Cross-Domain Self Organizing Maps

by

Estanislao L. Fidelholtz

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

August 2006

[September 2006]

Author [illegible]

Department of Engineering and Computer Science

August 22, 2006

Certified by [illegible]

Patrick H. Winston

Ford Professor of Artificial Intelligence and Computer Science

M.I.T. Thesis Supervisor

Certified by [illegible]

Arthur C. Smith

Professor of Electrical Engineering

Chairman, Department Committee on Graduate Theses

Cross-Domain Self Organizing Maps

by

Estanislao L. Fidelholtz

Submitted to the Department of Electrical Engineering and Computer Science
on August 22, 2006, in partial fulfillment of the
requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I present a method for organizing and relating events represented in two domains: the transition-space domain, which focuses on change and the trajectory-space domain, which focuses on movement along paths.

Particular events are described in both domains, and each description is fed into a self organizing map. After these self organizing maps have been trained with enough events, the maps are clustered independently. Then, after the two self organizing maps are clustered, the clusters in the two maps are themselves clustered, creating links between trajectory descriptions and the transition descriptions.

Thus, I provide a method for relating events seen in multiple perspectives. After training with 1914 different sentences about motion, my implemented system noted that particular motions along a path are highly correlated with particular transitions. For example, "the bird flew to the top of a tree" is part of a trajectory cluster that is highly correlated with a transition cluster in which a motion appears and a distance first decreases and finally disappears.

Thesis Supervisor: Patrick H. Winston

Title: Ford Professor of Artificial Intelligence and Computer Science

Acknowledgements

I would like to thank my advisor, Patrick Winston. He showed me the light and helped guide my journey through the many disciplines of Artificial Intelligence. His vision and ideas helped push this research forward. Best of all, he did all this with a great sense of humor, which made this whole experience very enjoyable.

I also extend many thanks to all my friends and colleagues. They helped keep my morale up and helped me get through tough times. Many provided me with great ideas and with insightful discussions. There would be too many to name here, but to all of you I am grateful.

Above all, a very special thanks goes out to my family, especially my parents. They have inspired me to achieve as much as I can, and have given me the moral support without which none of this would have been possible.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

In order to understand human intelligence, we must first understand how humans relate experiences represented in different domains. Spelke demonstrated how the brain uses language in order to relate vision with spatial orientation (1990). In trying to understand human intelligence, we must understand how one can relate expressions in two different language domains. I outline a method to relate simple expressions in the two language domains brought forth by Borchardt and Jackendoff.

Borchardt created a model of how the human mind understands descriptions, elucidated on how the mind figures out relations between changes from descriptions. This *transition space model* constitutes a very useful portrayal of how humans detect change and relate these changes in their environment together to infer causality (Borchardt 1993). Jackendoff came up with a different type of representation by creating a simple grammar which allows us to understand any relationship or movement between two objects (Jackendoff 1983). I take both of these representations, and propose a method to find a relationship between the two.

In order to find relationships, we need a method to cluster sentences with respect to each domain. I propose using Self Organizing Maps [SOMs] for this task. This method provides a means to relate objects that can bootstrap its clusters from only knowing how to relate two objects in a single domain (Kohonen 2001). This gives a basis for learning from experience without the need of having predetermined clusters of sentences.

In order to feed sentences into SOMs, they have to be a special type which can relate events. Seth Tardiff demonstrated that this can be accomplished by using intermediate features in his Self Organizing Event Maps [SOEMs]. In his work, Tardiff created a system which more closely mimics natural memory systems and is free from the limitations of an explicit representation of regularity (2004). This system exhibits all the characteristics needed, providing a method to cluster the events fed into these SOEMs.

In this thesis, I contribute a method for relating events represented in two domains. The two domains used in this thesis are Jackendoff's Trajectory Spaces and expressions represented in Borchardt's Transition Frames. They are fed into a Self Organizing Event Maps and clustered. The clusters are then related one to another creating a link from one domain to the other. This provides a framework for relating two different domains.

2 Background

2.1 Transition Spaces

Gary Borchardt wrote his PhD Thesis around a radical position – that human thinking is centered on state changes, as opposed to the classical view that it is centered on the state itself. Following this radical view, Borchardt created transition spaces, a representation of events as a series of state changes in the variables involved in the event. These changes are from a reduced domain of size ten, which include 'appears', 'disappears', 'increases', 'decreases' and 'changes', as well as the negation of these five changes (1993).

If, for example, we were talking about a ball bouncing on the ground, the important variables might be the distance between the ball and the ground, the velocity of the ball, and the state of the ball. The Borchardt transition space would be similar to the one pictured in Figure 2.1 on page 15.

This sort of representation has been shown to expose constraints, and has allowed for the creation of how-to books that understand and can answer questions about devices they describe.

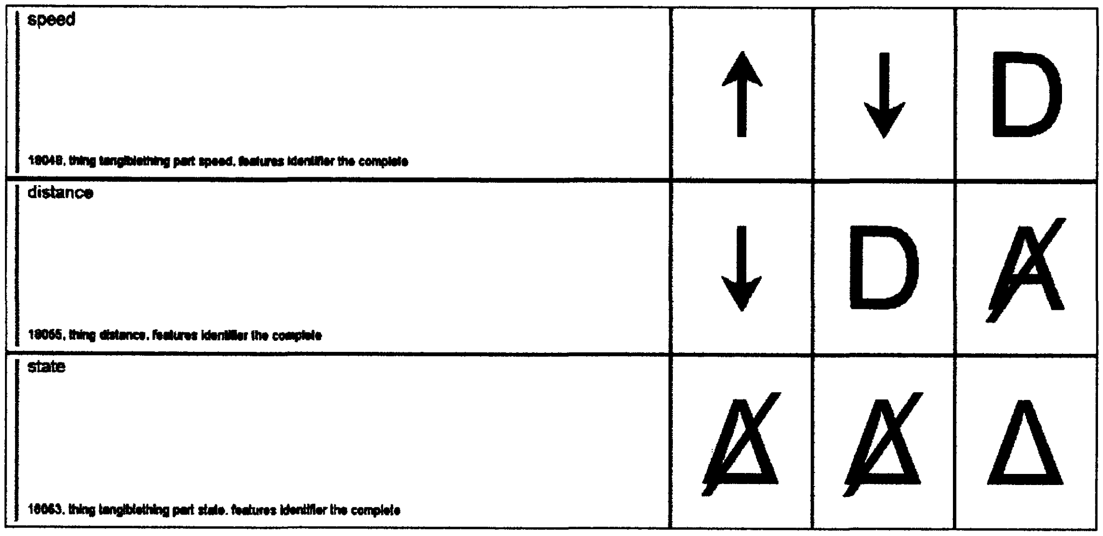

speed
18048, thing tangiblething part speed. features identifier the complete
distance
18065, thing distance. features identifier the complete
state
18063. thing tangiblething part state. features identifier the complete

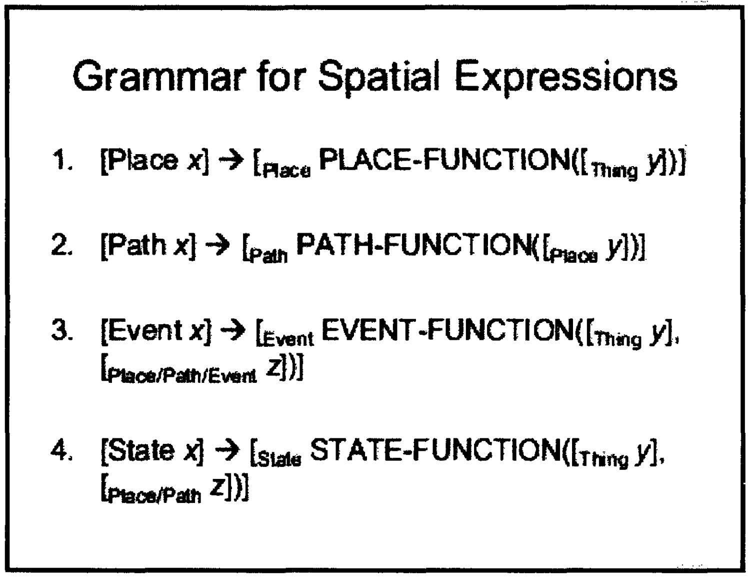
Figure 2.1 A Borchardt transition space depicting the event "The car crashed into the tree." The speed is that of the car, the distance is the distance between the car and the tree, and the state refers to the state of the car (figure taken from Winston 2006).

