability distribution. Let x_1^0 be random samples of independent observations on $X^0(v,0)$ and consider the problem of reaching a decision, based on these data, whether the true distribution of $X^0(v,0)$ corresponds to p_I or p_{II}. A node where $S(x^0) = I$ favor the hypothesis associated with Player I - the alternative that p_I is the true density of observations.

We should maximize the expected utility calculated by means of such strategies. A formula defining expected utility in terms of configurations as consequents is:

$$R(u) = \sum_S \sum_p P([P = p] \mid s) p(s) L(u \bullet s).$$

This is the version we will use in the remainder of the thesis.

3.5 The Game

We have studied the games on a graph in versions of increasing complexity. In this section we give the complete definition of the game. Let's analyze then the game from a local perspective. Locally, a node has a set of outgoing and incoming edges for different levels, mirroring what we called decision and causal edges. For expository reasons it will be convenient to treat the combination of an outgoing and an incoming edge as a statistical experiment. Such an experiment might consist for example of sampling N locations in the graph (outgoing) and observing its labels (incoming). Typically, a Player will perform local experiments to accumulate confidence at the level k and only then move on to $k+1$.

Let's again assume a graph with vertices V, where Player I and II are playing in a node v_0. We have seen that there is a random function S^0 that assigns a winner based only on the local (label) observation at v_0. But that this assignment may be faulty, and the player may correct it by observing values at neighbor nodes in the graph. To begin with, the player has at his disposal outgoing edges as possible actions which he can take, given he doesn't know the state of its surroundings $X^1(v_0)$. Let then X^1 be the node's neighborhood and S^1 a random function $\boldsymbol{S}^1 : X^1 \rightarrow \Pi = \{ I, II \}$. If the Player decides to take a global action e without local experimentation, it suffers a loss of $\boldsymbol{S}^1$. However, the player can perform local experiments and thus decrease the loss by gaining some information about $X^1(v_0)$. For that, he must decide

which tests he is going to run, in what sequence, when to terminate the tests, and what to do when that happens. The player doesn't have complete knowledge about $X^1(v_0)$ because of the cost of the tests.

We first define the local and global sample spaces for the player. Locally, the Player has a postulated sample space X^k which specifies all possible results of the local tests that the player can run. Associated with the space X^k is the local strategy space Π with elements π, and a function p defined on $X^k x \Pi$ such that, for a fixed π, p_π is a probability distribution on X^k. From the point of view of Player I, the case where $\pi \neq I$ represent the local mixed strategies for the other players. There is also the global sample space X^{k+1}, associated with a conditional distribution, defined on $X^k x \Pi$ but conditioned in X^{k+1}.

As an illustration, consider the following example: Player I has to decide whether its local label is $X^0(v,0)=\{0,1\}$, based on tests on surrounding three nodes. For each neighbor $u \in V$ tested, $X^0(v,1)$ is the random variable over $X^0(v,1)=\{0,1\}$. Then, the sample space consists of the eight configurations:

$x_1=(0,0,0)$ $x_2=(0,1,0)$ $x_3=(1,0,0)$ $x_4=(1,1,0)$
$x_5=(0,0,1)$ $x_6=(0,1,1)$ $x_7=(1,0,1)$ $x_8=(1,1,1)$

Figure 7: Sample space example.

If so, Player I must now associate with each outcome of a test an edge e in E (defining how the Player will react to the result of that test). This is a stochastic strategy, that is, a function D mapping X^k into E. It can be considered as a partition of the set X^{k+1} into mutually exclusive subsets

$$S = \{ x : d(x) = e\}$$

whose union is X^{k+1}. If the result of a single test is contained in S, edge e is chosen. Thus, if we return to the previous example, if E consists of two elements e_1 and e_2, then a strategy d is a division of X^{k+1} into sets S, such that, if $x_1 \in S$, the edge e_1 is chosen, and if $x \notin S$ then e_2 is chosen.

Each neighborhood $x \in X^{k+1}$ is in reality an N-tuple $x^{k+1}=(x_1^k, x_2^k, ..., x_N^k)$ where each x_i^k, $i=1,...,N$, is an instance of a random variable over different neighbors. Consider the case where the player expect to

observe only the first T neighbors before a return. If so, a strategy assigns to each neighbor x^k of X^{k+1} two numbers, an integer T which specifies the number of neighbors of x^{k+1} needed to be sampled before wining the game, and an element e of E which specifies the action that will be taken in each case (where E is the set of edges at the level $k+1$). Thus a strategy is a decision function d with values $d(x)=(\sigma(x), e)$ with range in TxE.

We will take a strategy to consist then of a partition T of X^{k+1} with elements $T_1,...T_N$ together with a sequence of functions $d_1,...,d_N$ such that for each t, d_t is a function of the t neighbors in X^{k+1}. In the next section we turn to the forms the loss and cost function takes.

3.6 The Solution

We first outline the game solution before formulating it completely. We have seen that each player $\pi \in \Pi$ at a node v_0 is related to a different hypothesis, a prior distribution for X^K. And a strategy for the players (say, Player I) consists in observing T neighbor nodes $v_1,v_2,v_3...,v_T$. To each observation at time t correspond a conditional distribution $P(X^k(v,t+1) \in S_I \mid X(v_1)^k,..., X(v_t)^k)$ aimed at predicting X^k from X^{k+1}. The loss suffered by choosing v is given by $-< log\ p(X^k(v_t) \in S_I \mid X^{k+1}) >$. The optimal strategy is a sequence of functions $d_1(v_1),...,d_N(v_N)$ which minimizes the worst case loss incurred in this game:

$$\arg \min_{e} \max_{e} - \left\langle \log p\left(X^{k}(v) \in S_{I} \mid X^{k+1}\right)\right\rangle.$$

Suppose that the Player I wishes to select a single node to construct its reaction. The rationale behind the minimax principle is for the player to focus on the riskiest regions of the graph. We show that, under regularity conditions, minimizing the generalized entropy constitutes the central step towards finding the optimal act against opponents for the log loss score, and it leads to an equilibrium solution in the game. The general relationship between game and information theory have been studied specially in statistics, for computational takes on this see [38][39]. This line of work may be used to derive tighter bounds and convergence guarantees to our game, but this will not be explored here.

Reconsider the example in *Figure 7*, but with truncated values $\sigma(x)$ as assigned by the return time function. The strategy maps the nodes to binary words. If the winning frequency of individual paths are known, efficiency can be related to the average return time. The shorter the return, the higher the efficiency (to use information theoretic terms). Thus the efficiency depends both on the distribution $P(X^k(v)|X^{k+1})$ and on the return time function σ - which we have taken to be the map providing these

lengths *(e.g.,* $\sigma(x_1^k)=3,\ldots,\ \sigma(x_6^k)=4$*)*. Then the average return time may be written as $<\sigma,P(X^k(v_0)|X^{k+1})>$, in bracket notation. There is then a mapping between $P(X^k(v_0)|X^{k+1})$ and σ, as:

$$\sigma(x) = -\log p(x),$$
$$p(x) = e^{-\sigma(x)}.$$

where $p(x)$ are the conditional probabilities for a strategy as defined in the last section.

The cost function, seen from the point of view of Player I, is the map $P(X^k(v_0)|X^{k+1}) \times \sigma \rightarrow [0;\ N]$ given by the average return time. The cost function can be interpreted as the average time necessary to win the game, and it is natural for Player I to minimize this quantity. The logic in assuming that Player II has the opposite goal comes from game theory (a player cannot do better than supposing that the opponent behaves rationally in a way which is the least advantageous to him). This turns out to correspond to Jaynes' reasoning behind his Maximum Entropy Principle [40]. Here however the principle is applied to classification, relating the prediction of X^k from X^{k+1}. This suggests for the player to seek the distribution with minimal mutual information, which will motivate the method we study in the next chapter.

