Self-Organizing Event Maps

by

Seth Ronald Tardiff

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author .....
Department of Electrical Engineering and Computer Science
May 7, 2004

Certified by .....
Patrick Henry Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Self-Organizing Event Maps

by

Seth Ronald Tardiff

Submitted to the Department of Electrical Engineering and Computer Science
on May 7, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

To take further steps along the path toward true artificial intelligence, systems must be built that are capable of learning about the world around them through observation and explanation. These systems should be flexible and robust in the style of the human brain and little precompiled knowledge should be given initially.

As a step toward achieving this lofty goal, this thesis presents the *self-organizing event map* (SOEM) architcture. The SOEM architecture seeks to provide a way in which computers can be taught, through simple observation of the world, about typical events in a way that is flexible and robust. The self-organizing event map, as a data structure, stores a plane of event models that are continually updated and organized according to events that are observed by the system. In this manner, the event map produces clusters of similar events and provides an implicit representation of the regularity within the event space to which the system has been exposed.

As part of this thesis, a test system that makes use of self-organizing event map architecture has been developed in conjunction with the Genesis Project at the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. This system receives input through a natural-language text interface and, through repreated training cycles, becomes capable of discerning between typical and exceptional events. Clusters of similar events develop within the map and these clusters act as an implicit form of the more commonly used (and explicit) notion of scripts and capability lists. For example, a trained map may recognize that dogs often run, but never fly. Therefore if a new input is received that describes a flying dog, the map would be capable of identifying the event as exceptional (or simply erroneous) and that further attention should be paid. By using clusters of similarity as an implicit representation, the self-organizing event maps presented here more accurately mimic natural memory systems and do not suffer from being tied to the limitations of a specific explicit representation of regularity.

Thesis Supervisor: Patrick Henry Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Acknowledgments

Most importantly, I owe my introduction to artificial intelligence and my optimism for the future of the field to my advisor, Patrick Winston. His unique style, depth of insight, and wealth of knowledge have brought me more opportunities than I could possibly have imagined.

My involvement in the field of artificial intelligence and the Genesis Group I owe to Jake Beal and Justin Schmidt. They have shown me how to tackle seemingly insurmountable problems and to enjoy the challenge of building AI systems, even during times of great difficulty.

I attribute a significant amount of inspiration to all members of the Genesis Group, past present, and future. I feel as though I have grown along with the group in the past few years and I am grateful for their understanding and motivation.

I also owe thanks to the countless teachers, TAs, fellow students, and friends who have been a part of my education. From preschool onward, I have been fortunate to be surrounded by dedicated and caring individuals, to whom I owe a great debt.

And of course, I owe the greatest debt to my parents and the rest of my family for giving me every opportunity that I could have ever wanted. They have given me the freedom and the ability to pursue my dreams and have made all of this possible.

Contents

List of Figures

Preface

The past fifty years of research in artificial intelligence has brought about tremendous advancement in our ability to build highly domain-specific "intelligent" systems. Beginning with the simple integration programs of Slagel and Moses and continuing through modern visual surveillance and planning programs, researchers have generally fared well in the building of systems that mimic specific parts of human behavior. I believe that for advancements in artificial intelligence to continue, a shift in thinking is required among the research community. We must turn our attention away from building highly specific and optimized algorithms and towards the glaringly open questions of how human perceptual systems interact, how the substructures of the brain operate in a massively parallel fashion, what role imagination and hallucination play in the problem solving process, and how we are able to instantly identify salient features of scenes and events.

In this opinion, I am thankfully not alone. The work presented in this thesis is intended to be a step along the path currently being pursued by researchers in both the computer and brain and cognitive science departments at MIT, most notably those in the Genesis Group. Although this work represents but a small piece of the overall goal of understanding and implementing a more thorough notion of human intelligence, I trust that it will play a role in future developments.

Chapter 1

Introduction

1.1 Overview

I assert that the next generation of breakthroughs in artificial intelligence research will center around the building of cognitively complete systems that learn to reason about the world in a human-like fashion. These systems will begin "life" having much potential, but little in the way of precompiled knowledge. Initially they will be relatively useless and cumbersome, operating in seemingly random ways and according to rules that are not apparent to any outside observer. Through exposure to the world and, most importantly, by becoming increasingly aware of the *regularity* in the world, these systems will begin to recognize patterns, to infer causations, and to function in a way that is far more like an adult than a child. Building such systems will certainly not happen anytime soon, but their advent is not out of the realm of possibility. This thesis is intended to be a step toward this goal.

This thesis presents the *self-organizing event map* (SOEM) architecture and an example implementation. The SOEM architecture has been inspired in large part by the work of Tuevo Kohonen on self-organizing maps (SOMs) (Kohonen, 2001) and by Stephen Larson in applying self-organizing maps to the grounding of symbols (Larson, 2003). A self-organizing map is essentially a data structure that organizes itself in response to repeated exposure to inputs. After sufficient training, a SOM facilitates the observation of clusters of regularity within the input space, as neighboring

elements of the map become increasingly similar to one another. The SOM structure has been used extensively in numerical analysis and in the identification of regularity within inherently numeric input spaces.

The SOEM architecture presented here extends the basic self-organizing map paradigm by allowing the organizing elements to be *events*, not simply pieces of numeric data. Within a self-organizing event map, each element is an event represented in the lexical conceptual semantics framework proposed by Ray Jackendoff in (Jackendoff, 1983). The input samples are thus also event representations. Through exposure to many cycles of training data, a self-organizing event map begins to recognize regularity in the event space by building clusters of similar events.

1.2 Motivation

A system built using self-organizing event maps has several important capabilities and properties, outlined below:

1. The SOEM architecture functions as an online learning algorithm and requires no outside supervision during the training process. Moreover, there is not a defined line between the *training process* and simple existence. The system is able to observe the world and continually organize itself in response to input events.

2. The architecture is independent of the input interface to which it is attached. The example system presented in this thesis makes use of a natural language parser that receives input in the form of English sentences. Feasibly, this input mechanism could be swapped out for another form of interface, such as a visual recognition system. So long as the input system is able to parse observations into discrete events, represented as lexical conceptual semantics frames, then the SOEM architecture will operate properly.

3. Once a self-organizing event map has been sufficiently trained, it can be used as a way of determining whether an event is exceptional or typical, given previ-

ous experiences. Many recognition and problem solving tasks involve choosing between multiple hypotheses on the basis of which one seems most likely. A trained event map is able to quickly identify which of two possible events is most likely to have occurred, in the context of previous experiences of the system.

4. A trained self-organizing event map can be used as a content-based storage system. An event map allows an input event to be quickly matched to some existing cluster of similar events and stored at a location within a cluster of similar events.

1.3 Implementation

The event map system described in this thesis has successfully implemented the SOEM architecture and is capable of observing the world through events described in natural language. The self-organizing event map at the heart of the system becomes increasingly clustered into events of high similarity and can be used to detect exceptional events. Event maps containing up to 2500 elements (events) have been generated and trained. The developing clusters are observed through the use of an element-average-distance metric that colors each cell of the map according to its similarity with its neighbors.

Assuming that a map has been trained through exposure to "regular" events (i.e. events that would normally occur in the world), the system can determine that one event, such as, "a *cow* walked through the field," is more likely to have occurred than another, such as, "a *car* walked through the field." When this example is presented to the implemented and trained system, the first event, involving the cow, is deemed significantly less exceptional than the second event involving the car. Had these two events been generated as possible parses of a perceived auditory input, the trained event map would easily have suggested the first parse as the most likely result.

The SOEM architecture is a powerful step toward the goal of developing systems capable of learning through observation. By augmenting and further implementing this architecture, along with developing other architectures in the same style, I hope

that future researchers will continue to take steps along this path.

1.4 Preview

The rest of this thesis provides the background and details of the self-organizing event map architecture and implementation.

In Chapter 2, the notion of an "event" is defined and a representation for events is described.

In Chapter 3, the motivation behind and details about various self-organizing data structures are presented. This chapter also motivates the idea of implicit representations and how they can be used to build powerful (and more human-like) artificial intelligence systems.

Chapter 4 provides information on various other background material not previously discussed. This other background includes descriptions of the BridgeSpeak parser, the concept of threads in memory, and the powerful notion of matching via intermediate features.

In Chapter 5, the design of the actual self-organizing event map system is presented in detail, including a discussion of various design decisions and tradeoffs. Further, the implementation is detailed, along with examples of the system in use. The capabilities and limitations of the current implementation are noted, and data supporting its effectiveness is provided.

Finally, Chapter 6 provides a discussion of how the existing implementation can be used and extended in the building of complete and robust AI systems. Various proposals for future work are presented in the hope that the work presented in this thesis will serve as the motivation for future developments.