2.2 Trajectory Frames

Many cognitive psychologists, linguists and AI researchers subscribe to the idea that thoughts about anything are equivalent to thoughts about physical objects moving along trajectories in a physical space. For example, when a person thinks of the sentence "Rome moved away from democracy and towards an emperorship," a person 'visualizes' an object, *Rome*, moving in a physical space away from a physical object, *democracy*, towards another physical object, *emperorship.*

Jackendoff's work emphasizes how human language is about real or abstract objects moving along trajectories. To this end, Jackendoff created a grammar consisting of four rules which describes all spatial relations (1983). Look at Figure 2.2 (page 17) for a summary of the grammar rules. An example of how this grammar would work is illustrated by Figure 2.3below.

Moreover, Jackendoff showed how much of English can be mapped onto this grammar to create paths. This has laid the foundation for creating programs that can visually represent movement of objects along trajectories constructed from normal English sentences.



Grammar for Spatial Expressions

1. [Place x] → [$_{\text{Place}}$ PLACE-FUNCTION([$_{\text{Thing}}$ y])]
2. [Path x] → [$_{\text{Path}}$ PATH-FUNCTION([$_{\text{Place}}$ y])]
3. [Event x] → [$_{\text{Event}}$ EVENT-FUNCTION([$_{\text{Thing}}$ y], [$_{\text{Place/Path/Event}}$ z])]
4. [State x] → [$_{\text{State}}$ STATE-FUNCTION([$_{\text{Thing}}$ y], [$_{\text{Place/Path}}$ z])]

Figure 2.2 The grammar for Jackendoff's trajectory spaces.

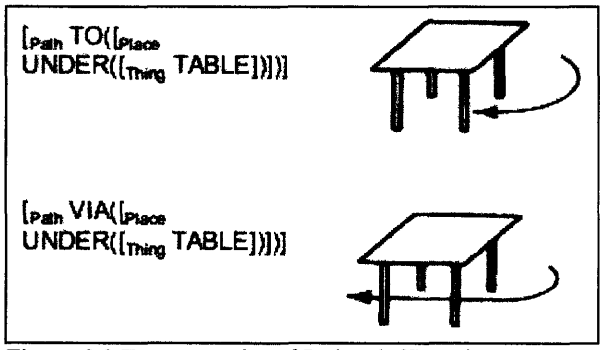

[$_{\text{Path}}$ TO([$_{\text{Place}}$ UNDER([$_{\text{Thing}}$ TABLE])])]
[$_{\text{Path}}$ VIA([$_{\text{Place}}$ UNDER([$_{\text{Thing}}$ TABLE])])]

Figure 2.3 Two examples of Jackendoff's trajectory spaces.

2.3 Self Organizing Maps

2.3.1 Description of Self Organizing Maps

SOMs are a learning method invented by Teuvo Kohonen that mimics biological systems and their ability to adapt to the regularities perceived in the environment. SOMs allow for unsupervised learning, clustering elements by their salient characteristics. Moreover, SOMs have the property of arranging higher dimensional data in a lower dimension while still preserving the topology of the underlying distribution (2001).

2.3.2 Algorithm of Self Organizing Maps

The following is the algorithm used to create the SOMs:

1. Initialize the map, or array, with random samples in each cell. These random samples must be of the same data type as what will be fed into the maps in subsequent stages. This gives us a basis for organizing this map so that it will group similar items together.
2. Feed the data points, one at a time. Each time you feed a new data element into the map, you follow the following steps:
    a. Find the data point which most closely resembles the new data point. This can sometimes be the most difficult task of the algorithm. It requires a method to measure the relative distance between elements.
    b. Insert new data point into the map, at or near the data point found in the previous step. There are several different variations as to how to do this. It is possible to simply replace the old element with this new one,

or one might be able to insert the element without destroying any previously placed data points.

c. Scale the neighbors so that they more closely resemble the new data point. This is an optional step, and also there are several ways this step can be implemented. When a simple numerical method is used, one can perform a weighted average calculation of the new point and the neighbor being changed, and place the result in the neighbor's place. When performing this with more complicated structures, such as sentences, one has to apply different mechanisms. Tardiff explained a method of doing this "averaging" with sentences by replacing certain words in the neighbor's sentence. The distance measure and the averaging technique have to be related.

This particular tool, the SOM, is good for the purposes of this thesis. It provides a means for clustering data in a low dimensional space, and has no assumptions about the underlying distribution of the data. In other words, learning about the clusters occurs in an unsupervised fashion.

2.4 Threads

2.4.1 Description of Threads

Vaina and Greenblatt proposed a new type of semantic memory called thread memory. They link semantic nodes, representing concepts, using keyed multilink, loop-free chains. These links represent a hierarchical structure, where the links run from superordinate to subordinate categories. This type of memory provides a model for how concepts can be stored, cross-referenced, indexed, retrieved, accumulated and modified 1979).

A typical thread might be:

mallard → living → animal → bird → duck → species-of-duck.

The first item in this notation is the token, or key, which in this case it would be mallard. It is the stimulus by which the rest of the thread is accessed. The rest of the thread will run from a more general category to a more specific category.

2.4.2 Distance Measure between Words

This particular representation provides a means for finding similarities between words by measuring the edit distance between threads. In this model, I use this representation when calculating how similar two objects are. One can simply measure the edit distance. For example, suppose we had the word "bird" and the word "human". The two threads might look like:

bird → living → animal

human → living → animal → primate

In order to find the edit distance between the two, one would find the part of the thread which is the same in both cases (living → animal for this example). From here, each remaining node (including the key) is considered as a difference (in this case bird, human, and primate are each a difference). Counting these gives us a measure of (dis)similarity between two words (three in this case).

2.5 Intermediate Features

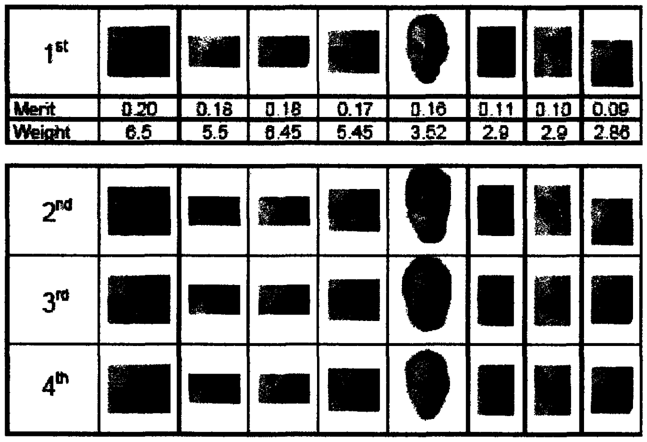
In the paper *Visual features of intermediate complexity and their use in classification*, Shimon Ullman, Michel Vidal-Naquet and Erez Sali demonstrated that it is features of intermediate complexity which are optimal for the basic visual task of classification. They showed that moderately complex features are more informative for classification than very simple or very complex ones (2002).

In their work, Ullman et al. take features of different sizes from objects they are trying to recognize. In order to recognize the objects Ullman makes use of the features represented as bitmaps, and attempts to find where it fits the best. These objects are either cars or faces. They take pictures of the objects, and break them down into the "simplest" subcomponents, such as the eyes, the nose, or the mouth, separately. When using each one of these subcomponents, it is not always easy to recognize the objects.

Then, they take the most complex features, which in the case of the faces would be the whole faces. These features did not provide enough information to recognize a picture as a face or car, as it becomes difficult either to align the faces or to measure all the differences.

The most informative features, as Ullman found, were those that were of intermediate size (in the face example used, these features would be sections of the face, such as the two eyes and brows, or the hairlines). These provided the most information and were the ones that most accurately categorized the faces.

In this thesis I make use of this theory, also known as the *Goldilocks' principle.* When comparing different elements, such as events, I only compare sections of the events. For example, when comparing trajectories, I separate out the trajectories into three different elements (intermediate features), the object, the reference, and the path. This provides enough information to represent the object in motion and the action of that object to get it in motion, while at the same time not making it too difficult to compute the differences between each of the features.