The notion of equilibrium applied to the graph dynamics game says that Player π should consider the minimum of $<\sigma,P(X^k(v_0)|X^{k+1})>$ over σ, which is the optimal response for Player I. The entropy can be conceived as minimal average return:

$$H(X) = min <\sigma,\ P(X^k(v_0)|X^{k+1})>.$$

The minimum is attained for the strategy adapted to X^{k+1}, assuming that $H(P(X^k(v_0)|X^{k+1})) < \infty$.

Seen from the point of view of Player II - the adversary - the optimal performance is thereof achieved by maximizing entropy. The maximal value this player seeks is

$$H_{max}(P(X^k(v_0)|X^{k+1})) = supH(P(X^k(v_0)|X^{k+1})).$$

On the side of Player I we can consider for each σ the risk

$$R(\sigma|P(X^k(v_0)|X^{k+1})) = sup<\sigma,\ P(X^k(v_0)|X^{k+1})>,$$

and then the minimum risk value is

$$R_{min}(P(X^k(v_0)|X^{k+1})) = inf\, R(\sigma|P(X^k(v_0)|X^{k+1})),$$

which is the value Player I seeks. We have now considered each side of the game. Combining them, we encounter concepts from the theory of two-person zero-sum games. Thus, an equilibrium solution is one where $H_{max}(P(X^k(v_0)|X^{k+1})) = R_{min}(P(X^k(v_0)|X^{k+1}))$, and the value of the game is $H_{max}(P(X^k(v_0)|X^{k+1})) = R_{min}(P(X^k(v_0)|X^{k+1}))$.

While for Player II the optimal strategies comes from a distribution $p_{II} \in P(X^k(v_0)|X^{k+1})$ with the property $H(p(X^k(v_0)|X^{k+1})) = H_{max}(p(X^k(v_0)|X^{k+1}))$, for Player I the optimal strategy is characterized by the function σ with the property that $R(\sigma|\, p(X^k(v_0)|X^{k+1})) = R_{min}(p(X^k(v_0)|X^{k+1}))$, which is a minimum risk strategy.

4. Learning

In the last chapter, we've studied a game where players situated in a node act by choosing edges – that is, their subsequent positions. When that happens, we can say the source node has activated the target node. An attractor set for a player is a set of nodes which, as a group, cause its own activation. Thus if u activates v, and v causes u, then $u + v$ comprise an attractor set. In an environment where u appears followed by v, $u + v$ can remain activated in detriment of other structures. Let's say that a set of nodes wherein each node's activation is caused by some other node in the set exhibits causal closure. It is of course highly unlikely that two nodes u and v that just happened to co-occur would happen to cause each other. However, this is more likely than the existence of a single node causing its own activation (a node with no repairs or contingencies, thus a fixed structure). But, when nodes interact their diversity increases, and so does the probability that some subset of the graph reaches a critical point where there is a causal pathway to every member.

If we want to devise a way to iteratively learn self-regulatory networks, we need to answer the question: once a set of nodes has achieved some amount of causal closure, which other nodes should it choose to incorporate the attractor set? The importance, or utility, of a new node relative to the existing set should express the new node's capacity to, if activated, cause (or lead back to) the original set – making the original set even more useful. An iterative, selective process using this quantity can elicit cliques in the network with progressive joint complexity and closure, eventually transforming some subset of nodes into a web for which there exists pathways counteracting a diverse set of contingencies. If we think of the graph as a language automata, this is an automata expansion procedure that attempts to gather a few initial nodes into increasingly resilient structures.

We have characterized a decision function with two elements: a partition and an action. In this section we take each of these elements in sequence.

4.1 Partition

The following is a straightforward derivation of concepts from information theory and statistical mechanics [41]. Mutual Information clustering has been considered in machine learning at [42],[67],[43] and [44].

Take a game field as defined in the last chapter. It offers the prior state-space for a dynamical system that moves through the graph's nodes. We have seen that several moves can have however similar influences in the system's local behavior. As result, the effective state space of the system is in reality more succinct.

Suppose we can observe the random variable X^{k+1} and wish to predict S^k. Any function on the global neighborhood defines a local statistic, a good local statistic outlines the influence of the neighbor nodes in what happens locally. Let S^{k+1} be a partition function $S^{k+1} : \{x_1, x_2, \ldots, x_n\} \rightarrow \{s_1, s_2, \ldots, s_l\}$, where $l << n$. Because S^{k+1} is a function of X^{k+1}, $I[S^k; S^{k+1}] \leq I[S^k; X^{k+1}]$. A statistic $S(X^{k+1})$ is sufficient when $I[S^k; S^{k+1}] = I[S^k; X^{k+1}]$, see [41]; that is to say, a sufficient statistic is a function of the data that is as informative as the original data.

For a node $v \in V$ and a node $u \neq v$ in its neighborhood, let π denote the local winner at v. Let $S^k{}_\pi$ be the set of labels in v that belong to π. Consider a *fixed* player $\pi' \neq \pi$ and a *changing* neighbor configuration $x(u)$. There is a certain conditional distribution over local configurations, $P(S^k{}_{\pi'} \mid X(u) = x, X(v) \in S^k{}_\pi)$ - which we shorten to $P(S^k \mid x, \pi)$. Let us say that two neighbor configurations are equivalent if they have the same conditional distribution,

$$x_1 \sim x_2 \longleftrightarrow P(X^k(v) \in S^k_\pi \mid x_1, \pi) = P(X^k(v) \in S^k_\pi \mid x_2, \pi)$$

The statistic S^{k+1} is simply the equivalence class of neighbor configurations over all neighbor nodes of v (and not the single node u).

$$S^{k+1}(v) = \{s^{k+1} \mid P(S^k \mid s^{k+1}) = P(S^k \mid x^{k+1})\}.$$

where $S^{k+1}(v) \subset X^{k+1}(v)$. The equivalence class (or state), written $S^{k+1}(v)$ is sufficient. Plus, the local and global neighborhoods are independent given the local state. In a broader scope, this equivalence relationship corresponds to a view on causality first advocated by Wesley [59][44]. And $H[S^{k+1}]$ is the amount of information on the neighborhood about the local state.

We now turn to an algorithm that will devise the set of neighborhood states from data. We first select the initial node v, and for each configuration estimate a local distribution over Π. We have now the local winner for each local configuration, $\underset{\pi \in \Pi}{\operatorname{argmax}} P(\pi \mid x)$. We will return to the issue of how to select the initial position (node), in this section we will focus on how a player organize its neighborhood to cope

with an unfavorable outcome on the local game. To each node u in its neighborhood of size $\aleph$, we determine which neighbor configurations are in fact present in the data. Then, for each neighbor configuration x, we estimate $P(S^k \mid x, \pi)$, using the Parzen window method. We can then cluster neighbor configurations based on the similarity of their conditional distributions. These clusters are then the estimated states of the dynamical system. We associate with each cluster a different conditional distribution and an entropic measure. The first is the weighted mean of the distributions of the configurations it contains, and the second is $H[S^{k+1}] = -\sum_i P(s_i) \log_2 P(s_i)$.

Perhaps the most straight-forward model for the joint intensity distribution $P(S^k \mid x, \pi)$ would be a mixture of Gaussians, but the mixture is sensitive to initialization parameters and in some cases results in an inaccurate prior model of the joint label function. We therefore estimate the underlying distribution based on the Parzen window density with Gaussians as the windowing function, see [46]. It is obtained by centering a small Gaussian kernell around each training point. In practice then the method samples directly the training data and provides a better explanation of the label correlations than the Gaussian mixtures that require the prior estimation of various parameters. For computational efficiency this probability distribution is stored in a look-up table, see [47].