Chapter 2

Events and Representations

Before any serious discussion of the event map system can commence, the term *event* must be defined. Further, methods for representing discretized events must be developed. This chapter discusses the notion of an event, with emphasis on the representation of events in a simple form that makes use of a highly-structured vocabulary.

2.1 The Event Notion

In the most simple form, the notion of an event can be summed up as, *the factual information regarding a specific period of time, involving actors, actions, and locations.* A more elaborate definition often includes the fact that each event involves a specific emotional state, some temporal information about the date and time, and context-specific information regarding sensory and perceptual attributes.

For the purpose of this thesis, the emotional and temporal aspects of events will be ignored. I will focus on the notion of an event as a collection of actors, actions, and locations. Each event will occur within a discrete amount of time, although this time period need be neither completely defined or especially short. For example, an event such as, "The earth revolved around the sun," takes place at an undefined time and for an extensive period.

This thesis takes a simple view of events because I believe it provides the first step towards building a more complete event system. The emotional aspects of an event,

for example, can be added at a later time through connections between the simple event descriptions and an emotional representation. The addition of further context is simply a matter of creating deeper associations between subsystems.

2.2 Event Representations

The general concept of representation is nicely defined by Winston in (Winston, 1993):

> "In general, a representation is a set of conventions about how to describe a class of things. A description makes use of the conventions of a representation to describe some particular thing."

Winston's definition also asserts that a representation has four key parts: lexical, structural, procedural, and semantic.

In the context of events as described above, a representation should therefore provide the vocabulary, structure, and methods for manipulating descriptions of actors, actions, and locations. Even within this relatively simple prescription, many options exist for how exactly to represent each component.

Some event representations, such as that proposed in (Borchardt, 1993), describe the relationships within an event by focusing on the transitions that take place. For example, in Borchardt's representation, a description of a ball falling would make explicit the fact that the distance between the ball and the floor is increasing along with the speed of the ball. Other event representations may focus on the relationships in time (Allen, 1983) and describe events using keywords related to the passage of time, such as "before," "after," and "while."

The representation used in this thesis is the *Lexical Conceptual Semantics*, described below.

2.2.1 Lexical Conceptual Semantics

A particularly nice event representation is proposed by Ray Jackendoff in (Jackendoff, 1983). In his proposal, Jackendoff argues that descriptions of events, even abstract

ones, can often be created in terms only of the *paths*, *objects*, and *places* that are involved. For example, the ball dropping event from above would be described by the ball going *from* the top of the table and traversing a trajectory *toward* the floor.

Paths, as defined by Jackendoff, can be summarized by making use of one or more of several key types: to, toward, from, away from, via, along, through, etc. Every general class of path that an object takes can be defined by one of these types.

Places are defined by two basic components: a reference object and an intransitive preposition. A reference object can be any "thing," such as "the car," or "an apple." An intransitive preposition is the modifier that describes how the place is related to the object. Examples of these modifiers would be, "under," "on," and "above." A full place definition is therefore a description of the form, "on the table," or "above the wall."

Jackendoff defines reference objects simply by their names and makes no attempt to incorporate more complete descriptions of objects. Section 4.1 describes how *threads* can be used to annotate the reference objects to produce a more complete and useful description.

Having defined the notions of places and paths, the LCS representation simply involves relating objects to paths and places. Expressed in a linearized, textual form, the formula for an event is,

```
[GO ([thing X], [path Y [place Z]])]
```

In this formulation, X can be any reference object, Y is a path descriptor such as, "to," and Z is a full place description, involving both an intransitive preposition and a reference object. A description of a ball falling to the floor is therefore,

```
[GO ([BALL], [TOWARD [ON FLOOR]])]
```

The representation also supports nesting and concatenation of paths and places so that more complex descriptions can be created. For example, a more complete description of the ball falling from the table to the floor is given by,

```
[GO ([BALL], [FROM [ON TABLE] [TOWARD [ON FLOOR]]])]
```

The complete representation proposed by Jackendoff also makes allowances for describing causation and states. These aspects of the representation are not relevant in the context of this thesis.

Chapter 3

Self-Organizing Models

Self-organizing computational data structures borrow concepts used in biological systems and allow robust, continually evolving systems to be developed. This chapter presents the biological evidence that supports the use of self-organizing structures and provides an overview of computational implementations of the self-organizing paradigm. The specific neural technique of self-organizing maps is introduced as a relevant foundation for the development of self-organizing event maps.

3.1 Biological Relevance

Although there still exists dissension over exactly how the brain processes and stores information, the idea of self-organization within the neural landscape is rather well accepted. Debates rage as to whether many functions of the brain are brought about by learning, evolution, genetics, or some other factor, but the precise answers to these questions are not relevant here.

Many studies have shown that neural connections and even spatial ordering of neural functions in the brain develop in response to regularity in the world to which the individual is exposed. For example, it has been shown that within the auditory cortex of the brain there is a map of neurons (known as the *tonotopic map*). The neurons that comprise this map are ordered spatially according to the pitch and frequency of perceived tones (Kohonen, 2001). This phenomenon is not specific to

the cortex, but can be found in other areas of the brain as well. For example, studies of rats' neural structures have shown that maps of their geographic environment develop within the hippocampus. In this case, a rat's position in a room or maze corresponds to a specific section of the hippocampul map (Kohonen, 2001).

Although the exact nature of the brain's processing is unclear, there is certainly biological evidence for a very rough and general description, such as the following:

When the sensory organs perceive input, these input values are passed into parts of the brain and "excite" certain neural cells. The excited cells, in turn, pass along their responses to their neighbors until some decay factor stops the propagation of the signals. Given a completely different input value, a different portion of the brain may become excited. Once exposed to many many instances of such inputs, clusters of regularity develop. The interesting aspect of this process is that it runs in parallel on many sections of the brain. For example, perception of a red car driving down a street may cause various clusters of neurons to become excited. One cluster may become excited because of the color of the car (a cluster corresponding to red visual input, perhaps). Another may become excited due to the loud jarring noise, and yet another because of a person walking past at the same time. The excitation of these multiple clusters at the same time causes some *cluster association* to take place. For example, if a certain individual is repeatedly exposed to this scene, he may learn (via cluster association) that a loud jarring noise is generally associated with the color red.

The above description is an admittedly ultra-simplified interpretation of neural processing and learning, but it is that simplicity that makes it a useful starting point for the development of computational models of the brain. The development of clusters within neural maps based on regularity in the input space has inspired many of the techniques that fall into the broad category of *neural networks*. A specific type of neural network that is very much based on the simple description of neural processing given above is the *self-organizing map*.

3.2 Self-Organizing Maps

The self-organizing map (SOM) is a computational data structure originally proposed by Tuevo Kohonen in 1982. The data structure and its associated algorithms seek to mimic, at least in behavior, the basic self-organizing nature of neurons as described in the previous section. The SOM seeks to identify regularity within an input space by organizing itself according to repeated training cycles.

In its most simple form, a SOM is a plane of cells arranged into some simple topology (often just a grid). Each cell represents a piece of data, usually a vector of numeric values. Input vectors of the same form are then presented to the SOM during a training phase. Each incoming input is used to slightly modify the map so that some portion of the plane of cells becomes "more like" the input vector. After repeated training cycles of this form, clusters of similarity develop and the plane of elements can be used to easily visualize (in a 2D form) the regularity within a multi-dimensional input space.

A common and easily-visualized example of a SOM is the color map shown in Figure 3-1. In this example, each cell of the map is a color defined by a 3-dimensional vector (red, green, and blue values). Initially, each cell is a random color. Through training, the map elements are refined and organized into clusters of similar colors.

Self-organizing maps are regarded as powerful statistical learning tools because they function unsupervised, allow for simple visualization, and can be used as a cluster development algorithm. Because of the flexibility of the general SOM paradigm, they have been used for a wide variety of applications, ranging from acoustic processing to stock market analysis. A good survey of implementations is given in (Oja and Kaski, 1999).