Merit	0.20	0.18	0.18	0.17	0.16	0.11	0.10	0.09
Weight	6.5	5.5	6.45	5.45	3.52	2.9	2.9	2.86

1st
2nd
3rd
4th

Figure 2.4 A table of Ullman's features used to categorize faces. The merit of each type of feature shows how informative that feature is. Note that the intermediate size features are more informative than complex features or very simple features (Ullman 2002).

2.6 Previous Work

Below I outline the different works which have explored the problem of relating two dimensions, as well as explored the idea of using a different representation to characterize events. Larson examined the idea of using SOMs in more than one domain in order to learn about the underlying world from which the measurements were taken. Tardiff extended the idea of using SOMs so that one can feed lexical structures, therefore being able to find similarities and differences between different events. I explain them in more detail in their respective sections.

2.6.1 Intrinsic Representation

David Larson's framework, introduced in *Intrinsic Representation: Bootstrapping Symbols from Experience*, learned from the world using SOMs. His world consisted of a mechanical arm and an eye which could manipulate and view blocks in their surroundings. The mechanical arm as well as the eye would collect data from its experience in the world, and would then feed this data to a SOM. He would have one map for what is sensed from the arm, such as its state (having the hand open or closed), its movement, and if it senses something. The other map would hold the information from what it was seeing. It would literally be a map of images. The eye could then look around and be able to know where it was looking (2003).

One of the most important things Larson showed was that there are certain characteristics that can be learned from using the different maps. First, he learned to coordinate the eye with the movement of the arm through statistical regularities detected between the two maps. Second, he showed a method which could build intrinsic symbols from the experience. Finally, his method was flexible. I extend this flexibility and show a method to perform the same type of relationships in a lexical space.

2.6.2 Self Organizing Event Maps

In Seth Tardiff's work, *Self Organizing Event Maps*, he showed that SOMs need not be numerical vectors. These can in fact be of any type of data type, just as long as the data has some sort of relative measure of similarity between two elements. He applies this to events represented in Jackendoff's trajectory frames. After the SOM is trained with a large amount of events observed, one can determine whether or not a new event is abnormal (2004).

In this thesis I put the two approaches together, and I show that Larson's intrinsic representations can be learned in different domains. It does not necessarily have to be those which are derived directly from our senses, but can also be other intermediate representations such as the one used by Tardiff.

3 Method

3.1 Overview of the Model

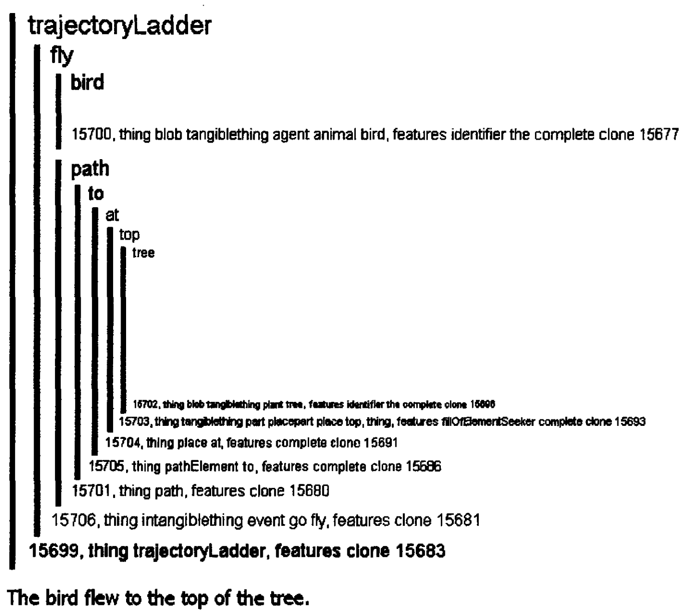
This model attempts to link representations of events based on two different modality inputs. The first modality is trajectory frames, which are a lexical description of spatial relations and motion along paths. For example, one could see two events, "the bird flew to the top of the tree" and "the bird flew from the water fountain." When translated into trajectory frames, they would look like Figure 3.1 and Figure 3.2. One could notice that in this representation the two are quite similar, consisting of an object (in this case *bird*), performing a type of movement (*fly*), through a path. Where the two are different are in the types of paths (one is *to*, the other is *from*) and the objects that create the path (*water-fountain* and *top of tree*). This gives us a basis for comparing the two elements in this modality.

The second modality used is transition spaces. These consist of descriptions of changes in some of the variables that are used to describe the events. Actual transition spaces could have a limitless number of variables, but since all of the events that I use to describe this model are events involving motion, I have decided to focus on two slots for each of the frames in the transition space:

1. The speed of the object in motion,
2. The distance between the object in motion and the reference object.

I also use three different frames that represent “moments in time.” These moments in time aren’t necessarily of the same size/length. They represent

1. The initiation of motion along the path,
2. The motion in progress, and
3. The finalization of motion or steady state after the event.



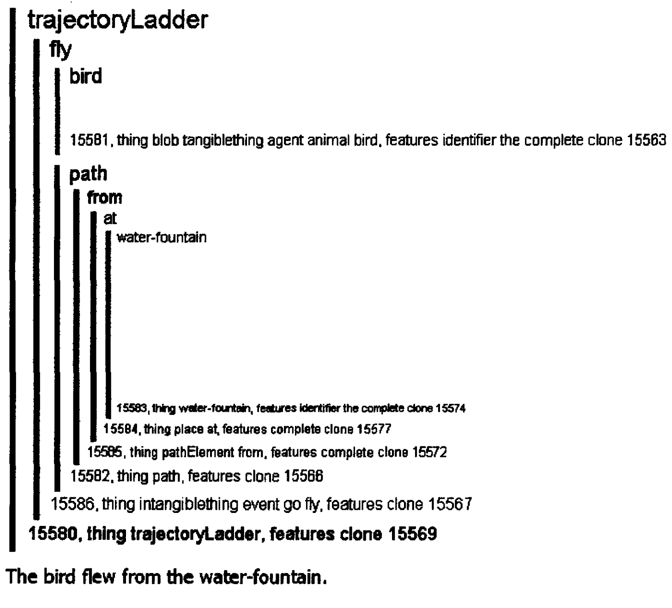
```
trajectoryLadder
 fly
  bird

  15700, thing blob tangiblething agent animal bird, features identifier the complete clone 15677
  path
   to
    at
     top
      tree

      15702, thing blob tangiblething plant tree, features identifier the complete clone 15696
     15703, thing tangiblething part placepart place top, thing, features fillOfElementSeeker complete clone 15693
    15704, thing place at, features complete clone 15691
   15705, thing pathElement to, features complete clone 15686
  15701, thing path, features clone 15680
 15706, thing intangiblething event go fly, features clone 15681
15699, thing trajectoryLadder, features clone 15683

The bird flew to the top of the tree.
```

Figure 3.1 A trajectory space for the sentence "The bird flew to the top of the tree" (figure taken from Winston 2006).



```
trajectoryLadder
 fly
  bird

  15581, thing blob tangiblething agent animal bird, features identifier the complete clone 15563
  path
   from
    at
     water-fountain

     15583, thing water-fountain, features identifier the complete clone 15574
    15584, thing place at, features complete clone 15577
   15585, thing pathElement from, features complete clone 15572
  15582, thing path, features clone 15566
 15586, thing intangiblething event go fly, features clone 15567
15580, thing trajectoryLadder, features clone 15569

The bird flew from the water-fountain.
```

Figure 3.2 A trajectory space for the sentence "The bird flew from the water-fountain" (figure taken from Winston 2006).

3.1.1 Circular Self Organizing Map

3.1.1.1 Description

The types of SOMs used are one-dimensional circular SOMs. This type of SOM was chosen for its simplicity, and for maintaining the dimensionality reduction necessary to extract relevant information and regularities out of the data that is fed into this SOM.

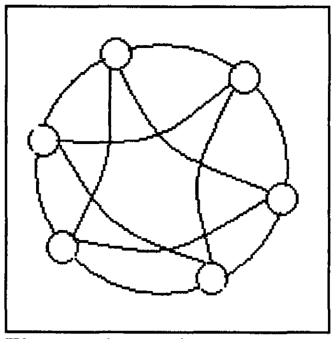


Figure 3.3 A 6-element 4-connect circular SOM. Note that each element is connected to 2 elements to each side of it.