We are interested in clustering (quantitizing) X^{k+1} into l disjoint (hard) clusters, which corresponds to finding the map $S : \{x_1, x_2, \ldots, x_n\} \rightarrow \{s_1, s_2, \ldots, s_l\}$. We will judge the quality of a cluster by its ensuing conditional loss in mutual information, $L = I(S^k, X^{k+1}) - I(S^k, S(X^{k+1}))$. For a fixed distribution $P(S^k, X^{k+1})$, $I(S^k, X^{k+1})$ is fixed; hence minimizing the loss amounts to maximizing $I(S^k, S(X^{k+1}))$. The loss in mutual information can be expressed as the distance of $P(S^k, X^{k+1})$ from the approximation $P(S^k, S(X^{k+1}))$; that is, $L = D(P(X^k, X^{k+1}) \mid\mid P(X^k, S(X^{k+1})))$, where D is the relative entropy. This form is, in turn, equivalent to:

$$L = \sum_s \sum_{x^{k+1}: S(x^{k+1}) = s} p(x^{k+1}) p(s^k \mid x^{k+1}, x^k) \log \frac{P(s^k \mid x^{k+1}, x^k)}{P(s^k \mid S(x^{k+1}), x^k)}$$

For each global configuration, x^{k+1}, we find its cluster index iteratively by $S(x^{k+1})^i = \underset{s}{\operatorname{argmin}} D(P(S^k \mid x^{k+1}, x^k) \mid\mid P(S^k \mid x^{k+1}, x^k))$. This corresponds to an iterative and agglomerative clustering procedure. We repeat, $i = i + 1$, until $D(P(S^k, X^{k+1}) \mid\mid P(S^k, S^i(X^{k+1})))$ stops decreasing under a quality threshold. As we give this procedure more and more data, it converges in probability on a local optimum set of states. As a practical matter, we need also to limit how far locally,

and globably, the neighborhood will extend, $\aleph^k$ and $\aleph^{k+1}$. We give the pseudo-code at the end of this chapter.

4.2 Strategy

Last section equipped us with the means to represent the interaction/relationship between two nodes. We are now in position to increase our scope to the entire graph. The background for this section consists of basic sequential stochastic decision processes [54][56][57][58].

We are interested in the return time function with the property

$$\sigma(s^{k+1}) = \textit{minimum total expected cost}$$

where the expectation is conditional on a given initial state, s^{k+1}. The problem we have defined on the last chapter involved associating with each state a strategy size given by the return time function, and its solution indicates the optimum strategy sizes for a process starting at arbitrary states. At each state that occurs, a decision must be made whether the player should move to some other position in the graph or not. Consider then a Markov chain with possible states in the set of partitions, s_0^{k+1}, s_1^{k+1}, s_2^{k+1}, ...s_l^{k+1} whose motion is governed by a probability law specified by a transition matrix P, where $P=(p_{ij})$ is a $l \times l$ matrix with elements $p_{ij} \geq 0$, such that $\sum_j p_{ij} = 1$. For any given state $0 \leq i \leq l$, the probabilities p_{ij}, $j = 0,1,2,...,l$, describe the distribution of the next state. We have a Markov system $s_0,s_1,s_2,...s_l$ in which every s_t belongs to the state space $\{0,1,2,...,l\}$ and, given that $s_t=i$ at any time t, the next state is a random variable with the conditional probabilities $P(s_{t+1} = j \mid s_t = i) = p_{ij}$.

If the underlying random process is Markovian, a decision on whether to increase the strategy size or not can be based on comparing the return probability for the present state with the expected return after transitioning to another state. Suppose that for player π the return probability associated with being in state i is $r_i = b \cdot P(s_\pi^k \mid s_t^{k+1}, ..., s_0^{k+1}) \geq 0$, which is readily calculated from the distribution associated with the neighbor state s_π^{k+1} and the parameter b. The immediate cost associated with a decision to move from state i is $\Delta h_i \geq 0$, which represents the cost of the next transition. It can be interpreted as the expected loss of replacing $s_t = i$ by a new state $s_{t+1} = j$ given by the probability distribution p_{ij}, $j = 0,1,2,...$

Consider the maximum expected net return given an initial state i, and name it f_i. From that position, a player is is allowed a transition to a different node $i \rightarrow j$, which leads a net expectation $-\Delta h_i + \sum_j p_{ij} f_j$.

The decision whether the player should move to the other node j depends on whether the return r_i for the current location surpass this value or not, that is:

$$f_i = \max \left\{ r_i, -\Delta h_i + \sum_j p_{ij} f_j \right\}.$$

The optimal return time for a state can be determined by the method: assume that the return time is 1, and increase if $f_i > r_i$.

Remember that T is the return time assigned by the mapping $\sigma : x^{k+1} \rightarrow T$ for a given initial state. Take T to be a random variable associated with the Markov chain $\{s_0, s_1, s_2, \ldots\}$. Let A be a subset of the state space and suppose that the initial state of the Markov chain is $s_0 = i$. The corresponding return time T is defined as the first time that the process reaches a state in A:

$$T = \inf \left\{ t \geq 0 \mid s_t \in A \right\}.$$

If $s_t \notin A$ for all $t \geq 0$, then $T = \infty$, which defines what we called a trap for the player.

Consider the expected net return, starting at state i and using a return time of T. With the cost for each transition, there is the conditional expectation

$$E \left\{ -\Delta h_{s_0} - \Delta h_{s_1} - \cdots - \Delta h_{s_{T-1}} + r_{s_T} \mid s_0 = i \right\}.$$

The maximum expected net return f_i is

$$f_i = \sup_{T \geq 0} E \left\{ -\Delta h_{s_0} - \Delta h_{s_1} - \cdots - \Delta h_{s_{T-1}} + r_{s_T} \mid s_0 = i \right\}.$$

The problem as formulated is far too computationally expensive to be practical; we can however simplify it while maintaining its optimality. Consider that a player is allowed at most n further steps before stopping. This restricts the choice of strategy lengths so that $0 \leq T \leq n$, and suggests a dynamic programming approach [57], which is one of the most fundamental and widely used techniques in

Artificial Intelligence. The essence of dynamic programming is Richard Bellman's well-known Principle of Optimality [55]. The principle is intuitive:

"An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

If we can frame the problem in such a way that this condition always holds, the optimal solution can be found by simple recursion. For each $i \geq 0$ and any non-negative integer n, call $f_i(n)$ the maximum expected net return given the initial state i and allowing at most n transitions. Clearly,

$$f_i(0) = r_i,$$

and, for $n \geq 1$, the recursion obeys the principle of optimality

$$f_i(n) = \max \left\{ r_i, - \Delta h_i + \sum_j p_{ij} f_j(n-1) \right\}$$

for each $i \geq 0$. That is, we can use the time limit n to decompose the function $\{v_i, i \geq 0\}$ into a monotone sequence of approximations. The maximum expectation $f_i(n)$ is non-decreasing in n, because the class of admissible return times grows when n is replaced by $n+1$. We have

$$f_i(n+1) \geq f_i(n),$$

and intuitively, it is clear that $f_i(n)$ converges to f_i as $n \to \infty$, if the limit is finite.

It is then straightforward to define attracting sets determined by each of the approximations $\{f_i(n), i \geq 0\}$. The limit function $\{f_i(n), i \geq 0\}$ satisfies the optimality equation, allowing us to define a decision region

$$A = \{i \geq 0 : f_i = r_i\}.$$

The set A form the optimal attracting regions for the problem.

For each $n \geq 1$, let

$$A(n) = \{i \geq 0 : f_i(n) = r_i\},$$

leadind to a decreasing sequence of regions: $A(1) \supset A(2) \supset \ldots$. And because $f_i(n) \leq f_i$ it follows that $A(n) \supset A$. Backward induction then shows that $f_i(n)$ is attained by the following return time:

$$T_i(n) = min\{t \geq 0: s_t \in A(n-t) \mid s_o = i\},$$

which leads to the iterative method from the last chapter, based on the attracting set $A(1)$. Recall that

$$A(1) = \{i \geq 0 : f_i(1) = r_i\},$$

where $f_i(1)$ is the maximum expected return, given the initial state i, when one transition is allowed:

$$f_i(1) = \max\{r_i, -\Delta h_i + \sum_j p_{ij} r_j\}.$$

The procedure is to stop as soon as the underlying Markov chain reaches a state in $A(1)$. It is a sub-optimal procedure, but in our setting it approaches the optimal policy. This occurs if the attracting set $A(1)$ is closed, in the sense that transitions from $A(1)$ to states outside it are not possible, which is true by construction.