3.2.1 The SOM Algorithm

The basic self-organizing map algorithm proposed by Kohonen is straightforward. The process outlined below assumes that a map of some topology has been created and that each element of the map, along with each input, is an n-dimensional vector

Figure 3-1: Color Som Example. The map begins in a randomly initialized state in the left-most picture. Through exposure to input colors, the map becomes organized into a color gradient, shown in increasingly mature form in the two right-most pictures. (Images courtesy of Tom Germano – http://davis.wpi.edu/matt/courses/soms/)

of numeric values. In the most simple setup, each vector within the map is initialized to a random value. Each training cycle consists of the following steps:

1. Given an input vector, $x(t)$, find the cell of the map, $c(t)$, whose vector, $m_c(t)$, most closely matches the input. Often, simple Euclidean distance between the vectors is used as the distance function.

$$c(t) = argmin_i||dist(x(t), m_i(t))|| \tag{3.1}$$

2. Find the *neighborhood set*, $N_c(t)$ of cells that are within distance d of the winner cell $c(t)$.

3. For each neighbor cell i in $N_c(t)$, alter its vector to be "more like" the input vector, $x(t)$. This process makes use of the learning rate, $\alpha(t)$. This rate function determines how much the existing vector is altered in response to an input.

$$m_i(t+1) = m_i(t) + \alpha(t)[x(t) - m_i(t)] \tag{3.2}$$

Note that all cells that are not within the neighborhood of the winning (most closely matching) cell are not altered. The process described above repeats for each input. Often the learning rate function, $\alpha(t)$, decays as t increases. In this way, the map becomes increasingly less sensitive to input as training progresses.

3.2.2 Augmentations

The basic self-organizing map algorithm as described above lends itself to several common augmentations and alterations. In most implementations, some number of these possible augmentations are employed, depending on the specific application.

Growing Self-Organizing Maps

Growing self-organizing maps (GSOMs) represent a class of SOMs that, as the name implies, are not initialized to a static size. The idea of growing maps is detailed in (Dittenbach et al., 2000). The motivation behind the GSOM is that the exact topology and size of a map often has a large effect on the training process of the SOM. By taking away the somewhat arbitrary design choice of size, GSOMs allow one more aspect of the map to be determined by statistical regularity, not by a programmer.

GSOMs are initialized as very small maps, often as small as of size 2x2. The basic SOM training algorithm is used as described previously. After some fixed number of training iterations, the map grows. The map does not grow arbitrarily, but by inserting new cells within an area of high dissimilarity.

A cell e is selected as the "error cell" by finding the element with the highest value of quantization error (qe). The value of qe for each cell, i, is found by calculating the sum of the distance between the cell's vector, m_i, and each input vector, x_j, that has been matched to the particular cell during the training process.

$$qe_i = \sum_j dist(m_i, x_j) \tag{3.3}$$

Once the cell e with the highest qe has been found, the most dissimilar neighbor,

n, of that cell is identified by applying the distance metric to each immediate neighbor of e. A row or column is then inserted into the map between e and n. The vector of each new cell is initialized to be the average of the two vectors on either side of the new cell.

A simple illustration of the growth process is shown in Figure 3-2

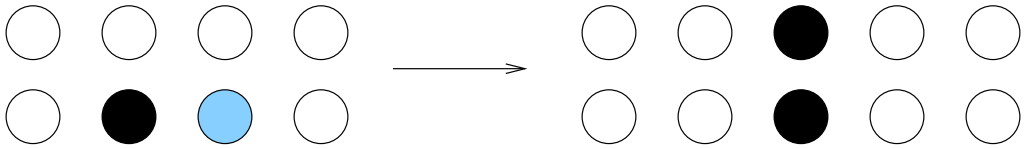

Figure 3-2: SOM growth example. In the initial map, shown on the left, the black element is selected as the error cell and the grey element as its most dissimilar neighbor. A column (shown in black) is then inserted between these cells in the updated map shown on the right.

Cluster Association

Once a self-organizing map has been sufficiently trained, clusters of similar inputs develop. These clusters are interesting for visualizing trends in the input set, but play a far more powerful role once cluster association is employed. Cluster association is most useful when two or more maps are run in parallel, perhaps on different representations of the same input. By associating a cluster in one map with a cluster in another, not only does regularity within each map become apparent, but the regularity between clusters is also learned. This process of cluster association plays an important role in learning associations between often related inputs.

Each map acts locally on its input and develops its own clusters. Once trained, the clusters are identified and each cluster is assigned an identifier. Then, on subsequent inputs, whenever a cluster a in map A is excited at the same time as a cluster b in map B, a mapping between clusters a and b is recorded. The strength of this association increases with each instance of this particular simultaneous excitation. Once the association becomes sufficiently strong, it is reasonable for the system to infer that if cluster a is excited by some input, then an excitation of cluster b cannot be far off.

3.3 Implicit Representation

There exists a fundamental difference between the two basic types of representations, explicit and implicit. An understanding this of distinction and the vital role that implicit representations play in biological systems is the final step in understanding the power of self-organizing maps.

Explicit representations are highly formalized encapsulations of information and are generally what people think of when they mention a "representation." For example, the LCS representation described in Section 2.2.1 is an explicit representation. The same formalism is shared among all instances of this representation. Generally, explicit representations are very much preferred by engineers and computer scientists because of this formalism. Moreover, creating algorithms that operate on explicit representations are usually easy to conceptualize and analyze.

Implicit representations, on the other hand, generally develop in a somewhat unpredictable manner and in a way that is not so easily understandable to the outside observer. The clusters of regularity within a trained SOM are, for example, an implicit representation of statistical regularity. There is no explicit knowledge within the map about the fact that inputs should fall into a particular clustering or even that any clustering should develop at all. This knowledge is represented *implicitly* by connections and spatial ordering.

Assuming that SOMs succeed in capturing information, namely regularity, in an implicit form, the question remains as to why this is more useful than a more explicit form of the same knowledge. First, given the biological evidence mentioned in Section 3.1, it seems likely that the brain functions and learns through the use of implicit representations. The brain does not resemble a relational database or have any predefined notion of how inputs should be clustered. If AI systems are to be built that mimic the functions of the human brain, then this fact should be exploited. Implicit representations allow highly flexible systems to be built that are not tied to a particular representation or type of knowledge.

The second, and possibly more convincing, argument for the use of implicit rep-

resentations lies in the fact that if a system is to observe the *real* world and operate on *real* inputs, then the space of inputs is so large and the amount of information to be represented is so vast that it is foolhardy to think that engineers can devise explicit methods of capturing the knowledge. Scripts (Schank and Abelson, 1977) are an example of a proposed method for representing common patterns of input, specifically event sequences. The problem with scripts is that properly accounting for every common pattern of input is a nearly impossible undertaking. Scripts may work well in a highly constrained domain, but inevitably fail once a large class of possible inputs is allowed.

Defining regularity based on neural clustering and associations between those clusters does not necessitate the explicit definition of each possible piece of regularity. The regularity simply "falls out" once proper training has taken place.

Chapter 4

Other Background

The implementation of self-organizing event maps makes use of several other previously developed tools. This chapter discusses these tools and lays the final groundwork for the presentation of the self-organizing event map architecture.

4.1 Threads

Threads are a data structure that stores the semantic meaning of a concept in the form of a linked chain of other semantic concepts (Vaina and Greenblatt, 1979). Originally proposed by Lucia Vaina and Richard Greenblatt, threads were intended to form the basis of a semantic memory system. Although threads never came to be the "silver bullet" that was hoped for, they are still relevant as substructures within memory systems.

The motivation behind threads comes not only from computational feasibility, but from biological evidence as well. Psychological studies have found that humans tend to remember the meaning of concepts in terms of ordered lists of other concepts (Vaina and Greenblatt, 1979). For example, a human may represent a terrier as an animate thing, an animal, and a dog.

A thread is a very simple data structure, consisting only of linked semantic nodes. The linking is done in a loop-free manner and can easily be expressed as a line of concepts. For example, the terrier mentioned above may be represented in thread

form as,

`terrier` → `animate-thing` → `animal` → `dog` → `species-of-dog`

Note that the key word ("terrier" in this case) is generally shown at the beginning of the chain, while the rest of the chain is ordered from most general (superordinate) to most specific (subordinate). The key is not included in the stored thread, but is shown to denote the concept to which the thread is linked.

The original Vaina and Greenblatt paper presents many further examples, uses, and modifications of the basic structure. For the purpose of this thesis, only three basic aspects of threads need be highlighted:

1. Threads are loop-free linked lists of semantic nodes
2. Every thread is tied to a key (often referred to as the *stimulus*)
3. The elements of a thread are ordered from superordinate to subordinate

4.2 The BridgeSpeak Parser

The natural language parser employed by the event map system is an existing program written by Patrick Winston that is currently included as an interface module in the Bridge System. This parser, termed *BridgeSpeak*, generates representations of events in the form of LCS frames, as described in Section 2.2.1. The vocabulary of BridgeSpeak is relatively limited in comparison to larger natural language systems, but the system is capable of parsing simple sentences concerning the interactions of objects. For example, a sentence such as, "The little boy walked from the tree to the car and rode to the store," can be parsed successfully.