These types of SOMs can have any number of elements. Each of the elements will have an even number of neighbors, half of them on one side of the element, the other half on the other side. A circular SOM in which each element is connected to 4 other elements would be called a 4-connect circular SOM. An example of a 4-connect SOM is given in Figure 3.3.

3.1.1.2 Clustering

The clustering algorithm used for the circular SOM is a greedy algorithm. The number of cuts performed is equal to the number of clusters desired. These cuts are made between adjacent elements which have the most stress, or are the most dissimilar.

Stress can be measured in a variety of ways. One possibility would be to use the simple dissimilarity between the two adjacent elements. While this is a good measure of the stress between the two clusters being formed in a 2-connect SOM, it fails to capture the stress of the links it would be disconnecting in higher order connected SOMs.

Another form of measuring the stress between two adjacent elements is the cross-sectional measure of stress, which is the form used in my implementation. This cross-sectional measure adds up the dissimilarities of all the links it would have to cut to separate two clusters between the adjacent elements. In a 4-connect SOM, there are 3 links which must be cut to create a separation between any two elements. This is pictured in Figure 3.4. Once all necessary cuts have been performed, the resulting connections define the clusters.

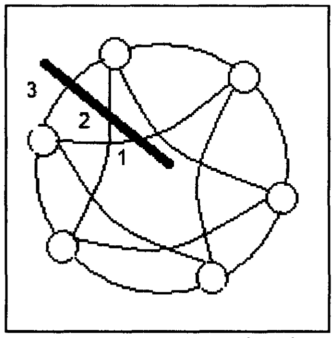


Figure 3.4 A cross-sectional cut (in red) on a 4-connect circular SOM.

3.1.2 Trajectory Frame Similarity

As mentioned before, one of the things needed is a measure of similarity between two trajectories. This is to relate two events when comparing them in a SOM. The similarity measure I use is a combination of the following:

1. The thread similarity measure discussed in section 2.4.2 of this thesis, applied to the object performing the action,
2. A measure of how similar the path functions of each of the events is, and
3. The thread similarity measured applied to the reference object.

I then take these three measures, and add them up to make up the trajectory frame similarity:

$$\boxed{dissimilarity = \Delta thread_{obj} + \Delta pathfunc + \Delta thread_{ref} + \Delta thread_{verb}}$$

Equation 3.1 Formula for dissimilarity of trajectory frames.

3.1.2.1 Thread Similarities

As mentioned in section 2.4.2, the thread similarities can be a function of the number of differences between the threads. In this case, I simply use the actual number of differences.

For example, using the two words human and bird, the threads would look like:

bird → living → animal

human → living → animal → primate

And their number of differences between them would be three. Therefore, the addition to dissimilarity would be a total of 3.

$$\boxed{\Delta thread = \#\ of\ edits}$$

Equation 3.2 Formula for dissimilarity between two words.

This same idea is also applied for verbs, where each verb has a thread associated to it describing the type of verb that it is.

3.1.2.2 Path Function Similarities

The first step in defining path function similarities was to recognize which are the possible path functions. The path functions that describe an action along a path were chosen to be one of three different functions:

1. To or towards,
2. From or away-form,
3. By or via.

The reason these three can be used to compare the movement along paths is because only one reference object is used. Whenever there are two objects, by definition there will exist a path between the two. These two objects are the performing object and the reference object.

The next step in defining the similarities is to assign a weight to each pair of path functions. This is done in Table 3.1 below:

	To	**From**	**Via**
To	0	2	1
From	2	0	1
Via	1	1	0

Table 3.1 Dissimilarity measure weights for pairs of path functions.

The resulting number is then added to the total dissimilarity.

$$\boxed{\Delta pathfunc = table\ lookup\ \ for\ \ dissimilarity}$$

Equation 3.3 Formula for dissimilarity between two path functions.

3.1.3 Transition Space Similarity

Besides a trajectory frame similarity measure, there also needs to be a transition space similarity measure. This transition space similarity measure is a mixture of similarities between each of the slots in each of the frames of the transition space. For each of the slots, I once again base the similarities on a pair-wise lookup of dissimilarity. The resulting dissimilarity is an average of the dissimilarities of each one of the slots.

$$\boxed{dissimilarity = \frac{\sum_{slots} \Delta slots}{\#\ of\ slots}}$$

Equation 3.4 Formula for dissimilarity of transition spaces.

As was done with the pair-wise dissimilarity of path functions, the pair-wise dissimilarity of slots is based on a table lookup. The measures are given in Table 3.2 below:

	A	Đ	↑	⤈	Ⱥ	Δ	⤉	↓	D	A̶
A	0	0.25	0.5	0.75	1	1.25	1.5	1.75	2	2.25
Đ	0.25	0	0.25	0.5	0.75	1	1.25	1.5	1.75	2
↑	0.5	0.25	0	0.25	0.5	0.75	1	1.25	1.5	1.75
⤈	0.75	0.5	0.25	0	0.25	0.5	0.75	1	1.25	1.5
Ⱥ	1	0.75	0.5	0.25	0	0.25	0.5	0.75	1	1.25
Δ	1.25	1	0.75	0.5	0.25	0	0.25	0.5	0.75	1
⤉	1.5	1.25	1	0.75	0.5	0.25	0	0.25	0.5	0.75
↓	1.75	1.5	1.25	1	0.75	0.5	0.25	0	0.25	0.5
D	2	1.75	1.5	1.25	1	0.75	0.5	0.25	0	0.25
A̶	2.25	2	1.75	1.5	1.25	1	0.75	0.5	0.25	0

Table 3.2 Dissimilarity measure weights for pairs of slots in a transition frame.

After matching up each respective slot with its counterpart in the other event, we then perform a lookup in the difference between the two types of changes.

$$\boxed{\Delta slots = table\ lookup\ for\ dissimilarity}$$

Equation 3.5 Formula for dissimilarity between two slots.

3.1.4 Blending Events in the Self Organizing Maps

In order to have a functional SOM, there needs to exist a method to blend two elements with each other. In this case, we need a way of getting an "in-between" point both between two trajectory frames and between two transition frames.

Blending occurs as a function of another parameter also given to the blending function, alpha. Alpha is going to tell us by how much the dissimilarity will be reduced starting at the first event going to the second. If alpha is equal to 50%, then the new event created from blending will be exactly in between the first even and the second event. In other words, the dissimilarity between the new event and one of the two original events will be the same, regardless of which event is picked.

3.1.4.1 Trajectory Blending

In order to blend two trajectory frames, each of the frame's slots; object, reference, verb and path function; is processed individually. In the case of the object slot, the reference slot and the verb slot we will find an in between point along the edit distance of the two threads. In the case of the path function, the alpha-weighted average distance from one of the two events is found by looking it up in Table 3.1, and then looking up the path function which is closest to this value.

For example, if we had the two events "the bird flew to the tree" and "the plane flew via the tree," the result of blending the two with alpha equal to 50% would be "the thing flew to the tree." From the four slots of the trajectories of the two events, notice that only two of them are different, the object (bird vs. plane) and the path function (to vs. via). The reason we get "thing" from blending "bird" and "plane" is because "thing" is exactly 3 edits away from both "bird" and "plane." The reason we get "to" from blending "to" and "via" is because the average distance from "to" of the two path functions is 0.5, which in this case gets rounded to 0. This produces a path function which has 0 distance from "to," which can only be "to" itself. Since the other two slots are the same in the two events, when blending them together they will remain the same.

3.1.4.2 Transition Blending

In order to blend two transition spaces, we also look at each of the six slots individually when mixing the two transitions. The way this is performed is by first assigning a number to each of the possible values of the slot. For transitions, all of the possible characters that can fill a slot, along with arbitrary numerical values assigned to them, are shown in Table 3.3.

A	~~D~~	↑	~~↓~~	~~Δ~~	Δ	~~↑~~	↓	D	~~A~~
1	2	3	4	5	6	7	8	9	10

Table 3.3 All possible values for a slot in a transition frame, along with their values.

After finding the assigned numerical values for the two characters being compared in a slot, we sum the two after weighing the reference event numerical value by (1 – alpha) and the value of the event to which it is being compared by alpha. This sum (along with appropriate rounding) will give us the numerical value of the character which will be used for that slot. This process is then repeated for each of the six slots that make up a transition space.

