Mining propositional dynamics from sensory input

Paul R. Cohen[*] **Wesley Kerr**[*] **Yu-Han Chang**[**]

[*]Department of Computer Science
University of Arizona
Tucson, AZ
{cohen,wkerr}@cs.arizona.edu

[**]Information Sciences Institute
University of Southern California
Marina Del Rey, CA
ychang@isi.edu

Abstract

Although dynamics are often represented by real-valued series, some domains are best represented propositionally, and their dynamics are to be found in the dependencies between propositions over time. This paper shows how to mine these dynamics.

1. Introduction

It is common to describe states as sets of logical assertions and to model dynamics in terms of probabilities of state transitions. However, state transition models such as Markov chains or Markov Decision Processes are not concerned with changes in the internal structure of states. Consider six propositions, shown below, that are true over intervals represented as horizontal lines:

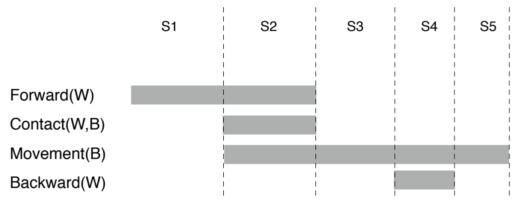

Figure 1: Four propositions and their truth values over time.

One could say there are five states, in which different subsets of the propositions are true (e.g., $S_1 = \{Forward(W)\}$) and represent the sequence above as $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5$; or one could represent the sequence in terms of patterns of temporal relationships between propositions.

By *mining propositional dynamics* we mean discovering patterns of changes in the truth values of propositions over time. Finding non-accidental patterns of changes in relational structure (and discarding changes that are probably accidental) is the first step in causal modeling of developmental robotics.

Other examples include changes in the structure of food networks (predator-prey relationships), or the disposition of units in adversarial games, or the relationships between characters in stories. In all these examples, relational structure changes over time, not accidentally, but causally.

2. Representing Sensor Dynamics

The examples and experiments reported in this paper are taken from a game called Wubble World, in which softbots' interactions with objects are governed by simulated physics (Hewlett et al., 2007, Kerr et al., 2007). A very simple example is shown in Figure 1, which shows a wubble (W) moving toward a block (B), then making contact with B, after which B starts to move, then breaking contact with B, at which point W backs up and eventually B stops moving.

It is easy to model the dynamics of an individual proposition, such as `Contact(W,B)`, with a two-state finite state machine. What we really want to know, however, is whether the dynamics of a proposition depends on the dynamics of others. The following sections describe how we represent dependencies between the dynamics of pairs of propositions (Sec. 2.2), and how we test whether pairs of propositions change together more often than would be expected by chance (Sec. 3.). Then we generalize the representation of dependencies and the testing method to dependencies between n-tuples of propositions. The inherent combinatorial cost of generating and testing n-tuples is checked by an incremental procedure that "seeds" the search for dependent n-tuples with pairs of propositions.

2.1 Bit Arrays and Compressed Bit Arrays

The propositional data in Figure 1 can be represented as a *bit array* in which each row represents a logical proposition, each column represents a moment in time, and each cell contains 1 or 0 depending on whether the corresponding proposition is true or false at that moment:

```
Forward(W)      111111110000000000
Contact(W,B)    000011110000000000
Movement(B)     000011111111111111
Backward(B)     000000000000111100
```

A related data structure, the *compressed bit array* (CBA), provides a canonical representation of complex patterns of sensor dynamics. If we will be satisfied with an *ordinal* time scale for sensor dynamics — a scale that preserves the order of changes to a proposition's status but not the durations of intervals — then we can compress the bit array by removing identical consecutive columns, as shown in Figure 2. The purpose of this compression is less to save space than to produce an abstraction of the bit array in which patterns of changes that are identical but for their durations are represented identically.

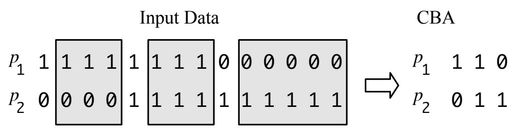
Input Data
CBA
p_1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
p_2 0 0 0 0 1 1 1 1 1 1 1 1 1 1
p_1 1 1 0
p_2 0 1 1

Figure 2: Discarding all but one of identical columns (shown shaded) in a bit array produces a compressed bit array (CBA).

The CBA corresponding to the dynamics in Figure 1 is:

```
Forward(W)      11000
Contact(W,B)    01000
Movement(B)     01111
Backward(W)     00010
```

Note that it has five columns, corresponding to the five "phases" shown in the figure.