Not only does BridgeSpeak parse sentences into LCS frames, but the resultant elements of the frames are annotated with threads (see Section 4.1 for a discussion of threads). The parser includes a knowledge-base containing simple classification hierarchies and traits of both general and specific objects. For example, a parse of the sentence given above would include an annotation on the "boy" object resembling the following:

`boy-132` → `animate-thing` → `human` → `male` → `young` → `little`

In this case, `boy-132` is the specific identifier generated for the boy object and the elements of the thread represent the knowledge of the object in increasing order of specificity.

4.3 Intermediate Features

The matching and model refinement procedures developed in Chapter 5 make use of the idea of intermediate features presented by Shimon Ullman in (Ullman et al., 2002). Ullman presents an argument and supporting data for the use of features of intermediate complexity in the classification of images. Ullman argues that the human visual system analyzes objects in stages, beginning with more local features and progressing through larger, more complex features. The data collected from experiments on human visual classification shows that it is features in the intermediate area of this complexity spectrum that prove to be the most useful. For example, Ullman presents evidence that it is not simply the shape and size of a human nose that makes it recognizable, but that the relationships (distance, angles, etc.) between the nose and the other features of the face are used far more heavily in recognition tasks.

Although the research described in (Ullman et al., 2002) primarily concerns the use of intermediate features in image classification, parallels can be drawn to other domains as well. A large portion of the event map system described in this thesis makes use of matching and "recognition" of events. Thus, this thesis extends the idea of intermediate features beyond the visual domain and into the realm of event perception. The "intermediate features" employed in the event map architecture are generally pairs of more elemental features. These pairs make explicit the relationships between objects, places, directions, and actions within an event. These relationships prove to be very useful in the matching of events. A further discussion of the use of intermediate features in the event map system follows in Chapter 5.

Chapter 5

Architecture and Implementation

The basic self-organizing map structure and algorithms described in Section 3.2 have been adapted and extended to function using *events* as the input vectors. This section details the design and implementation of the event map system. Further, this chapter presents the results of the implementation and provides data supporting its effectiveness in achieving its goals.

5.1 Architecture

The event map system has been designed to achieve several goals that have been motivated and discussed in previous sections. In summary, the goals of the system are:

1. Function unsupervised and as an online learning algorithm
2. Bring out regularity in the events to which the system is exposed
3. Build clusters of similar events for future use in cluster association
4. Provide functionality for the detection of exceptional events

To achieve these goals, a system was designed that makes use of the tools and ideas presented earlier. The high-level (user-oriented) architecture of the system is as follows:

1. An event is presented to the system in the form of an English sentence
2. The sentence is parsed into a lexical conceptual semantics frame
3. The input frame is presented to the event map and is matched against all existing data points in the map; a "winner" cell is selected whose event description most closely matches the input frame
4. The winning cell and its neighbors are updated to reflect the influence of the input data
5. The process repeats for each input event
6. The map becomes increasingly "trained" as clusters of similarity begin to appear
7. Once sufficiently trained, the map is capable of identifying events as being exceptional (i.e. not consistent with previously seen input events)

For the purpose of this thesis, the definition of an event will be limited to what will be termed *simple events*. Simple events are scenarios of a predefined form, involving a single actor, action, and a path. An arbitrary number of adjectives are also allowed. Examples of simple event are, "the cat climbed up the tree," and, "a small boy ran from the big ugly dog." This restriction is made for several reasons. First, simple events are guaranteed to be parseable by the BridgeSpeak parser into predictable forms. Second, these events are structured enough that a large number of "random" instances can be generated when the event map is initialized. Finally, the matching and blending algorithms described below are easier to conceptualize and study when events are of a bounded size. Because many complex events can be broken into sequences of simple events, this restriction does not necessarily represent a significant limitation to the effectiveness of the event map system.

5.1.1 The Event Map

The heart of the system is the map itself. The event map is a grid of elements, with each element representing an event. The elements do not necessarily correspond

exactly to events that has been observed, but represent *models* of typical events. Through implementation of the SOM algorithm as described in Section 3.2.1, the event map organizes itself during a training phase into clusters of similar events. Once trained, the event map provides a representation of event regularity, thus facilitating the creation of tests that discern between typical and exceptional events.

Topology and Size

The event map is a two-dimensional grid of data points. The exact size and topology of the map is a somewhat arbitrary design choice. In the current design, the map is simply a square grid of points. The edge length of the map is an initialization variable (for some context, maps using edge lengths varying from one to fifty have been created and used). The size of the map has a profound effect on the map's effectiveness in creating useful clusters and is dependent upon the nature of the events to which the map will be subjected. If the input space is highly regular, a smaller map will be more effective, as the amount of unused space will be kept as small as possible. A more diverse input space lends itself to the use of a larger map, as a larger map allows more clusters to effectively develop.

Map Elements and Initialization

Each element of the map is itself an event, described by an LCS frame. These events can be matched and updated as described below, but the basic structure remains the same throughout the training process. The maps used in this design are statically sized and each element must be initialized to some value. In the current design, each element represents a randomly generated event. In this random initialization process, a database of nouns, verbs, and modifiers is used to generate sentences of a predefined form. Each sentence follows the form, {`article`} {`noun`} {`verb`} {`direction`} {`article`} {`noun`}. Therefore, a random sentence may be, "The cat flew under a cow." The random sentence is then parsed into the LCS form and used as a data point on the map. This initialization process allows obviously non-sensical events to be created, but this nonsense is eliminated by statistical regularity during

the training process.

Viewers

One of the major benefits of self-organizing maps is that they lend themselves nicely to simple, two-dimensional viewers. Two basic viewers are used in this design. One viewer simply displays the map and its elements. All elements can be selected to reveal the details of the events they represent. Figure 5-1 shows an example of the simple SOM viewer.

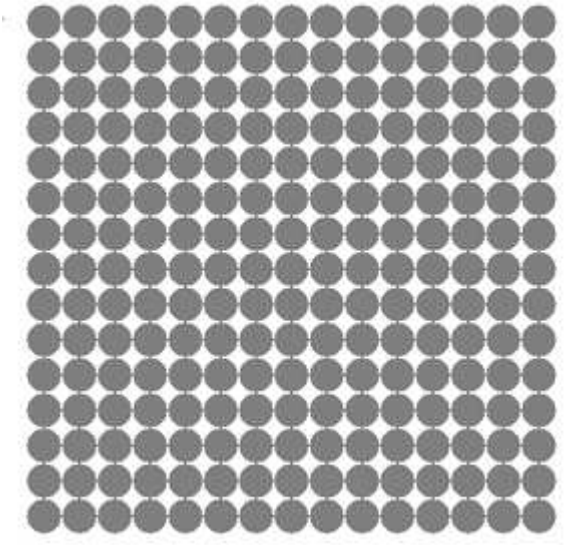

Figure 5-1: Example simple SOM viewer. The event map is shown as an n by n grid of elements, each element corresponding to a model event. In this view, all elements are colored grey.

The second type of viewer is more useful in viewing the state of the map as it undergoes training and allows developing clusters to be observed. This viewer is essentially the same as the simple topology view, but each event is assigned a color corresponding to the average "distance" between itself and its eight immediate neighbors. The distance metric used in this calculation follows the same form as the matcher described in Section 5.1.3. The elements are colored with varying shades of red, with bright red representing perfect similarity (the element is an exact match to all of its neighbors) and black representing complete dissimilarity. Once colored in

this manner, clusters can be observed in a trained map and a subjective evaluation of the map's training state can be made. An example of the average distance viewer is shown in in Figure 5-2.

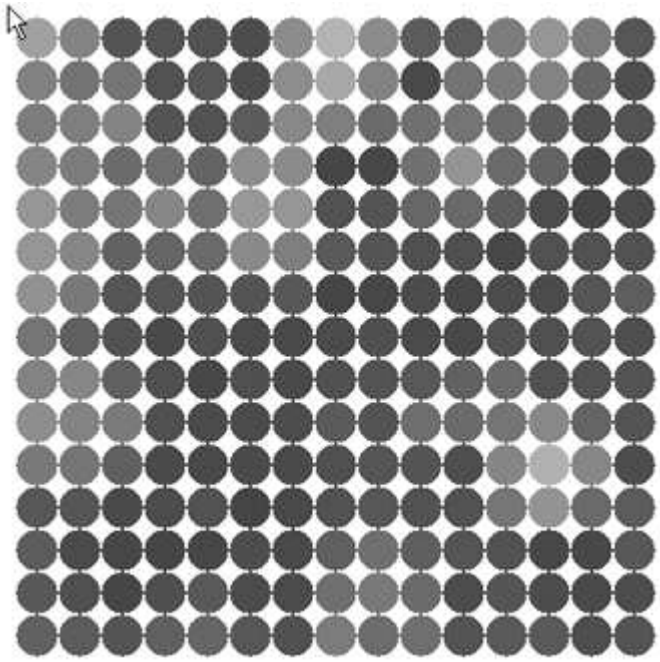

Figure 5-2: Example distance element viewer. The map is displayed in the same manner as in the simple SOM viewer, but each element is assigned a color according to how similar the cell is to its neighbors. High similarity is denoted by lighter color.