3.2 Implementation

The main objective of this thesis is to show that by extracting different salient features from the same events, one can relate two different domains. The circular SOM allows for salient features to be extracted in each of the two domains. Further clustering and relating of the clusters between maps can provide us with a means of discovering regularity among two different modes. In order to do this, I have implemented the following system.

This system was implemented using DrScheme with the Pretty Big language pack, which includes Mr. Ed and Advanced Student language modules. It is running on a Windows© machine that has an Intel™ Core Single processor and 512 MB of RAM. The system I have implemented is based on the following types of objects:

1. SOM objects,
2. thread objects,
3. thread-list object,
4. trajectory objects, and
5. transition objects.

3.2.1 Self Organizing Map Objects

The SOM objects are simple containers which hold the elements of the SOM. These elements are organized as they would be if they were fed into a circular SOM as described in section 3.1.1. It includes some methods which are useful in both training and evaluating the SOMs represented by this object.

When the SOM is initialized, it will create random trajectories or random transitions, depending on which type of SOM is being dealt with.

For example, if we wanted to create a 5-element SOM, where each element is connected to 2 neighbors, and it was made up of trajectories, we would run the following command:

```
> (define tsom (create-som 'trajectory 5 2))
```

This will initialize the SOM with random values.

If we wanted to feed it the sentence "The bird flew to the tree", we would first need to create a trajectory by putting the intermediate features as follows:

```
> (define traj1 (create-trajectory '((obj bird) (ref tree)
(verb fly) (pathfunc to))))
```

This system makes use of the object oriented code used in the course 6.001: Structure and Interpretation of Computer Programs. This code runs methods out of objects using the function `ask`.

```
> (ask tsom 'get-som)
```

The `GET-SOM` method is for display purposes, it maps a function on each of the elements that comprise the SOM so that they display their contents on screen.

```
(((obj duck) (ref bush) (verb hover) (pathfunc from))
 ((obj mallard) (ref field) (verb run) (pathfunc to))
 ((obj human) (ref field) (verb run) (pathfunc to))
 ((obj turtle) (ref box) (verb land) (pathfunc via))
 ((obj monkey) (ref drawers) (verb crawl) (pathfunc to)))
```

As we can see, this is a 5 element SOM with random trajectory elements. For example, the last line in that output would correspond to the sentence "The monkey crawled to the drawers."

From here, to make sure our SOM is working properly, we can feed it a single trajectory many times, using the `FEED-AND-UPDATE` method. This method trains the SOM one cycle with the object passed as the argument, such as a trajectory or a transition. Each time the SOM is trained with an input, it locates the most similar element, merges it with the input element. It then proceeds to merge the input element into the neighbors of the most similar element. In the end, we would expect all the elements of the SOM to slowly change into the trajectory we are feeding in[1]:

```
> (ask tsom 'feed-and-update traj1)

...

...

...

> (ask tsom 'feed-and-update traj1)

(((obj bird) (ref tree) (verb fly) (pathfunc to))

 ((obj bird) (ref tree) (verb fly) (pathfunc to))

 ((obj bird) (ref tree) (verb fly) (pathfunc to))

 ((obj bird) (ref tree) (verb fly) (pathfunc to))

 ((obj bird) (ref tree) (verb fly) (pathfunc to)))
```

[1] Nevertheless, it would not have to converge to a single-element SOM necessarily. It is possible that the SOM would reach a quasi-stable cycle where every update does not change the target SOM. There are several parameters involved, including and not limited to the alpha function, the number of connections, the method of breaking ties, and the method of merging two elements.

You'll notice that between monkey and bird it might say something like "living". This is because the implementation of trajectories is based on threads. I will now explore some properties of trajectory, transition and thread objects.

There are two other methods in the SOM object, `GET-CLUSTERS` and `GET-STRESS`. The `GET-CLUSTERS` method will return a list of clusters, and each cluster will be a list of indices to the elements of that cluster (the list will be a 1-based index list). The method `GET-STRESS` returns a list of stresses, where each stress is the stress in the intersection between the nth element and the nth-1 element (in the case of the first element, it would be the stress in the intersection between the first element and the last element, because of the circular nature of the list).

An example of their use is the following:

```
> (define tsom (create-som 'trajectory 5 2))
(((obj socks) (ref drawers) (verb crawl) (pathfunc via))
 ((obj monkey) (ref mountain) (verb jump) (pathfunc via))
 ((obj mallard) (ref bush) (verb fly) (pathfunc to))
 ((obj duck) (ref house) (verb crawl) (pathfunc via))
 ((obj human) (ref field) (verb jump) (pathfunc to)))
> (ask tsom 'get-clusters 2)
((5 1 2 3) (4))
> (ask tsom 'get-stress)
(20 19 23 15 17)
> (ask tsom 'get-clusters 3)
((2 3) (5 1) (4))
```

3.2.2 Threads

The threads are all created as objects, and there exists a thread for each verb or noun in the lexicon. This includes threads for superordinate things such as "living." Every time a thread is created, it is added to a global thread-list, which is another object. This thread-list can be considered our dictionary which is used to lookup the threads based on their keys [the words themselves].

Note that the thread objects are defined with the same format as described for threads in Vaina and Greenblatt. That is to say that the threads themselves do not contain the word being looked up [the key], only all the superclasses in topological order. For example, the thread for "duck" would be (thing living flying-living bird).

For example, if we knew that there existed the thread for "duck," we could access this thread by running the following command:

```
> (ask thread-list 'get 'duck)
(instance #<procedure:handler>)
```

It returns the object representing the thread for duck. We can then ask this object for its contents, as follows:

```
> (ask (ask thread-list 'get 'duck) 'get-key)
duck
> (ask (ask thread-list 'get 'duck) 'get-thread)
(thing living flying-living bird)
```

If we have two different thread-objects, we can then compare the two.

```
> (define jet-thread (ask thread-list 'get 'jet))
> (define drawer-thread (ask thread-list 'get 'drawers))
> (ask jet-thread 'get-thread)
(thing inanimate flying-inanimate plane)
> (ask drawer-thread 'get-thread)
(thing inanimate box)
> (ask jet-thread 'compare-to drawer-thread)
5
```

This shows us that the edit distance along threads from the word jet to the word box is equal to 5 (the five edits are: delete jet, delete plane, delete flying-inanimate, add box, add drawers).

Moreover, this system allows for two threads to be alpha-blended. What I mean by alpha-blending two threads is that it will move away from the original word and closer to the comparison word along the edit distance by alpha percentage. In other words, if there were 5 edits between "jet" and "drawers", and alpha was equal to 50%, it would perform 2 or 3 of the edits, and return the resulting key. Whenever the alpha-factor multiplied by the edit distance results in a fractional number (2.5 in this case), half the time it will round up, the other half it will round down.

```
> (ask jet-thread 'merge-with drawer-thread 0.5)
inanimate
```

This does not affect the thread objects at all, only produces a new symbol from their threads.

3.2.3 Transition and Trajectory objects

Transitions and Trajectories are conformed by objects in this system. They have homologous methods which allow the comparison and combination of two or more objects. One can call the method `COMPARE-TO` and give it another object of the same type, and it will measure the distance between the two objects. Besides the `COMPARE-TO` method, transitions and trajectories count with a few other methods necessary for its proper function and installation into the system. The method `GET-SLOTS` simply returns the list structure of the particular type of the object. The method `SET-SLOTS` lets a user destructively change the slots the object is holding and replace them with new slots as given by the argument passed to the method.

```
> (define traj1 (create-trajectory '((obj bird) (ref tree)
(verb fly) (pathfunc to))))
```

This creates a trajectory for "The bird flew to the tree," same as before.

```
> (define traj2 (create-trajectory '((obj plane) (ref tree)
(verb fly) (pathfunc via))))
```

This creates a trajectory for "The plane flew via the tree."

```
> (ask traj1 'compare-to traj2)
7
```

This measure is derived from the fact that plane is an edit distance of 6 from bird, and "via" is dissimilar to "to" by 1.