4.3 Conclusion

To learn the network, the basic procedure is to construct the attractor for each of n different players at positions chosen by the different players $\{v_1, \ldots, v_n\}$ with a neighborhood of size $\mathcal{N}$ (a parameter). The training data is denoted Ω, and to each player correspond a subset of the data Ω_π, $0 < \pi \leq n$, such that $\Omega = \Omega_I \cup \Omega_{II} \cup \ldots \cup \Omega_n$. As noted, it is computationally useful to first select the configurations that are actually observed in the data before partitioning the different labels. This step involves an unproblematic parameter p_{min} which is a probabilistic threshold. A heuristic to select the initial positions for a player π is to choose the position with largest average KL divergence (relative entropy) between the player π and others local distributions X^0, as estimated by a histogram over the training data.

The conditional distribution over graph labels is the empirical one determined by the training data. We now assume a training set Ω of data points with known labels $X(v, t, \omega) \in X$, $\omega \in \Omega$. Given $D \subset \Omega$ and $\#D$ is the number of elements in D, the conditional distribution is

$$P(X(v,t+1) = x_2 \mid X(u,t) = x_1) = \frac{\#\{\omega \in \Omega \mid X(v,t+1,\omega) = x_1, X(u,t,\omega) = x_2\}}{\#\{\omega \in \Omega \mid X(v,t+1,\omega) = x_1\}}.$$

In conclusion, the overall method is (see *Figure 8* for pseudo-code): choose a node v and for each local game outcome, partition its entire neighborhood into equivalence classes of prediction on the node's local labels, $X(v,0)$. Each equivalence class is associated with a different conditional probability and a mutual information level. Different classes (and the labels wherein) then forecast different local configurations, which in turn have different utilities to the player, requiring different responses. If a class makes a favorable and certain prediction, we take it to belong to v's neighborhood S_1. Such labels have strategies with size $\sigma = 1$ (requiring just one action/edge for repair). The labels that do not belong to v's immediate neighborhood are partitioned further, into equivalence classes of prediction on S_1. They may belong to v's neighborhood's neighborhood or so on, and have a more indirect influence on v. The strategies sizes grow accordingly. With this process, the Player focuses its resources on the riskiest distributions, and (given the assumption of correct partition-sets) asymptotically approaches the optimal strategy-set.

Parameters: k, $\{\aleph^1, \aleph^2, \ldots, \aleph^k\}$, p_{min}, h_{min}, c_{min}, m, w, h

- Subsample image grid to size w × h
- Estimate marginal distribution P(Π | x) for each x ∈ X^0 and each v ∈ V using a histogram.
- For each player π
  - Select $v_\pi = \underset{v}{\text{argmax}} \sum_{\pi' \in \Pi} D(p(x \mid \pi) \,||\, p(x \mid \pi'))$
    - For each player π′ if H(π′|x) > h_{min}
    - Let S_1 be the set of all labels in X^{k+1} truncated by 1 where x ∈ S_1 if P(x) > p_{min}
      - For j = 1,…, $\aleph^k$
        - t=0
        - For each $x^{k+1} \in S_j = \{s_1, \ldots, s_m\}$
          - Estimate distribution $P(S^k|x,\pi)$ using Parzen window method.
          - (1) $S_j^{(t+1)(k+1)}(x) = \underset{s^{(t+1)(k+1)}}{\text{argmin}} D(p(S^k \mid x, \pi) \,||\, p(S^k \mid s^{(t+1)(k+1)}, \pi))$
          - t = t + 1
          - if $D(p(S^k \mid x) \,||\, p(S^k \mid s^{(t+1)(k+1)}, \pi)) - D(p(S^k \mid x) \,||\, p(S^k \mid s^{(t)(k+1)}, \pi))$ > cmin, goto (1)
        - For each i, $s_i \in S_j$
          - Calculate $u_i(1)$
          - If $u_i(1) > r_i$ add s_i to S_{j+1}

Figure 8: Overall procedure.

Notice that, for each player, we re-organize its sample space into a set of new variables S_1^{k+1}, S_2^{k+1}, ..., S_n^{k+1} taking values in different labels at different positions throughout the graph, with the obvious normalizations.

5. Flow

In the previous chapter, we have considered a method capable of learning the representation of an attractor structure. The representation captures the notion of a strategy-set of a game played in the graph. It consists of a set of tests a player can run in different locations throughout the graph. Now the player has to decide which tests to run on which nodes. One approach would be the player run every test on every node. Another would be a dynamic policy, in that case at each time tick the player could decide which tests to perform on which node, based on the result of previous tests and the information extracted on the last chapter about the costs, equivalence relationships and effectiveness of the tests. This chapter explores this idea: how to dynamically decide which tests to run on which position in the graph to construct the best classifier for the overall state of the graph. Recent relevant references on active and semi-supervised learning are [50], [51], [52] and [53].

5.1 Graph Flow

For illustration, let's consider a random walk in a graph with nodes s_0, s_1, s_2, s_3 and transition matrix given by

$$P = \begin{pmatrix} 0.3 & 0.7 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0.6 & 0.4 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix}.$$

The walk will stop with high probability when it reaches either state s_1 or s_3. The states s_1 and s_4 are commonly called absorbing states. These are states that, once entered, cannot be left. When the chain is started in a non-absorbing state, it will ultimately end up in an absorbing state [64]. In this chapter, we will take the sets in the players' local partition to be absorbing states and the neighbor's partition as non-absorbing states.

Remember that S^k is the set of local states and S^{k+1} the set of neighbor states. For neighbor state s_i^{k+1} and local state s_j^k, write R_{ij} for the probability that the chain starting in s_i^{k+1} will end up in state s_j^k, and R the matrix with entries R_{ij}. The matrix dimensions are $m \times n$, where m is the number of local states and n neighbor states. For example, the entries $B_{i,j}$ will give the probability that a walk initiated at s_i^{k+1} will

reach the label set $s_i^{\,k}$ before reaching any of the player's other states. It is straightforward to obtain the matrix R by Markov chain algorithms, and thus to simulate the dynamical system on the graph. See [63], [65] and [66] for a more detailed and general description. To that end, suppose that P is an absorbing Markov chain, and organize it so that the local states come first and the neighbor states come last – that is, in the canonical form:

$$P = \begin{pmatrix} I & \mathbf{0} \\ K & Q \end{pmatrix},$$

where I is a $m \times m$ identity matrix; $\mathbf{0}$ is a matrix of dimension $m \times n$ with all entries 0. And consider the matrix $N = (I - Q)^{-1}$ and the column vector $\mathbf{1}$ with all entries 1. Then the vector $t = N\mathbf{1}$ provides the expected number of steps before absorption for each starting state. The absorption probabilities R are calculated from N with the equation

$$R = (I - Q)^{-1}K.$$

Suppose also that for a Player π, $\boldsymbol{r}_\pi$ is a random function $\boldsymbol{r}_\pi$: $S^{k+1} \rightarrow s^k{}_\pi$ with domain the state space of a Markov chain P such that for i in S^{k+1},

$$\boldsymbol{r}_\pi(i) = \sum_j P_{ij}\boldsymbol{r}_\pi(j),$$

which defines a binary value $\boldsymbol{r} \in [0,1]$ for the player's individual states. The value $\boldsymbol{r} = 1$ means that the node belongs to Player π, or predict it wins. Order the vector $\boldsymbol{r}_\pi$ in the form

$$\boldsymbol{r}_\pi = \begin{pmatrix} \boldsymbol{r}_\pi^{\,o} \\ \boldsymbol{r}_\pi^{\,u} \end{pmatrix},$$

where $\boldsymbol{r}_\pi^{\,o}$ stands for the observed or sampled values $\boldsymbol{r}_\pi$ of nodes on the current level and $\boldsymbol{r}_\pi^{\,u}$ unobserved values. Since $R = NK$,

$$\boldsymbol{r}_\pi^{\,u} = (I - Q)^{-1}K\,\boldsymbol{r}_\pi^{\,o}.$$

We start with only the local value at the node v_0 observed in the graph; that is, no tests in the neighborhood performed. We then choose the tests to be performed by minimizing the risk of the prediction using the graph, which can be computed with matrix methods. The Bayesian risk R of the state $s_t \in S_t$ node u can be estimated with

$$R(s_t) = \sum_{t=1}^{n} H(s_t)\, p(s_t | s_{t-1}, \ldots, s_1) \simeq \sum_{t=1}^{n} H(s_t)\, \boldsymbol{r}.$$