5.1.2 Creation of LCS Frames

The creation of the LCS structures used by the event map is handled by the BridgeSpeak parser described in Section 4.2. This parser reads English descriptions of events and generates frames based on its knowledge of verbs, nouns, and other common words. The atomic parts of each LCS frame also make use of the thread structures and it is these thread structures that are vital to the matching and refinement processes.

5.1.3 Matching

The mechanism for matching an input event against existing map elements is one of the most important aspects of the system. The matcher used in the current design makes use of the concept of intermediate features described in Section 4.3. For each pair of events to be tested, the intermediate features of each event are extracted from the LCS representation. The intermediate features of the first event are then matched against the intermediate features of the second to produce a final match score. Details of this process are described below.

Intermediate Feature Creation

Before matching takes place, the intermediate features of each event are extracted from the LCS descriptions. In the event map system, intermediate features are represented as *feature pairs*, each of which is simply a pair of more atomic features. "Atomic" features are defined as the objects, verbs, places, and directions involved in the event. The extraction of feature pairs is made most clear through an example.

Given the LCS description of the event, "the dog ran to the car," the intermediate feature set would include four pairs:

```
(dog, run)
(dog, car)
(to, car)
(at, car)
```

Each pair of nouns comprises one pair (`(dog, car)` in this case). Each actor and its associated action comprises another pair (`(dog, run)`). Each place and the direction involved with the place comprises a pair (`(to, car)`). Finally, each place and the intransitive preposition involved with the place comprises another pair (`(at, car)`).

This generation process does not simply create all pairs of features. The exact pairs to create have been chosen carefully to represent *important* aspects of the event. For example, in the event described above, pairs such as `(dog, at)` and `(dog, to)` have been omitted because they do not encapsulate relevant information about the

event. For simple events, only four feature pairs are generated, as in the above example.

It should be noted that the creation of feature pairs preserves the threads that accompany each noun. For example, each instance of "dog" appearing in the pairs above also contains the thread corresponding to that particular dog. This maintenance of threads is vital to the matching process described below.

Feature Pair Matching

Once events have been parsed into their intermediate feature sets, the matching mechanism compares the feature pairs of each event and generates a match score. As previously discussed, the current design assumes a fixed number of feature pairs for each event. Given simple events, each feature pair falls into one of four categories: (object, object), (object, verb), (direction, object), or (place, object). In this formulation, an "object" is any noun and is therefore annotated with the appropriate thread. A direction is one of the path-types allowed by the LCS representation (to, toward, from, away from, from, and via). A place is one of the place-types allowed by the LCS representation (at, on, under, above, and below).

The feature pair abstraction allows the matching algorithm to focus on matching only pairs of the same type. In other words, given two events, each feature pair from the first event must only be matched against its counterpart from the second event. The matching algorithm is therefore linear in the number of feature pairs.

Within each comparison of corresponding feature pairs, the known hierarchies of nouns, verbs, and directions are used. The elements of each pair are compared using simple "is-a" tests, with more weight being given if the two elements match exactly. For example, "boy" and "human" would match because a test of (is-a "boy" "human") passes. For objects, these is-a tests use the threads that are attached to each instance. For directions and places, a simple hierarchy of types is queried. For example, "to" is a specific type of "toward" in the LCS framework, and therefore (is-a "to" "toward") evaluates to true. The other knowledge regarding the hierarchy of keywords used in LCS frames follows directly from the definitions made by Jackendoff

and discussed in Section 2.2.1.

All tests for matching are performed symmetrically. Because is-a tests are inherently non-symmetric, two tests are performed on each pair of elements, one in each direction. If either test evaluates to true, the match is deemed successful.

Whenever two elements of a corresponding pair of feature pairs are matched successfully, the "match score" of the feature is incremented by 0.25. The matching of each pair of feature pairs involves four comparisons, and therefore the total possible score for a set of corresponding feature pairs is 1.0. To find the total match score for an event, the scores of each set of corresponding feature pairs are averaged. The total possible match score for two events is therefore also 1.0.

An example of this matching and scoring process is illustrated below. In this example, two feature pairs are being compared: `(dog, tree)` and `(animal, car)`.

1. Initialize score = 0.0
2. (is-a dog animal) OR (is-a animal dog): TRUE: score = score + 0.25
3. (is-a tree car) OR (is-a car tree): FALSE
4. (is-a dog car) OR (is-a car dog): FALSE
5. (is-a tree animal) OR (is-a animal tree): FALSE

The match score of the above pair of feature pairs is therefore 0.25. This process is repeated for each set of corresponding feature pairs in the event.

5.1.4 Refining the Map

As prescribed by the general self-organizing map algorithm, the event map refines itself at each training step. After successfully locating the most closely matching ("winning") element of the map, the matched cell and its neighbors are refined to more closely match the input event. This process also uses an intermediate features representation as an internal structure, along with threads and hierarchies.

As in the matching process, the model refinement procedure also operates on sets of *corresponding* feature pairs. Again, because each event includes four distinct feature pairs, there are four parts of each pair of events to examine. Whereas the matching algorithm uses is-a tests on each element of each set of corresponding features, the refinement algorithm uses a generalization and/or specification step for each pair of elements.

In this generalization and specification step, two elements are compared based on their threads (or hierarchies) and a new element is produced that is the "most specific non-contradicting" blend of the two inputs. This notion of non-contradiction is easily understood in the context of threads. This description makes use of the notion that one thread can "cover" another if each of the elements in the second thread are present in the first. Given two threads, A and B, a third thread, C, is produced according to the following process:

1. Initialize thread C to be a copy of thread A

2. Calculate two boolean values, c1 and c2, where,
   c1=true iff thread B covers thread A and,
   c2=true iff thread A covers thread B

3. If c1=false and c2=true then,
   remove the most specific element of C

4. If c1=true and c2=false then,
   add the most specific element of thread B to the end of thread C

5. If c1=false and c2=false then,
   remove the most specific element of C

Once this thread creation process is complete, the new thread C replaces thread A in the corresponding element. An illustration of this process is provided below. In this example, thread A is the thread attached to a "plant" object:

`plant` → `inanimate-thing` → `living-thing` → `plant`

Thread B is the thread attached to an "oak tree" object:

`oak-tree` → `inanimate-thing` → `living-thing` → `plant` → `tree` → `hardwood` → `oak-tree`

Thread C is initialized as a copy of the plant thread. Thread A (the plant) does not cover thread B (the oak tree), but thread B does cover thread A. Therefore, the most specific element of thread C is removed and thread C is returned as the new updated thread. Thread C in its final form:

`plant` → `inanimate-thing` → `living-thing` → `plant` → `oak-tree`

When a pair of directions or a pair of verbs are to be blended, the new element is simply the first common superclass of the two inputs. This operation is performed using a hierarchy of LCS elements. It should be noted that a more recent version of the BridgeSpeak parser generates threads for verbs as well as for objects. Given this new functionality, the blending of verbs can be done using the same algorithm as shown above. At the time of implementation, this functionality was not present in the parser, and the blending of verbs uses a separate hierarchy.

This refinement procedure does not seek to generate a "perfect blend" of the two inputs, but only to make one event "slightly more like" the other. Given a sufficiently effective training process, most aberrations introduced by this refinement procedure will be reconciled.

5.1.5 Saliency Detection

The term *saliency* is used here to denote how exceptional (i.e. non-typical) an event is, given the previous experiences of the event map system. Determining the saliency of an input event is essentially the process of deciding how closely the input matches existing events within a trained map. There are three basic ways of implementing such a procedure within a self-organizing map, each of which is described below.

Hit Counting

Determining how exceptional an event is via hit counting necessitates the addition of a piece of state to each map element. This state records the number of times that an element has been selected as a winning cell during training and, perhaps, also records the number of times that an element has been selected as a neighbor cell. For example, an element's "hit score" might be incremented by 1 each time it is selected as a winning cell and incremented by 0.5 each time it is a neighbor. This aggregate hit count can then be used to determine if an event is exceptional.

Given a new input event and a selected winning element of the map, the hit count of the winning element is checked. If this count is above a certain threshold, the input event is considered typical. If the count is below the threshold, the input event is flagged as exceptional.

Such an approach is often ineffective and requires the addition of state to each element, taking away from the purity of the self-organizing map algorithm.