Unlike threads, whenever we call the `MERGE-WITH` method of one object onto another, the first object is destructively replaced with the results, as shown by the following sequence of commands:

```
> (ask traj1 'get-slots)
((obj bird) (ref tree) (verb fly) (pathfunc to))
> (ask traj1 'merge-with traj2 0.5) ;; it uses alpha-
```

blending just as threads do.

```
> (ask traj1 'get-slots)
```

```
((obj thing) (ref tree) (verb fly) (pathfunc to))
```

3.2.4 Clustering

One of the most difficult tasks of implementing this system is implementing the clustering algorithm. While conceptually it is simple to understand, there are many more design choices involved than seems apparent. One of the most important of these design choices is figuring out the number of clusters for each of the types of SOMs.

3.2.4.1 Number of Clusters

In order to figure out the number of clusters that would be used I adapted a form of Cattell's scree test. The scree test involves plotting some sort of variance measure against the factors for which one has to decide a number to use (1966).

The idea is that the plot would decay in exponential form, such that it would have the greatest reduction of variance for the first numbers of factors, and would then have a diminishing reduction in the variance for each additional factor. What this means is that after one reaches a certain number of clusters, if one tries to add more, it will not reduce the variance of the cluster sizes significantly. This method of deciding number of factors can be performed visually, by inspecting a graph that decays in an exponential fashion and picking out the point where "the elbow bends."

The variance, in this case, would be the standard deviations of the sizes of the different clusters created, represented as a percentage of the size of the largest cluster created (see Equation 3.6). The reason we want to decrease this variance is because we would like to have the sizes of the clusters be as close together as possible, and avoid having some incredibly large clusters and some really small clusters.

$$\boxed{\text{variance measure} = \frac{stdev(\textit{sizes of clusters})}{\max(\textit{sizes of clusters})}}$$

Equation 3.6 Variance measure for number of clusters.

Using this method of calculating the variance, I plotted several clusterings of SOMs of 25, 50, 75 and 100 elements. These circular SOMs were all 4-connected. What I discovered from plotting these both for trajectory-based SOMs and for transition-based maps was that regardless of the size of the map (as long as it was a reasonable size) we would need around 6 clusters, as determined by the scree test[2].

[2] The scree test is so called from "the analogy of the steep descent of a mountain till one comes to the scree of rubble at the foot of it." (Cattell 1966)

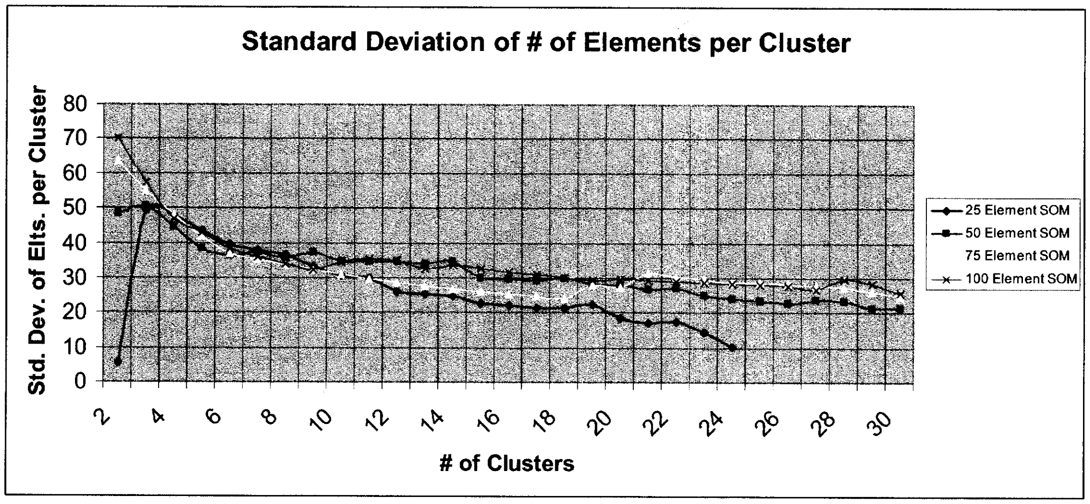


Figure 3.5 Scree plot to determine number of clusters for a trajectory SOM.

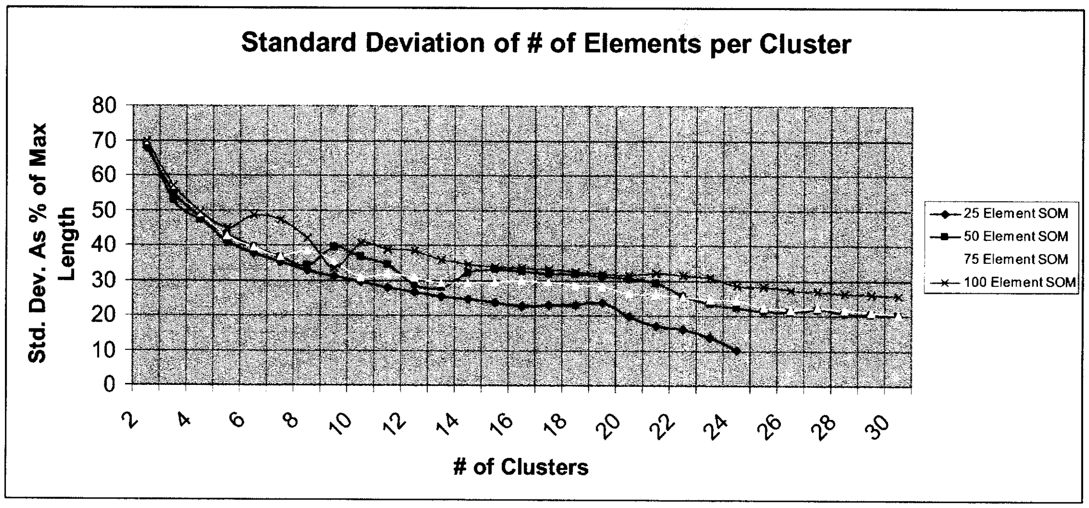


Figure 3.6 Scree plot to determine number of clusters for a transition SOM.

3.2.4.2 Mapping Clusters between Maps

After clustering the trajectory SOM and the transition SOM, the number of times a cluster from one of the two maps links to a cluster in the other map is evaluated.

Each of the maps is trained with the same sentences with a data set of 600 sentences selected at random from a possible 1914 sentences created by combining the different objects, references, verbs and path functions in combinations that would be plausible in the real world. These maps are then clustered. Finally a set of 300 test sentences not included in the original data set of 600 is selected. This set is then used to link the two SOMs. The number of times a cluster maps to another is recorded.

From the links created, two types of measures are recorded in order to discern the effectiveness of the relations derived. The two ratios used as measures are coverage, and consistency.

3.2.4.2.1 Consistency and Coverage Ratios

Consistency deals with each individual cluster in one of the SOMs. It measures the percentage of links from this cluster that connect to the predominant cluster in the other SOM. If, for example, we had a cluster in the trajectory SOM that links to one cluster in the transition SOM 30 times, to another cluster 20 times, and to a final cluster 10 times, then its consistency ratio would be 50%, because the number of links to the predominant cluster [30] would be half of the total number of links that originate in the trajectory cluster [that is, 30 + 20 + 10 = 60].

Coverage deals with the SOMs as a whole. It measures the percentage of the clusters in a SOM that are covered by associations from clusters in the other SOM. For example, if the 6 clusters from the transition SOM between them only connect to 4 clusters of the 6 in a trajectory SOM (because there are collisions when mapping from transition clusters to trajectory clusters), then the coverage ratio would be 66%.

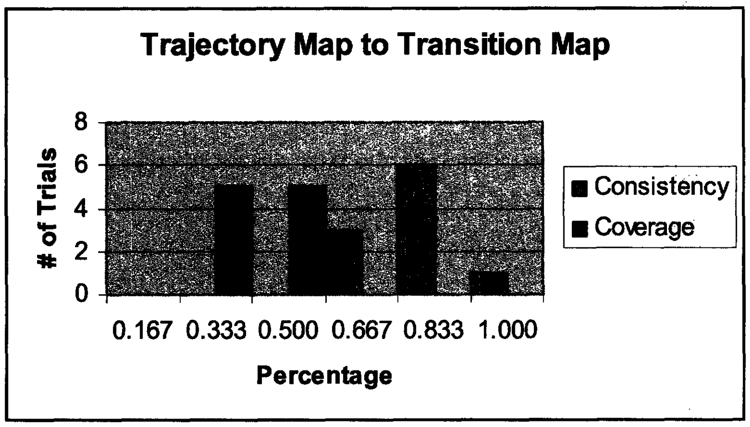


Figure 3.7 Histogram of consistency and coverage ratios for 10 trials when mapping from trajectories to transitions.