After we query an unobserved state s_t the return function $\boldsymbol{r}$ will be altered. The new function is denoted r^{+s}. The active learning criterion we will use is that of myopically choosing the next query s_t that minimizes the expected risk:

$$s_t = \operatorname*{argmin}_{s'} R(s').$$

To this end, we need to re-compute the function r^{+s} after adding a state to the observed set.

We have seen that an individual player associate with each state s_π^{k+1} a conditional distribution $p(X^k \mid s_\pi^{k+1})$, or simply p_π. In that respect, to say that a player has chosen a node is equivalent to saying that the player has chosen a distribution, p_π. Let $x_1, x_2, \ldots, x_c$ be random samples of independent observations on X^{k-1}, and define the value of observed state with index i for $k > 0$ as

$$r_\pi^o(i) = \begin{cases} 1, \; if \; \prod_{j=1}^{c} \frac{p_\pi(x_j)}{p_{\pi'}(x_j)} > \rho, \vee \pi' \neq \pi \\ 0, \; otherwise \end{cases},$$

which is a "winning criteria" calculated after the players have selected their individual tests. In this case the criteria is derived from the likelihood ratio among the player's local distributions (where ρ is a threshold). It is intuitively clear that large ratios support the hypothesis associated with Player π, the hypothesis that p_π is the true density of observations. Notice then that $r_\pi^o \in \{0,1\}$ is the player's observed value for the random function $\boldsymbol{S}: X^k \rightarrow \Pi = \pi$ for the observed nodes, while $r_\pi^u \in [0,1]$ is its estimated value of the same function for the unobserved nodes.

In conclusion, r_π is a binary random field associated with a given player π and its binary function $P(S_{t-1} \in \pi)$. The field allows us to efficiently estimate the expected generalization error after testing a node. Once the nodes are selected, a classifier can be designed using both the observed and still unobserved

information. We can understand the return probability *r(i)* in terms of random walks on graphs: starting from an unobserved state *i*, a player moves to a node *j* with probability p_{ij} after one time tick. And the walk continues until the player reaches an observed node. Then *r(i)* is the probability that the player, starting from a node *i*, reaches an observed node with value 1 (that is, a position where $S_{t-1} \in \pi$) - remember that we have taken the observed nodes as the absorbing boundary for the random walk on the graph. We can then think of the observations at time *t* as opening either "pathways" or "blocks" to outcomes in *t-1* – that is, walking to a given observed node means walking directly to a favorable (or not) outcome in the level below.

Parameters: k, μ, ρ, c, m, u

- For each player π
  - For j = 1,…, k-1
    - Select node s_j randomly
    - Add s_j to observed set
- For each player π
  - For j = 1,…, k-1
    - Calculate r_π° for s_j

- Let μ = 1.0
- While μ < u
  - Let winner() = -1 be a k-length vector
  - For each player π
    - For j = 1,…, k-1
      - If winner(j) ≠ π
        - Select $s_j = \underset{s'}{\operatorname{argmin}} R(s')$
        - Add s_j to observed set
  - For each player π
    - For j = 1,…, k-1
      - If winner(j) ≠ π
        - Calculate r_π° for s_j
          - If $r_\pi^\circ = 1$
            - winner(j) = π

  - For each player π
    - For t = 1,…, k-1
      - Select $s_t = \underset{s' \in s_j}{\operatorname{argmax}} P(s' \mid r_\pi)$
  - Build NB-classifier from each player's $s_1, \ldots, s_{k-1}$
  - Run classifier and calculate μ

Figure 9: Sampling procedure.

While we have developed the machinery for selective sampling, we still need to devise a decision function for classification. The selected node s_t, the corresponding distribution and the set of labels $x \in s_t$ compose a feature vector for classification with first-order dependencies on s_{t-1} and s_{t+1} (in case $t\text{-}1 \geq 0$ and $t < k$). A Naïve Bayes (NB) classifier is a causal net with a structure that assumes that feature values are conditionally independent given the class label. Our classification process takes a sequence of samples from the different players $\{x_{u,\pi}\}$, where sample $x_{u,\pi}$ requests the label of node u, $X(u)$, from a instance associated with player π. Given the feature vector $\mathbf{x} = \{x_{u,\pi}\}$, Bayes' rule can be used to compute the

winner $argmax_{\pi} P(\boldsymbol{S} = \pi \mid \boldsymbol{X} = \boldsymbol{x})$, which is returned as the most likely classification given **x** and the players' models (due to the independence assumption, it does not matter from which player the observed value comes). Although independence is generally a poor assumption, in practice the NB often competes well with very sophisticated classifiers. The number of samples requested from each player in one iteration is $\sum_{i=1}^{k} |s_i|$, where $|s|$ is the number of labels in the chosen cluster s. It is a simple matter to assemble a Naïve Bayes classifier, see [61] or [5]. The accuracy of the classification can be estimated with the gini index, μ, over the winners.

Let's take a final stock. Suppose that n nodes, $x^n = x(v_1,1),\ldots,x(v_m,n)$, are tested sequentially in a graph. After each test x_{t-1}, $t = 1,\ldots,n$, a player is asked how likely each $x(u) \in X_t$ is to be the result of the next test. When the actual test is performed, the player π suffers a loss, which we have taken to be $log\ p(x_t \in S_{\pi} | x^{t-1})$. In order to achieve a minimal expected loss the player should predict with the conditional distribution $p(x_t \in S_{\pi} | x^{t-1})$. We have defined a set equivalence on labels for this process, where each class S_t corresponds to a distinct distribution for states for time $t-1$ (a mixed strategy to be played at that time). We have also seen that to each equivalence class $s(x^{t-1})$ correspond a distribution $p(X^{t-1})$, which in turn allowed us to identify the worst-case prediction classes under log-loss over the sequential game. Given the player's choice to improve its strategy on different positions in the graph, the player is best of improving the strategies starting in these classes. For labels in these classes, we optimized the best amount of time to let the stochastic process starting at that position to run, leading to a function σ over labels we called the return time function. In our solution formulation, this corresponds to the assumption that opponents will choose high-risk positions, $max_s\ R(s)$. Earlier in this chapter, we have devised the other side of this assumption, which express the player's contrasting desire to minimize the risk, and thus $\min_s \max_s R(s)$.

To get to this point, we have made assumptions. We have assumed that strategies are memoryless, thus not extending more than one step in the game tree. We have broken down, so to speak, the game into n subgames – which is reflected on the solution concept, a subgame perfect equilibrium solution. A subgame perfect equilibrium of an extensive-form game is a strategy whose restriction to each subgame is a Nash equilibrium of the subgame. We have only drafted this development, and much remains to be done. We have taken the game to be information asymmetric. And we have taken the losses to be expected losses (that is, calculated over the training set as opposed to run-time losses). We have assumed that our clustering algorithm converges to the optimal set of states, that the random fields r_{π} approximates the return probabilities after observations and that the marginal classification decisions combine into a

joint decision function. Yet, we claim to have addressed and brought attention to what we consider is the core aspect of sequential games, which is their interactive nature. In this respect, the players not only look ahead and plan what they would do in various contingencies, but also, as situations progress, constantly reassess their individual plays in response to new, actively obtained information.