Cluster Detection

Identifying exceptionality through cluster detection requires that a clustering algorithm has been run on a trained map and that an explicit notion of clusters exists within the system. Assuming that clusters within a map have been identified and a winning cell selected for a given input event, the process of saliency detection reduces to simply answering the question, "Does the input event most closely match an element that is within a known cluster?" Because a map is usually not fully clustered, this method is effective in determining if a given event fits into a cluster or if it simply matches a noise point that lies outside of any well-defined cluster of other events. If the input does not match a clustered element, then it can be considered to be more highly exceptional.

Distance Scoring

To determine exceptionality via distance scoring, a winning element is first selected, as in each of the above methods. The matcher is then used repeatedly on each neighboring element to generate a similarity score between the input event and the neighbor. The average distance between the input event and each of the neighbors (including the winning cell itself) is calculated, resulting in a numeric score.

Distance scoring has two nice properties that separate it from the two metrics described above. First, no additional information about previous hits or existing clusters is required. This property is made relevant by the concept of implicit representation described in Section 3.3. Second, distance scoring results in the creation of numeric scores, as the name implies. The exceptionality of an event is therefore not a binary value (as in cluster detection). Because of this property, multiple events can easily be compared. A common use of exceptionality tests in biological systems is in the comparison of multiple hypotheses. Distance scoring easily supports the comparison of multiple hypotheses to determine which is most (or least) common.

For the above reasons, distance scoring is the metric employed by the current self-organizing event map implementation.

5.2 Implementation and Results

The architecture has been implemented in Java. Event maps of sizes ranging from one to 2500 elements have been created, trained, and tested.

When initially started, the event map system creates an interface, an example of which is shown in in Figure 5-3. The map is randomly initialized and shown in the window, along with text input boxes for both a training sentence and a "story" file name. The map can be trained either by providing individual sentences into the large text box (and pressing "Train") or by providing the name of a story file.

When a single sentence is provided, the sentence is parsed via BridgeSpeak and matched to a cell of the map using the algorithms described in Sections 5.1.3. The winner cell and its neighbors are then refined using the process described in Sec-

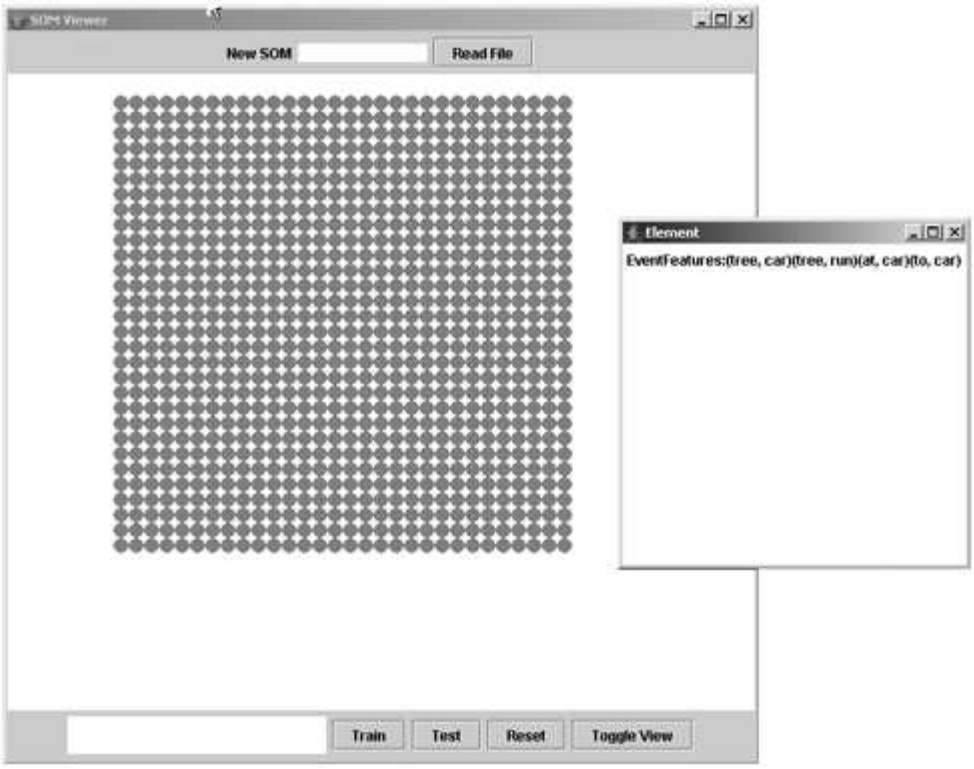

Figure 5-3: The full event map GUI at startup. The map is shown using the simple SOM viewer and text input boxes for training sentences and story file names are provided. The SOM can be trained by either entering a story file name and clicking "Read File" or by entering a single sentence in the lower box and clicking "Train." The view can be toggled between the simple SOM view and the distance element view by clicking "Toggle View." When an element of the map is clicked, a window appears as shown on the right. This window displays the feature pairs of the selected event.

tion 5.1.4. The radius of the neighborhood is a system parameter that defines how far out from the winning cell the refinement process should extend. For example, given a radius of 1, the neighborhood of a cell will be the 8 cells immediately surrounding it. Given a radius of 2, the neighborhood will consist of not only the immediate 8 neighbors, but the next ring of cells as well. Best results were found while using a radius of 1 for smaller maps (edge length less than 20) and while using a radius of 2 for larger maps. Radii larger than 2 tend to give too much weight to incoming events and little consistency is maintained.

In most cases, the example maps shown in this section will make use of the average

distance coloration viewer described in Section 5.1.1. In this view, each cell is colored according to how similar the cell is to its neighbors. By viewing the maps in this way, clusters are easily seen and the effectiveness of the training process can be evaluated.

The rest of this section discusses the types of training scenarios that were given to the map and the results of that training, along with examples of trained event maps. The usefulness of the map as a saliency detection mechanism is described and, finally, the performance of the event map system is discussed.

5.2.1 Training Scenarios and Results

The event map system supports the reading of "stories" to expedite the training process. These stories are simply lists of event sentences that are read from files. Two basic types of training scenarios were used as test cases to illustrate the effectiveness of the event map system in detecting regularity in the input. These two basic classes of inputs will be referred to as *regular* and *irregular*.

Regular Scenarios

The stories that describe relatively normal and non-contradictory events are considered regular. An example of a regular story is provided in Appendix A. These regular stories consist of events that would be considered "normal" by someone with a very cursory common-sense knowledge of the world and its components. For example, dogs run, birds fly, cars drive, boys climb trees, and so on. Nowhere in a regular story will a cow swim or a house walk down the street.

An example map trained using a regular story is provided in Figure 5-4.

Irregular Scenarios

Randomly generated, irregular scenarios were also used, mostly as a control group. Because the event map system is designed to recognize regularity in its training samples, a set of irregular events was devised to show that, in the absence of regularity in the input, little regularity develops in the map. The irregular training events were

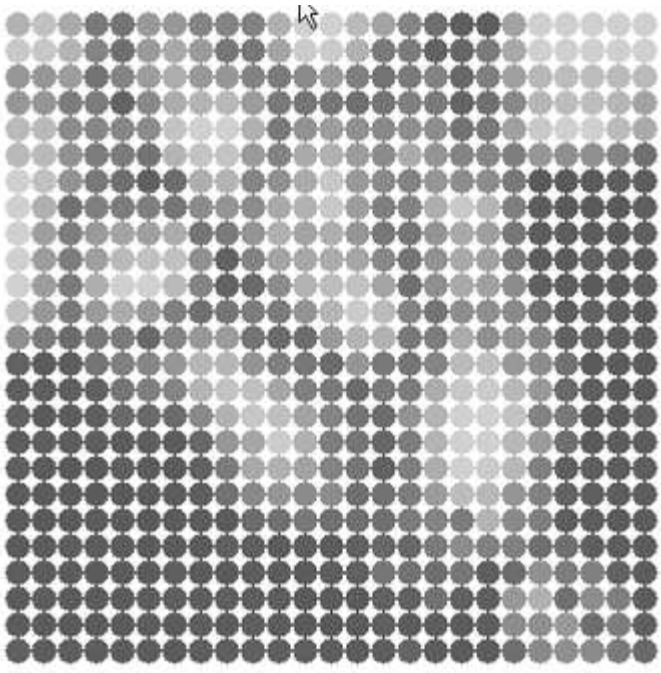

Figure 5-4: A trained map, shown using the distance element view. This map was trained using the story in `regular.txt`. 10 readings of this story were done. Clusters of similar events are represented by light coloration. An example cluster can be seen in the upper right corner. This cluster corresponds to events involving birds traveling from other animals, brought on by repeated exposure to events describing birds flying from cats and people.

basically randomly generated and allowed to include contradictory and nonsensical statements.