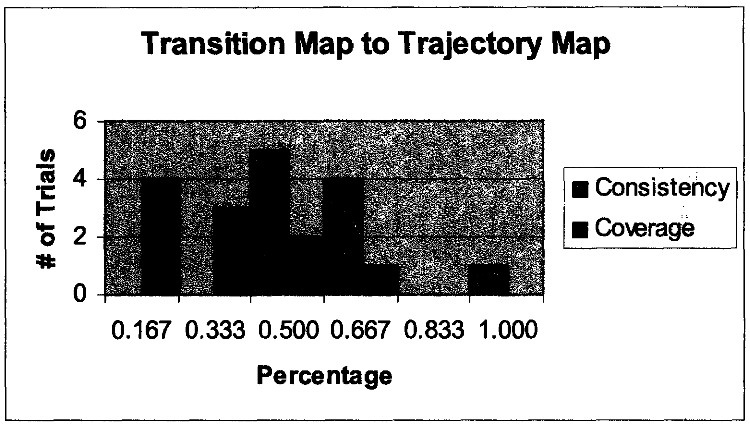


Figure 3.8 Histogram of consistency and coverage ratios for 10 trials when mapping from transitions to trajectories.

3.2.4.2.2 Examples of Cluster Mappings

The following is an example of the output of the clusters in a trajectory map and a transition map. This particular output represents the indices in the list of elements of each map. For example, the 33rd through the 47th elements in the trajectory map's list of elements would make up the first cluster of trajectories. There would be two clusters of a single element each, the 32nd element and the 48th element. Likewise in the transitions you would have a single cluster that would hold the 28th through the 93rd elements in the list of transitions stored in the transition map.

Trajectory Frame Clusters:

```
((33 34 35 36 37 38 39 40 41 42 43 44 45 46 47)
 (32)
 (78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
  96 97 98 99 100 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  17 18 19 20 21 22 23 24 25 26 27 28 29 30 31)
 (48)
 (49 50 51 52 53 54 55 56 57 58 59)
 (60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77))
```

Transition Space Clusters:

```
((96)
 (28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
  46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
  64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
  82 83 84 85 86 87 88 89 90 91 92 93)
 (97 98 99 100 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
  18 19 20 21 22 23 24 25 26)
 (27)
 (95)
 (94))
```

After feeding 300 random sentences into these maps and these clusters, the following table displays the number of links found between the clusters:

	Transition Cluster 1	**Transition Cluster 2**	**Transition Cluster 3**	**Transition Cluster 4**	**Transition Cluster 5**	**Transition Cluster 6**
Trajectory Cluster 1	0	0	6	1	0	1
Trajectory Cluster 2	0	0	0	0	0	0
Trajectory Cluster 3	0	0	77	102	0	93
Trajectory Cluster 4	0	0	0	0	0	0
Trajectory Cluster 5	0	0	4	1	0	2
Trajectory Cluster 6	0	0	3	2	0	8

Table 3.4 Example number of links between transition clusters and trajectory clusters.

From Table 3.4, we get that the consistency from trajectory clusters to transition clusters is equal to 58%, from averaging the consistency of each of the clusters which have some events mapped to it. The coverage of transition clusters from the trajectory clusters is 50%, as we can see the Trajectory Clusters 1 and 6 would map to Transition Cluster 3, Trajectory Cluster 3 to Transition Cluster 4, and finally Trajectory Cluster 5 to Transition Cluster 6. On the flip side, the consistency from transition clusters to trajectory clusters is 90%, but the coverage of trajectory clusters is only 17%.

The following is an example mapping from a trajectory cluster to a transition cluster, where we can see the contents of the clusters.

Trajectory Cluster 1

```
 (((obj bird) (ref object) (verb fly) (pathfunc via))
  ((obj walking) (ref place) (verb motion) (pathfunc via))
  ((obj walking) (ref air) (verb motion) (pathfunc via))
  ((obj grounded-living) (ref thing)
   (verb active-ground-motion) (pathfunc via))
  ((obj turtle) (ref air) (verb slide) (pathfunc from))
  ((obj living) (ref place) (verb motion) (pathfunc via))
  ((obj grounded-living) (ref place) (verb motion)
   (pathfunc from))
  ((obj turtle) (ref house) (verb run) (pathfunc via))
  ((obj grounded-living) (ref inanimate) (verb verb)
   (pathfunc via))
  ((obj living) (ref house) (verb verb) (pathfunc via))
  ((obj bee) (ref air) (verb hover) (pathfunc via))
  ((obj living) (ref air) (verb multipod-motion)
   (pathfunc via))
  ((obj flying-living) (ref thing)
   (verb active-ground-motion) (pathfunc via))
  ((obj flying-living) (ref place) (verb hover)
   (pathfunc via))
  ((obj clothes) (ref place) (verb motion)
   (pathfunc from)))
```

As we can see, this particular cluster of trajectories mostly involves grounded living things which move via some object. This cluster maps to the third transition cluster:

Transition Cluster 3

```
(((dec app) (chg inc) (inc ndec))
 ((dec app) (chg inc) (inc ndis))
 ((dec app) (nchg inc) (inc inc))
 ((dec app) (nchg inc) (inc ndis))
 ((dec app) (nchg ndis) (inc inc))
 ((dec app) (nchg inc) (inc ndis))
 ((ninc app) (nchg inc) (inc ndis))
 ((dec app) (ndec inc) (inc inc))
 ((dec app) (ndec ndis) (ndis inc))
 ((dec app) (nchg inc) (inc inc))
 ((ninc app) (nchg ndis) (ndis inc))
 ((ninc app) (nchg inc) (inc inc))
 ((dec app) (nchg inc) (inc inc))
 ((dec app) (nchg inc) (inc inc))
 ((dec app) (nchg ndis) (ndis ndis))
 ((dec app) (nchg inc) (inc inc))
 ((dec app) (nchg inc) (ndis ndis))
 ((ninc app) (nchg ndis) (ndis ndis))
 ((ninc app) (nchg ndis) (ndis inc))
 ((chg app) (ndec ndis) (inc inc))
 ((ninc app) (nchg inc) (inc ndec))
 ((dec app) (nchg inc) (ndis ndis))
 ((dec app) (nchg inc) (inc inc))
 ((dec app) (nchg inc) (ndis ndis))
 ((dec app) (nchg ndis) (ndis inc))
 ((dec app) (nchg inc) (inc inc))
 ((dec app) (nchg inc) (inc ndis))
 ((dec app) (chg inc) (inc inc))
 ((ninc app) (ndec inc) (inc ndec))
 ((ninc app) (ndec inc) (inc ndis)))
```

We can see that this transition cluster mostly contains a homogeneous type of transition spaces. All of them involve the speed (second element of each frame) first appearing, and then either increasing or not disappearing. The distance either decreases or does not increase, then does not change, and finally increases or does not decrease. These types of motion seem to correspond to the path function "via."

4 Discussion

4.1 Results

The method described provides a means for relating the clusters from two different SOMs. As we can see from the results, when going from the clusters in the trajectory map to clusters in the transition map the consistency is relatively high, but the coverage is low (almost always below 50%), as it turns out most clusters from a trajectory SOM map the same cluster in a transition SOM. When going from the clusters in the transition map to clusters in the trajectory map, the coverage is usually better, unless we end up with a very large cluster in the trajectory map that tends to attract most of the links from the transition map clusters.

The reasons for these lopsided results have to do with two factors, the data that I am feeding in and the distance metric used for transitions. While trajectory frame SOMs are slightly more homogenous in terms of the sizes of its clusters, transition frame SOMs tend to have a much larger cluster in the end, most probably due to the lack of variation of the input transitions coupled with a distance metric that does not provide much variation, either. Nevertheless, some of these deficiencies could be corrected with an improved representation of transitions.

When talking about intermediate representations, it is most likely that the representation of transitions for the purposes of this project were too simple. Moreover, the methods to compare and merge did not take into account possible inconsistencies in a transition. For example, a distance could possibly disappear and then increase.