5.2 Results

If we have a training set consisting of binary images of m different objects (or a background), we can make each correspond to a different player in a classification game. To recognize a given object, the NB classifier can be tested against different positions on the image grid. The following results are for a two-level network, with 'point' and 'shape' graphs. They were learned with 4 and 16 pixels neighborhoods, respectively. The point graph is first evaluated; then, for locations and players where the accuracy of the NB classifier $P(S^0 \in \pi \mid X(v_1, 1) = x, X(v_2, 1) = x, ..., X(v_j, t) = x)$ falls above the *0.96* threshold, we run the shape graph. And the maximum a posteriori winner at the shape level identifies the object.

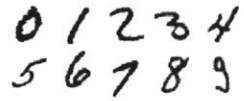
We have tested this procedure against two well-known and publicly available datasets. Both address the problem of visual recognition, the first of objects (COIL-20) and the second of handwritten characters (MNIST). In both experiments, the values of some parameters need to be fixed. Notably, accuracy is very much influenced by the number of clusters used, l. The optimal size is problem-dependent. In our experiments, this parameter has been tuned manually. Manually tuning parameters is a common practice; nonetheless, cross-validation could be used to determine its value automatically. All images were preprocessed with a Canny edge detector and a distance transform operator [68]. In addition to the different objects, in both cases we included a null-hypothesis player (black background).

The MNIST database [70] is a widely used collection of images of handwritten digits, examples in *Figure 9(a),* and poses a different challenge to object recognition systems. It consists of a collection of 70,000 images. They have been centered and sized-normalized to a 28x28, 256 grey level images. The MNIST is a large collection of 2D shapes in ten distinct classes (digits, corresponding to different players) with often subtle similarities and differences. In the literature, the first 60000 images are generally used for learning. Evaluation on this dataset shows competitive results, we have obtained an error rate of 0.72%, whereas the reported error rates in [70] vary from 12% to 0.7% (the best being the boosted LeNet-3 system). A more recent system has reported a 0.4% error rate [71].

The Columbia Object Image Library [72] is a database of images of 20 objects (corresponding to the different players, with 72 images/poses per player), examples in *Figure 9(b)*. Each image has been normalized to 128x128 pixels and are in true color. For our experiments, we have resized the original images to 32x32 pixels. From the complete sequence of images, it is usual practice to take the odd ones as training images and the even as tests. Our system has achieved 0% error rate in the COIL-20 dataset, as have a few other systems. At the moment, however there doesn't seem to exist an alternative for 3D object recognition that is as popular.

For both problems, our results are very competitive with the best results achieved. The significance is high since we strictly followed public protocols and since we have not tuned the system to the specific datasets. We also value the non-incremental nature of these results (that this is a fairly new and untested approach and it should be better studied). A disadvantage of our algorithm is its learning and recognition computational inefficiency. Recognition models were learned from ½ to ¾ of a week in a personal computer. It can also be said that the approach is altogether more complex than simpler systems with comparable success (for example, [19]).

(a)



(b)

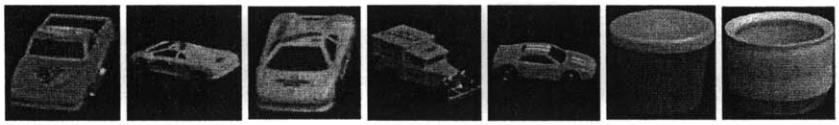


Figure 10: Datasets.

While for the COIL-20 dataset a few systems have reached zero error rates, for the MNIST dataset it has been argued that not much performance improvement will be possible (or will be meaningful). These results (on datasets with unocluded objects, homogeneous background and controlled illumination) can be misleading, since they do not encompass the true difficulty of the general object recognition problem. It seems that only recently the computer vision field has turned apt to address its central problem: general object classification in unconstrained environments.

Many researchers today use their own private data, which unfortunately prevent comparisons. We are forced to do the same: we captured a video sequence of a lab scene with a moving camera. The scene contains two tables in a cluttered office environment; on the first table there is only a mug, while on the other there is a (different) mug as well as several other random objects (bowl, keyboard, etc.). The justification for using a video clip (instead of setting-up separate still images) is based on the general observation that the output of many vision algorithms (specially ones based on classifiers) varies constantly with unapparent image changes (such as slight illumination or pose changes, camera movements and motion blur), making them not always suited for real world applications. The video starts with a frontal take of the two mugs (with the first table closer), and as it proceeds, the camera travels from the first table to the second, then the camera elevation varies from low (table level) to high on the second table (moving from a frontal to a top view of the objects), afterwards the camera returns to its original position and the mug is waved in front of the camera, *Figure 10*. The video was re-sampled at 6 fps with duration of 40 seconds, which we divide in 12 blocks of 20 frames.

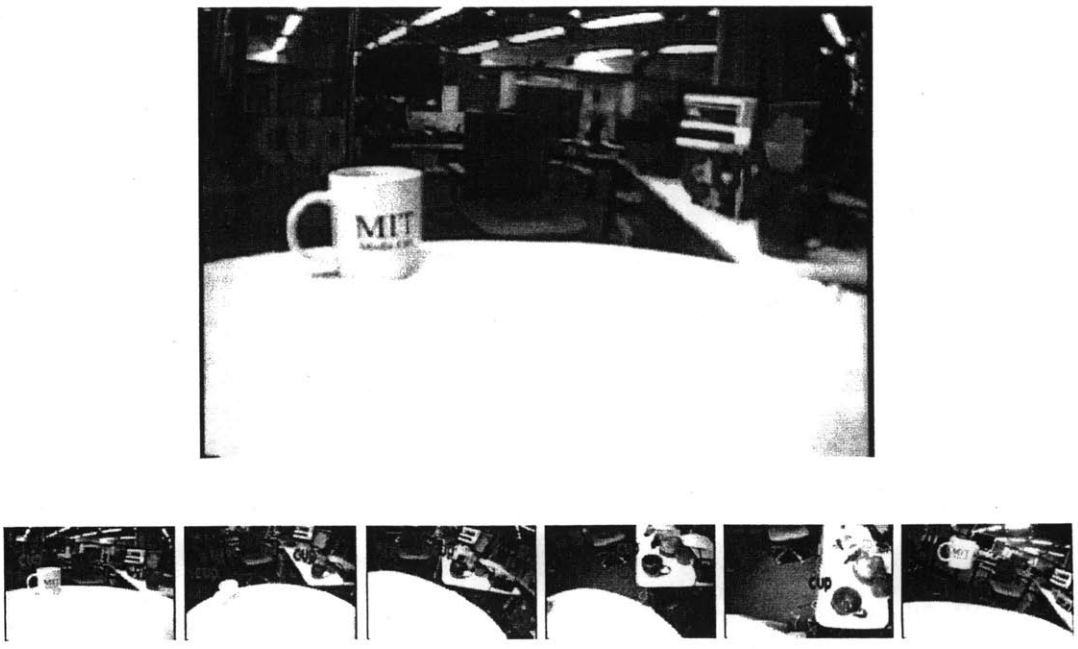


Figure 11: Experiment.

We trained the system with three players, corresponding to frontal-imaged mugs (70 frontal images at slight different angles of 4 different mugs), top-imaged mugs and generic background (null hypothesis, 110000 images of random scenes). In this way, we can compare the performance of the proposed system with a decision tree approach and demonstrate the graph dynamical system's ability to better generalize across object instances, even in varied backgrounds. We have built a CART tree [21] with Haar-like features using the Intel OpenCV library. To make the comparison more meaningful we change the first level label-set of the graph dynamical system from the binary set, $X^0=\{0,1\}$, to indices in the bank of Haar-like filters, $X^0=\{0,1,2,3,4,5,6,7,8,9\}$. *Figure 11(a)* illustrates the set of filters used, all computed from simple differences between the sums of pixels within two adjacent rectangular regions. They were originally proposed by Papageorgiou [73] and can be extracted quickly. Both algorithms will run classifiers (the NB or the CART) against different positions in the image and in *20* different scales.