A map trained using irregular input is shown in Figure 5-5. The clustering that can be seen in a regularly trained map does not appear after training with an irregular training set. A closer examination of an irregularly trained map does indicate that some clusters begin to form, but these clusters are the result of over-generalization. For example, a cluster may basically represent an event in which, "a thing moved to a thing." This over-generalization phenomenon increases in severity with further training cycles. A map trained for many cycles with random and irregular data will often become one large cluster of over-generalized event models.

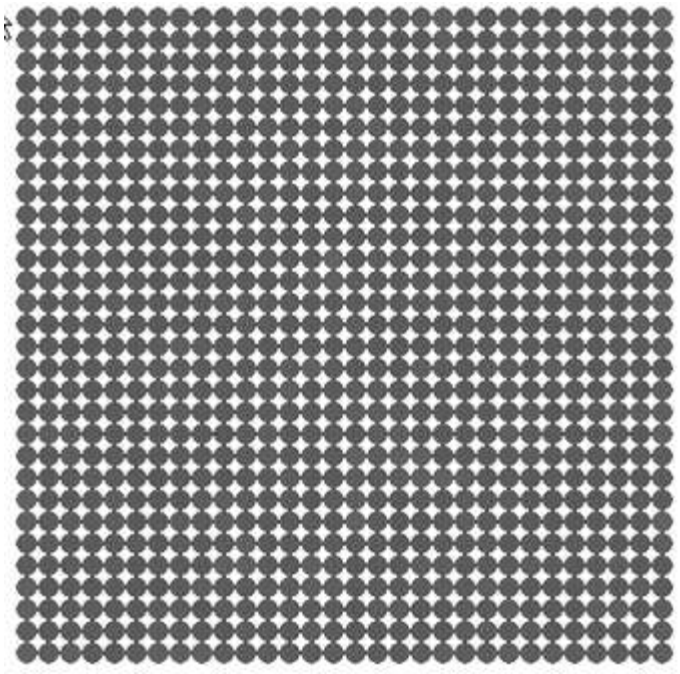

Figure 5-5: A map trained using randomly generated events. Few clusters have appeared because the events that occur in the story are contradictory and little regularity can be gleaned.

5.2.2 Saliency Results

A major goal of the event map system is to provide a mechanism for discerning between typical and exceptional events. Once a map has been trained (preferably using a regular story), the "Test" button of the interface (see Figure 5-3) can be used to generate a saliency score for a new event. This score represents the degree to which the event is considered exceptional by the event map. When a sentence is typed into the text input box and the Test button is selected, the system parses the sentence and generates a score by using the distance scoring mechanism described in Section 5.1.5. In this process, the test event is compared to the map elements and the most similar cell is selected. The average distance between the input event and the neighbors of the selected cell is the score. The score is therefore a representation of how unusual the input event is. An event with a score of 1.0 is extremely exceptional, while an event with a score of 0.0 is highly typical. Both of these extremes are unlikely, and most scores fall towards the middle of this spectrum.

Generating only a score alone is not tremendously useful until the score is put into context with others. Given two events, the scores of each event can be compared to determine which event is most exceptional. The ability of the system to make useful determinations of this type also increases with the amount of training that the map has undergone. The graph in Figure 5-6 shows two events' scores versus the number of training cycles. In this example, what would most often be considered a "typical" event and an exceptional event are tested. At the beginning of the training cycle the scores are somewhat misguided, but the distinction between the two events becomes significantly clearer as the map becomes trained.

5.2.3 Performance Characteristics

The following discussion assumes that the value n defines the length of an edge of a square event map (thus the map has n^2 elements). All quoted times are derived from tests on a machine having a 300MHz Pentium II processor, 512Mb of RAM, Windows XP Professional operating system, and a HotSpot Java virtual machine.

The dominating limitation of the current implementation is the initialization process in which randomly generated English sentences are converted to LCS frames. During initialization, each of n^2 randomly generated sentences must be parsed and inserted into the map. A sampling of the initialization times for various sizes of n is given in Figure 5-7.

Each training cycle in which a single event is used to refine the map takes only approximately 0.75 seconds when $n = 25$. Nearly 76% of this time is taken by the initial parsing of the sentence, about 21% by the matching process, and only about 3% by the map refinement procedure.

Given this data, the natural-language parsing is clearly a bottleneck of the system, although individual training cycles run quite quickly, even on large maps. In general, it is feasible to create and train event maps of size $n \leq 50$ on a typical personal computer. Generating and using maps of larger sizes would most likely necessitate parallelization of the system or the use of a more highly tuned English parser.

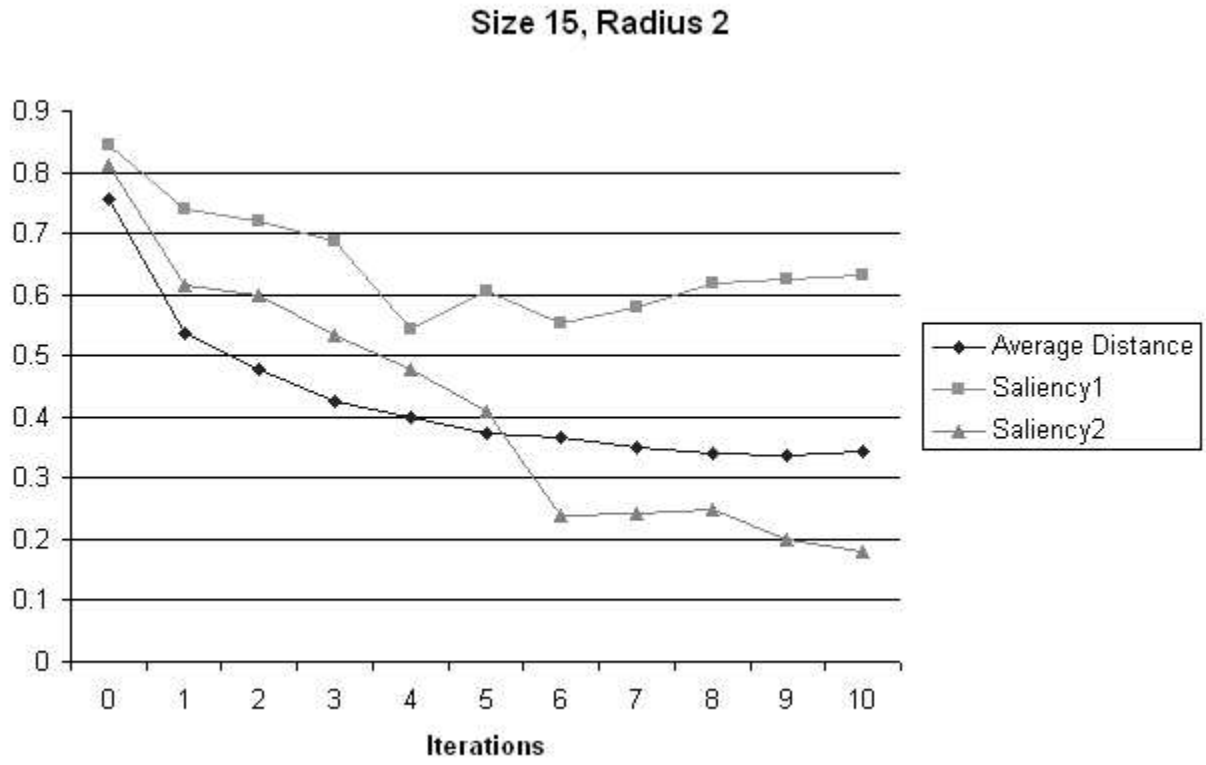

Figure 5-6: A graph of saliency scores versus number of training cycles (readings of the story in `regular.txt`) for two events tested on a 30x30 map. The square marked line corresponds to the score of the event, "a dog flew to under the table," and the triangle marked line corresponds to the score of the event, "a dog ran to the boy." The first event is deemed more exceptional by the system and the difference between the two scores generally increases with training. The value of the *map average distance* is also shown (diamond marked line). For each cell in the map, the average distance to all neighbors is found. The map average distance is the average of these individual cell averages. This value is a simple way of quantifying how well the map is trained – lower distances represent a more fully trained map.