While the results from using these two particular domains, trajectory frames and transition spaces, did not provide conclusive evidence, this framework does allow for further work.

4.2 Further Work

The work described in this thesis provides a framework that can be built upon. There are some improvements that could be made, and it opens more doors to further research characteristics between domains.

One of the improvements, as previously mentioned, is using more complex transition spaces. These transition spaces only had two variables and three different frames inside the spaces. This would seem like it would be enough to describe the motion of an object along a given path, nevertheless it seems like more complex features would be necessary in order to find distinguishing features in motion events.

Another possibility to improve would be to improve the distance metric between transitions. The straight-line averaging of the six slots across the three frames on two variables does not seem to capture all the necessary information to distinguish between different types of motion[3].

There is also the possibility of implementing a different type of clustering algorithm. The constraint imposed on this system, which is to have consecutive elements of a circular SOM be part of a cluster, may introduce more variance into the clusters and to what other clusters they map to. This might make each of the clusters less consistent.

[3] It may be possible to avoid discontinuities along different frames for a single variable by performing alpha-blending across frames. Of course, the distance metric would have to change for each possible value of the slot so that the most distant is the most discontinuous when given a particular value. Further work could demonstrate if this is viable.

Instead, one possibility might be performing a greedy algorithm for removing links until you end up with the desired number of disjoint sets or clusters in an SOM.

Michael Coen has proposed some interesting methods of performing clustering based on information provided in two domains (Coen 2005). It may be possible to adapt this type of clustering using the circular SOMs. It may also be interesting to see if this clustering technique could be applied to two dimensional SOMs trained with the same data.

5 Contributions

In this thesis I have contributed the following:

- A **method for relating two domains** based on training them from same events,
- Provided an **intermediate representation** of trajectories which allows us to extract salient information,
- Developed a **method to merge threads, trajectories and transitions,** based on their respective distance metrics,
- Presented the **circular self organizing map**, which is a one dimensional map where the neighborhood of each node are the most nearby cells along the dimension, and
- Adapted the **scree test** in order to find the optimal number of clusters in a circular SOM.

Bibliography

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

Raymond B. Cattell. The scree test for the number of factors. *Multivariate Behavior.* Res. 1:245-76, University of Illinois, Urbana-Champaign, Illinois, 1966.

Michael Coen. Cross-Modal Clustering. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (AAAI 05), pp. 932-937. Pittsburg, PA. 2005.

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.

Elisabeth S. Spelke. *Principles of object perception.* Cognitive Science, 14(1):29-56, 1990.

Seth Tardiff. *Self-Organizing Event Maps.* Master's thesis, Massachusetts Institute of Technology, 2004.

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. *Representationally Complete Analogical-reasoning Systems And Steps toward their Application in Political Science*. Grant proposal, MIT Computer Science and Artificial Intelligence Laboratory, 2006.

Appendix A Thread Definitions

```
(define objects-list '(pigeon duck mallard moth bee human
monkey spider turtle car plane jet socks))
(define refobj-list '(plant tree bush box drawers shop
house mountain field air water-fountain))
(define verb-list '(fly walk run crawl hover propell slide
jump land))


(define fly-n-list '(pigeon duck mallard moth bee plane
jet))
(define ground-n-list '(human monkey spider turtle car
socks))
(define fly-v-list '(fly hover propell land))
(define ground-v-list '(walk run crawl slide jump))
(define pathfunc-list '(to via from))


(create-thread 'thing '() 'noun)
(create-thread 'living '(thing) 'noun)
(create-thread 'inanimate '(thing) 'noun)


;; LIVING
(create-thread 'flying-living '(thing living) 'noun)
(create-thread 'grounded-living '(thing living) 'noun)


(create-thread 'bird '(thing living flying-living) 'noun)
(create-thread 'flying-insect '(thing living flying-living)
'noun)
(create-thread 'pigeon '(thing living flying-living bird)
'noun)
(create-thread 'duck '(thing living flying-living bird)
'noun)
(create-thread 'mallard '(thing living flying-living bird
duck) 'noun)
(create-thread 'moth '(thing living flying-living flying-
insect) 'noun)
```

```
(create-thread 'bee '(thing living flying-living flying-
insect) 'noun)


(create-thread 'walking '(thing living grounded-living)
'noun)
(create-thread 'insect '(thing living grounded-living
walking) 'noun)
(create-thread 'primate '(thing living grounded-living
walking) 'noun)
(create-thread 'human '(thing living grounded-living
walking primate) 'noun)
(create-thread 'monkey '(thing living grounded-living
walking primate) 'noun)
(create-thread 'spider '(thing living grounded-living
walking insect) 'noun)
(create-thread 'turtle '(thing living grounded-living
walking) 'noun)


;; INANIMATE
(create-thread 'flying-inanimate '(thing inanimate) 'noun)
(create-thread 'grounded-inanimate '(thing inanimate)
'noun)


(create-thread 'car '(thing inanimate grounded-inanimate)
'noun)
(create-thread 'plane '(thing inanimate flying-inanimate)
'noun)
(create-thread 'jet '(thing inanimate flying-inanimate
plane) 'noun)


(create-thread 'clothes '(thing inanimate) 'noun)
(create-thread 'socks '(thing inanimate clothes) 'noun)


;; LOCATIONS
(create-thread 'unmovable-living '(thing living) 'noun)
(create-thread 'plant '(thing living unmovable-living)
'noun)
```

```
(create-thread 'tree '(thing living unmovable-living plant)
'noun)

(create-thread 'bush '(thing living unmovable-living plant)
'noun)


(create-thread 'place '(thing inanimate) 'noun)

(create-thread 'box '(thing inanimate) 'noun)

(create-thread 'drawers '(thing inanimate box) 'noun)

(create-thread 'shop '(thing inanimate place) 'noun)

(create-thread 'mountain '(thing inanimate place) 'noun)

(create-thread 'house '(thing inanimate place) 'noun)

(create-thread 'field '(thing inanimate place) 'noun)

(create-thread 'air '(thing inanimate place) 'noun)


(create-thread 'object '(thing inanimate) 'noun)

(create-thread 'water-fountain '(thing inanimate object)
'noun)


;; VERBS
(create-thread 'verb '() 'verb)
(create-thread 'motion '(verb) 'verb)
(create-thread 'action '(verb) 'verb)

(create-thread 'active-air-motion '(verb motion) 'verb)
(create-thread 'active-ground-motion '(verb motion) 'verb)
(create-thread 'winged-motion '(verb motion active-air-
motion) 'verb)
(create-thread 'bipod-motion '(verb motion active-ground-
motion) 'verb)
(create-thread 'multipod-motion '(verb motion active-
ground-motion) 'verb)

(create-thread 'fly '(verb motion active-air-motion winged-
motion) 'verb)
(create-thread 'walk '(verb motion active-ground-motion
bipod-motion) 'verb)
(create-thread 'run '(verb motion active-ground-motion
bipod-motion) 'verb)
(create-thread 'crawl '(verb motion active-ground-motion
multipod-motion) 'verb)
(create-thread 'inactive-air-motion '(verb motion) 'verb)
```

```
(create-thread 'inactive-ground-motion '(verb motion)
'verb)
(create-thread 'hover '(verb motion inactive-air-motion)
'verb)
(create-thread 'propell '(verb motion inactive-air-motion)
'verb)
(create-thread 'slide '(verb motion inactive-ground-motion)
'verb)

(create-thread 'action '(verb) 'verb)
(create-thread 'ground-to-air-action '(verb action) 'verb)
(create-thread 'air-to-ground-action '(verb action) 'verb)
(create-thread 'jump '(verb action ground-to-air-action)
'verb)
(create-thread 'land '(verb action air-to-ground-action)
'verb)
(display "Successfully loaded threads...\n")
```

Appendix B Example Sentences

Sentence: The bird flew to the top of the tree.
Trajectory: ((obj bird) (ref tree) (verb fly) (pathfunc to))
Transition: ((dec app) (dec inc) (dis nchg))

Sentence: The car ran by the house.
Trajectory: ((obj car) (ref house) (verb run) (pathfunc via))
Transition:B ((dec app) (nchg inc) (inc inc))

Sentence: The human jumped from a mountain.
Trajectory: ((obj human) (ref mountain) (verb jump) (pathfunc from))
Transition: ((app app) (inc inc) (inc inc))