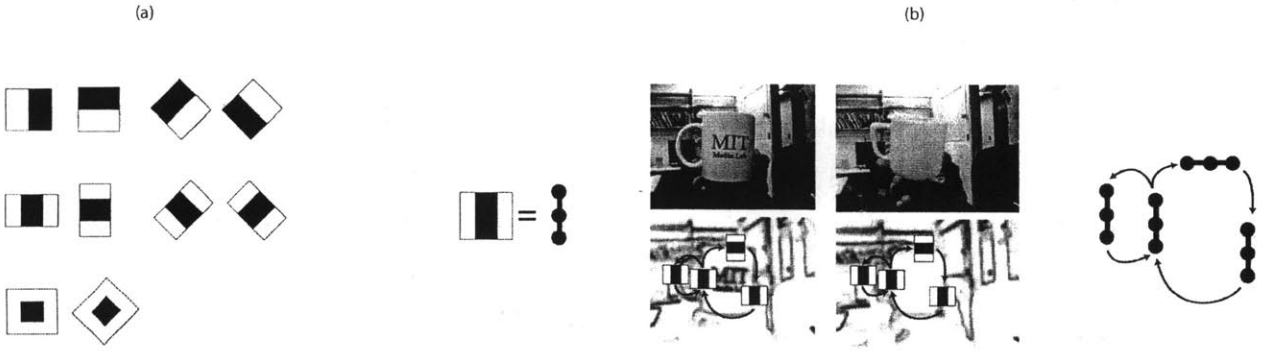


Figure 12: Features.

We have found the CART system to work best when different classifiers were assembled for individual object instances in the scene (thus in spite of using one classifier for all mugs, use an ensemble). We have also manually determined its best parameters for this scene to be a tree size of *14* and a window size of *21x21*. Each recognition trial operates on one video frame. *Figure 12* depicts the results over sequential blocks of 20 frames in the clip. True positive trials correspond to situations where the object was in the scene and was recognized in the right position, while false positives correspond to situations where the object was recognized but was not in the scene or was in a different position. We classified the trials in this way subjectively, by analyzing the video and the two systems' outputs.

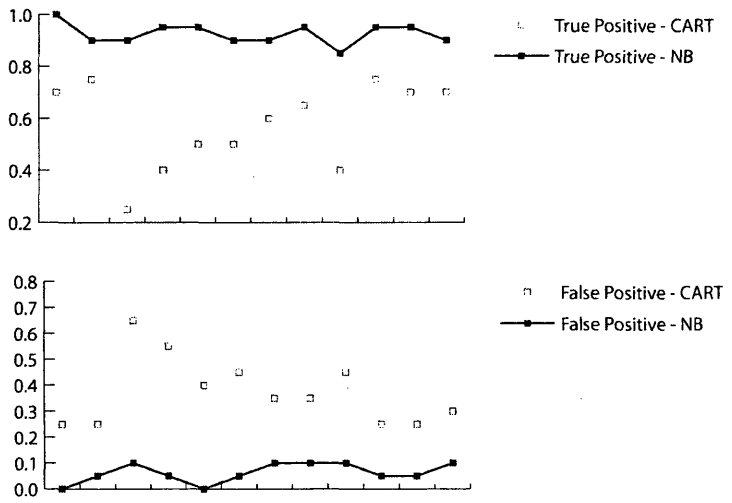


Figure 13: Result on cluttered scene.

For both systems, the worst error rates were generally obtained in blocks corresponding to the camera movement between tables (blocks 3 and 7-9) and thus corresponding to atypical views of the objects (non-front or non-top point of views). Together with the fact that only one NB classifier can recognize all mug instances, this observation shows that the graph dynamical system generalizes better the structure of the objects. *Figure 13* shows some of the frames misclassified by the NB system. We have observed that most misclassification were either due to overly poor conditions for low-level processing or occlusion of critical object parts. Although the filter bank used is recognizably low-quality, a state-of-the-art edge detector would not radically improve the results (this is illustrated on the bottom *Figure 13*). The graph dynamical system as proposed here works in a streamline bottom-up fashion (from features to objects, committing to all-or-none decisions on the way). We believe that developing mechanisms to integrate information in the opposite direction is the key to true improvement, for example by taking temporal (tracking), scene or semantic information into account.

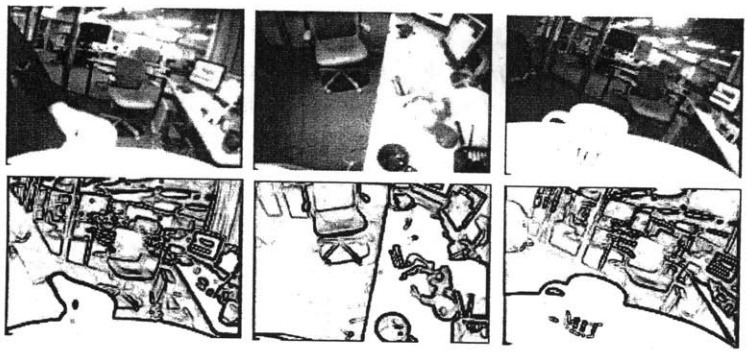


Figure 14: Misclassified frames.

6. Bibliography

[1] Newman, M. (2000). *Models of the Small World.* Journal of Statistical Physics, 101, 819-841.

[2] Wiener, N. (1948). *Cybernetics of Control and Communication in the Animal and the Machine.* Wiley.

[3] Hungerford, T. (1996). *Abstract Algebra: An Introduction.* Brooks Cole.

[4] Russell, S., Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

[5] Duda, D., hart, P., Stork D. (2000). *Pattern Classification, 2nd Edition.* Wiley.

[6] Smith, L., Jones, S. (1993). *Cognition Without Concepts.* Cognitive Development, 181-188.

[7] Schyns, G., Rodet L. (1997). *Categorization Creates Functional Features.* Journal of Experimental Psychology: Learning, Memory & Cognition.

[8] Tulving, E.,Thomson D. (1973). *Encoding Specificity and Retrieval Processes in Episodic Memory.* Psychological Review, 80, 352-373.

[9] Clark, H. (1983). *Making Sense of Nonce Sense.* The Process of Language Understanding. 297-331. Wiley.

[10] Goldstone, R. , Lippa, Y., Shiffrin, R. (2001). *Altering Object Representations Through Category Learning.* Cognition, 78, 27-43.

[11] Sanocki, T. (1992). *Effects of Font and Letter Specific Experience on the Perceptual Processing of Letters.* American Journal of Psychology, 105, 435-458.

[12] Coltheart, V. (1999). *Fleeting Memories.* MIT Press.

[13] Fu, K. (1983). *A Step Towards Unification of Syntatic and Statistical Pattern Recognition.* IEEE Trans. PAMI, 5, 2, 200–205.

[14] Bunke, H. (2001). *Recent Advances in Structural Pattern Recognition with Application to Visual Form Analysis.* IWVF4, LNCS, 2059,11–23.

[15] Kosinov, S., Caelli T. (2002) *Inexact Multisubgraph Matching Using Graph Eigenspace and Clustering Models.* SSPR & SPR, LNCS, 2396, 133–142.

[16] Gold S., Rangarajan, A. (1996). *A Graduated Assignment Algorithm for Graph Matching.* IEEE Trans. PAMI, 18(4), 377–388.

[17] Hancock E., Wilson, R. (2002). *Graph-based Methods for Vision: A Yorkist Manifesto.* SSPR & SPR, LNCS, 2396, 31–46.

[18] Luo, B., Hancock, E. (2001). *Structural graph matching using the EM algorithm and singular value decomposition.* IEEE Trans. PAMI, 23,10,1120–1136.