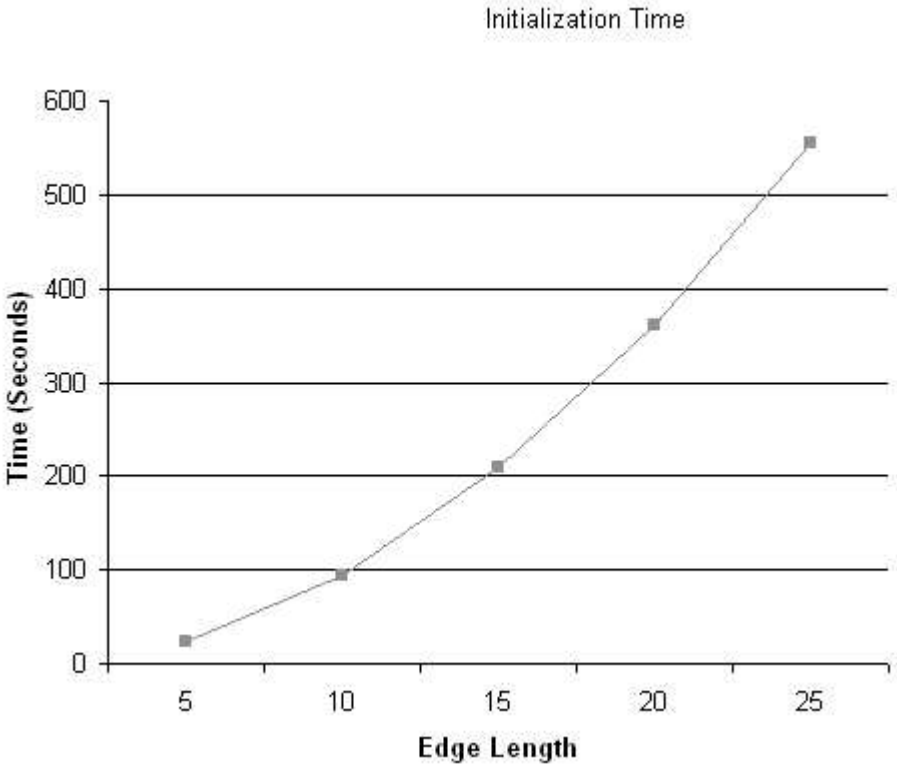

Figure 5-7: A graph of initialization time versus n.

Chapter 6

Discussion

The self-organizing event map architecture suggests several other potential uses that are not explored by the implementation described in Chapter 5. The implementation presented in this thesis has focused on the use of the SOEM architecture as a substrate for detecting exceptional events, but several other potential uses exist. This chapter offers suggestions for further implementations and applications of the SOEM architecture.

6.1 Representation Translation

Many representation-based AI systems make use of several different representations of the same data. The use of multiple representations allows several aspects of the data to be made explicit within the system. For example, the Bridge System represents events in both Jackendoff-style LCS frames (Jackendoff, 1983) and Borchardt-style transition-space frames (Borchardt, 1993). The LCS frames allow the trajectories and places involved in an event to be made explicit, while the transition-space frames allow the transitions to be readily observed. By using these two different representations of the same data, the Bridge System is able to generate both linguistic and visual descriptions of events.

Inherent in a system involving multiple representations is a need to translate data from one representation to another. In the Bridge System, for example, a rules-based

translator forms transition frames from LCS frames, and vice versa. The SOEM architecture would allow this translator to be less algorithmic and pre-programmed and to learn the associations between the different representations through exposure to input events.

An implementation of the SOEM architecture could be used to solve the representation translation problem within the Bridge System in the following way:

1. Initialize an event map that stores events in the LCS framework and that includes matching and refinement procedures that are specific to the LCS semantics.

2. Initialize a second event map that stores events as transition frames and that also includes procedures that make use of transition-space semantics.

3. Attach each map to a parser that generates the appropriate representation.

4. Repeatedly input events that are parsed by each parser and used as training samples on each respective map.

5. Once clusters of similar events within each map begin to develop, start recording the associations between each map. When an event matches within a cluster in the LCS map and the same event matches within a cluster in the transition-space map, increase the strength of association between those clusters.

6. Once cluster association has taken place, the maps and their associations can be used to translate between the two representations. Given an event described in LCS, match the event to the LCS map and follow the association link to the most closely associated cluster within the transition-space map.

There are, of course, many implementation details to be worked out before a complete translator can be built using the process described above, but the basic outline should prove useful.

The key point here is that the notion of a cluster is a representation that is shared among different maps, despite possible huge variations in the actual event

representations. By providing a common medium through which associations can be made and information passed, the SOEM architecture and its clusters provide a powerful way of creating an intermediary between representations.

6.2 Multi-Sensory Input

Systems such as the Bridge System also often make use of input that arrives via different sensory systems. For example, the Bridge System includes both linguistic and visual faculties, and seeks to make use of input from both perspectives. Associating inputs coming from different senses is another way in which the cluster associations between trained event maps can be used.

Generating associations between inputs arriving via different senses is simply a specific application of the representation translation procedure described above. An event map should be attached to each form of input and clusters of similarity allowed to form. Once clustered, the different inputs can be associated by cluster association. Put into the context of the Bridge System, these associations would amount to a crude accounting of how linguistic descriptions of an event are related to visual inputs and could be used to infer descriptions for future input events.

An example of the use of self-organizing maps in a multi-sensory system is provided by Larson in (Larson, 2003).

6.3 Content-Based Storage

The use of self-organizing structures allows for the creation of content-based storage systems. Extrapolation of this fact leads to the conclusion that self-organizing event maps allow for the creation of content-based *event* storage systems. Although the current implementation of the SOEM architecture is not intended to function as a memory system (at least as opposed to a regularity learning system), the basic architecture lends itself nicely to this purpose.

Each element of an event map corresponds to a particular type of previously seen

input. By matching new inputs to their corresponding places in the map, each input can be stored in a location that matches its content. Similarly, events can be retrieved by finding the most closely matching part of the map and fetching the previous inputs that have matched that location.

By storing not only the evolving event model in a cell of the event map, but also a store of inputs that have been previously mapped to each element, a simple event memory system could be built. This memory would store events together in clusters of similarity and would allow for simple recall and the finding of related events.

Chapter 7

Contributions

In this thesis, I have,

- Identified self-organizing maps as a general method for detecting regularity in an input space and applied the basic self-organizing map algorithm to the abstract notion of events so that clusters of similar events can be made apparent.
- Implemented self-organizing event maps in a functioning system that is capable of receiving natural-language input and organizing itself based on observed events.
- Provided an existence proof of how self-organizing event maps can be used to identify exceptional events and to provide models of typical event scenarios.

In addition to the key contributions listed above, I have argued that the use of implicit representations is superior to the more typical explicit representations, such as scripts, used in many event memory systems. Finally, I have shown how content-based event storage systems, representation translators, and multi-sensory systems can be built using self-organizing event maps as the fundamental data structure.

Appendix A

Sample Training Data

Regular story (`regular.txt`):

```
The boy ran to the tree.
The cat walked to the house.
The car drove to the house.
The bird flew from the cat.
The girl rolled toward the tree.
The dog walked toward the boy.
The boy threw the stick to the dog.
The dog walked to the stick.
A cat dove toward the bird.
The bird flew from the feeder.
The boy ran toward the dog
The dog ran from the boy.
A man went to the kitchen.
```

Bibliography

James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

Gary C. Borchardt. Causal reconstruction. AI Memo 1403, MIT Artificial Intelligence Laboratory, February 1993.

M. Dittenbach, D. Merkl, and A. Rauber. The growing hierarchial self-organizing map. In S. Amari, C. L. Giles, M. Gori, and V. Puri, editors, *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2000)*, volume 6 of *IEEE Computer Society*, pages 15–19. Como, Italy, 2000.

Ray Jackendoff. *Semantics and Cognition*, volume 8 of *Current Studies in Linguistics Series*. MIT Press, Cambridge, Massachusetts, 1983.

Teuvo Kohonen. *Self-Organizing Maps*. Number 30 in Springer Series in Information Science. Springer, third edition, 2001.

Stephen D. Larson. Intrinsic representation: Bootstrapping symbols from experience. Master's thesis, Massachusetts Institute of Technology, 2003.

Erkii Oja and Samuel Kaski, editors. *Kohonen Maps*. Elsevier, Amsterdam, The Netherlands, 1999.

Roger C. Schank. *Dynamic Memory – A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, 1982.

Roger C. Schank and R. Abelson. *Scripts, Plans, Goals, and Understanding*. Erlbaum Press, Hillsdale, NJ, 1977.

Shimon Ullman, Michel Vidal-Naquet, and Erez Sali. Visual features of intermediate complexity and their use in classification. *Nature Neuroscience*, 5(7):682–687, July 2002.

Lucia M. Vaina and Richard D. Greenblatt. The use of thread memory in amnesic aphasia and concept learning. AI Working Paper 195, MIT Artificial Intelligence Laboratory, 1979.

Patrick H. Winston. *Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition, 1993.