[19] Belongie, S., Malik J., Puzicha J. (2002). *Shape Matching and Object Recognition Using Shape Contexts*, IEEE Trans. PAMI.

[20] Breiman, L., Friedman, J. (1984). *Classification and Regression Trees.* Chapman & Hall.

[21] Amit, Y., Geman, D., Wilder K. (1997). *Joint induction of shape features and tree classifiers.* IJCAI.

[22] Steffen, L. (1996). *Graphical Models.* Oxford University Press.

[23] Robinson, R. (2004). *An Introduction to Dynamical Systems.* Prentice Hall.

[24] Heckerman, D. (1998). *A tutorial on learning with Bayesian networks.* In M. I. Jordan (Ed.), Learning in Graphical Models. Kluwer.

[25] Buntine, W. (1996). *A guide to the literature on learning probabilistic networks from data.* IEEE Transactions on Knowledge and Data Engineering 8, 195–210.

[26] Oliver, M., Smith, M. (1990). *Influence Diagrams, Belief Nets and Decision Analysis.* Wiley.

[27] Tong, S., Koller D. (2001). *Active Learning for Structure in Bayesian Networks.* International Joint Conference on Artificial Intelligence.

[28] Horvitz, E., Breese, J., Henrion M. (1988). *Decision Theory in Expert Systems and Artificial Intelligence*. Journal of Approximate Reasoning, Special Issue on Uncertainty in Artificial Intelligence, 2, 247-302.
[29] Koller, D., Pfeffer, A. (1997). *Representations and Solutions to Game-theoretic Problems*. Artificial Intelligence 94, 167-215.
[30] Holcombe, W. (1982). *Algebraic Automata Theory*. Cambridge University Press.
[31] Béal, M., Perrin, D. (1997). *Symbolic dynamics and Finite Automata*. In Handbook of formal languages, Vol. 2, 463-505. Springer, Berlin.
[32] Berstel, J., Perrin, D. (1984). *Theory of Codes*. Pure and Applied Mathematics. Academic Press.
[33] Morton, D. (1970). *Game theory*. Basic Books.
[34] Grädel, E., Thomas, W., Wilke, T. (2002) *Automata, Logics, and Infinite Games*. Lecture Notes in Computer Science, 2500, Springer-Verlag.
[35] Arnold, A., Vincent, A., Walukiewicz I. (2003). *Games for Synthesis of Controllers with Partial Observation*. Theoretical Computer Science, 303, 1, 28, 7-34.
[36] Gurevich, Y., Harrington, L. (1982). *Trees, Automata and Games*. Proc. 14th ACM Symp. on Theory of Computation, 60–65.
[37] Cassandras, C., Lafortune, S. (1999) *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.
[38] Haussler, D. (1997). *A General Minimax Result for Relative Entropy*. IEEE Trans. Information Theory, 43, 1276-1280.
[39] Kazakos, D. (1983). *Robust Noiseless Source Coding Through a Game Theoretic Approach*. IEEE Trans. Information Theory, 29, 577-583.
[40] Kapur, J. (1989). *Maximum Entropy Models in Science and Engineering*. Wiley.
[41] Cover, T., Thomas, J. (1991). *Elements of Information Theory*. Wiley-Interscience.
[42] Tishby N., Pereira F. and Bialek W. (1999). *The Information Bottleneck Method*. Proc. of the 37-th Annual Allerton Conference on Communication, Control and Computing, 368-377.
[43] Globerson, A., Tishby, N. (2003). *Sufficient Dimensionality Reduction*. Journal of Machine Learning Research, 3:1307-1331.
[44] Newman, M. (2003). *The Structure and Function of Complex Networks*. SIAM Review, 45, 167-256.
[45] Shalizi, C., Crutchfield, J. (2002). *Information Bottlenecks, Causal States, and Statistical Relevance Bases*. Advances in Complex Systems, 5, 1-5.
[46] Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning*. Springer.
[47] Kikinis, R., Shenton, M., Jolesz, F., Gerig, G., Martin, J., Anderson, M., Metcalf, D., Guttmann, C., McCarley, R. Lorensen, W., Cline, H. (1992). *Quantitative Analysis of Brain and Cerebrospinal Fluid Spaces with MR imaging*. J. Magn. Res. Imag., 619-629.
[48] Chalones, K., Verdinelli, I. (1995). *Bayesian Experimental Design: A Review*. Statistical Science, 10, 237-304.
[49] Burkhardt, H., Siggelkow, S. (2001). *Invariant Features in Pattern Recognition – Fundamentals and Applications*. Nonlinear Model-based Image/Video Processing and Analysis, 269-307, Wiley.
[50] Cohn, D., Ghahramani, Z., Jordan, M. (1996). *Active Learning with Statistical Models*. Journal of Artificial Intelligence Research, 4, 129-145.
[51] Lindebaum M., Markovitch S., Rusakov D. (2004). *Selective Sampling for Nearest Neighbor Classifiers*. Machine Learning, 14.
[52] Zhu, X., Lafferty, J. (2004). *Harmonic mixtures: Combining Mixture Models and Graph-based Methods for Inductive and Scalable Semi-supervised Learning*. Machine Learning: Proceedings of the Twenty-Second International Conference.
[53] Blur, A., Langley, P. (1997) *Selection of Relevant Features and Examples in Machine Learning*. Artificial Intelligence, 97, 245-271.

[54] Lawler, G. (1995). *Introduction to Stochastic Processes*. Chapman & Hall.
[55] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.
[56] Whittle, P. (1982). *Optimization over Time: Dynamic Programming and Stochastic Control, Vol. 1*. Wiley.
[57] Ross, S. (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press.
[58] Rosenthal, A. (1980). *Dynamic Programming is Optimal for Certain Sequential Decision Processes*. Journal of Mathematical Analysis and Applications, 73, 134-137.
[59] Wesley, S. (1984). *Scientific Explanation and the Causal Structure of the World.* Princeton University Press.
[60] Friedman, N., Geiger, D., Goldsmith, M. (1997). *Bayesian Network Classifiers*. Machine learning, 29, 131-163.
[61] Greiner, R., Grove, A., Roth, D. (2002). *Learning Cost-sensitive Active Classifier*. Artificial Intelligence, 139.
[62] Blum, A., Chawla, S. (2001) *Learning from Labeled and Unlabeled Data using Graph Mincuts*. ICML.
[63] Nash-Williams, C. (1959). *Random Walk and Electric Currents in Networks.* Proceedings Cambridge Phil. Soc., 55, 181–194.
[64] Isaacson, D., Madsen, R. (1976). *Markov Chains: Theory and Applications*. Wiley.
[65] Balabanian, R., McPeek, J., *Electrical Network Theory*. (1969). Robert E. Krieger Publishing Company.
[66] Ahuja, R., Magnanti, T., Orlin, J. (1993). *Network Flows - Theory, Algorithms, and Applications.* Prentice Hall.
[67] Dhillon, I., Mallela, S., Kumar, R. (2003). *A Divisive Information-theoretic Feature Clustering Algorithm for Text Classification.* Journal of Machine Learning Research, 3, 1265–1287.
[68] Rosin, P., West, G. (1995). *Salience Distance Transforms*. Graphical Models and Image Processing, 57, 6, 483-521.
[69] Lowe, D. (2004). *Distinctive Image Features from Scale-invariant Keypoints*. International Journal of Computer Vision, 60, 2, 91.
[70] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86, 11, 2278-2324.
[71] Simard, P., Malvar, H. (2004). *An Efficient Binary Image Activity Detector Based on Connected Components*. International Conference on Acoustic, Speech and Signal Processing (ICASSP), Montreal.
[72] Nene, S., Nayar, S., Murase, H. (1996). *Columbia Object Image Library (COIL-20)*. Technical Report CUCS-005-96.
[73] Papageorgiou C., Oren M., Poggio T. (1998). *A General Framework for Object Detection.* International Conference on Computer Vision.