2.2 *Representing Pairs of Changes*

The bit arrays we work with are long, representing thousands of time ticks in the agent's sensed world. In such data we can find statistical dependencies between changes. However, the bit array itself does not express whether one set of changes depends on another. In the following paragraphs we develop an explicit representation of dependencies between changes to propositional state.

How should we express dependencies between intervals? The answer to this question will determine how to gather statistical evidence for dependencies.

There are seven qualitative ordering relationships between two intervals: the well-known *Allen Relations*, shown in Table 1 (Allen, 1983). A single instance of an Allen relation between propositions might be accidental, but in a long sequence of changes to graphs we might find statistical dependence between the elements of the relation; for example, we might find that `Move(B,loc2)` and `Contact(W,B)` are related by *finishes* more often than would be expected by chance.

CBA	Allen	Meaning
x (1) y (1)	*equal*(x,y)	x starts and ends with y
x (10) y (01)	*meets*(x,y)	x ends and y starts
x (11) y (01)	*finishes*(x,y)	y starts after and ends with x
x (10) y (11)	*starts*(x,y)	x starts with y and finishes before y finishes
x (010) y (111)	*during*(x,y)	x starts after and finishes before y
x (110) y (011)	*overlaps*(x,y)	y starts after x starts and finishes after x finishes
x (100) y (001)	*before*(x,y)	y starts after x finishes with small delay

Table 1: Allen Relations

3. Testing for Dependent Pairs of Changes

Suppose we have a bit array of length N in which two propositions, p_i and p_j, are related c times by an Allen relation R. We denote this $Count(R(p_i, p_j)) = c$. The question is whether p_i and p_j are *really* related by R or whether the c occurrences of $R(p_i, p_j)$ are accidental. If c is much bigger or smaller than expected under a model of chance occurrence of $R(p_i, p_j)$, then we're inclined to say p_i and p_j are related by R — their co-occurrence in relation R is not accidental. For example, traffic lights are red or green for alternating intervals, and cars move or sit stationary, and the question is whether the changes between these intervals are independent. Figure 3 shows intervals during which a traffic light is red (labelled 2,4,6, and 8) and intervals during which traffic is stopped (labelled B, D, F, H and J). The most common relation between these intervals is *overlaps*(`Light`,`Stop`); it occurs four times. The question is whether *overlaps*(`Light`,`Stop`) is a real relationship or an accidental one.

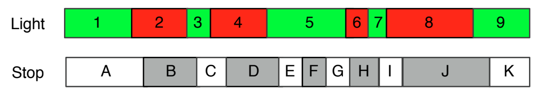

Figure 3: The proposition `Light` = *"the light is red"* is true in intervals 2, 4, 6 and 8. The proposition `Stop` = *"the cars are stopped"* is true in intervals B, D, F, H, and J. The Allen relation *overlaps*(`Light`, `Stop`) occurs four times.

We will tend to believe $R(p_i, p_j)$ is real if c is much larger or much smaller than expected under a model of chance occurrences of $R(p_i, p_j)$. Such a model should estimate c under the assumption that the locations of intervals of p_i and p_j are independent. Said differently, such a model should assume that whichever processes cause p_i and p_j to be true, these processes don't influence each other, nor does another process influence both of them. One approach might be to model the independent processes that generate intervals in which p_i and p_j are true, and then derive from these models expectations of c. A simpler and probably more robust approach is to estimate c by randomization.

The intuition behind randomization is that the configuration of elements in a particular sample might have been different under random sampling, so even if one doesn't know the parameters of the population from which the sample was drawn, one can mimic the process of drawing other samples by changing the sample configuration at random. In this way, one can get a distribution of sample statistics (called a randomized sampling distribution) that can be used to test hypotheses (Cohen, 1995). A particularly easy kind of hypothesis to test with randomization is that two (or more) configurations are independent.

Suppose the bit pattern in Figure 3 is our sample, and $c = Count(overlaps(\texttt{Light}, \texttt{Stop})) = 4$ is our sample statistic. To find the distribution of c under the hypothesis that `Stop` and `Light` are independent, we mimic the process of drawing other samples, as shown in Figure 4.

A *pseudosample* is constructed by shuffling the intervals in `Light` and `Stop` independently, subject to the constraint that intervals in which a proposition is true must alternate with intervals in which it is false.[1] Because the intervals in `Light` and `Stop` are shuffled independently, any Allen relations that are observed between these intervals are accidental. Each pseudosample yields a single value of $c^* = Count(overlaps(\texttt{Light}, \texttt{Stop}))$. (The asterisk denotes that c^* is not the sample statistic, c, but a statistic derived from a pseudosample.) For instance, the peudosample in Figure 4 yields $c^* = 1$. Repeating the process k times yields a randomized sampling distribution of c^*.

Because the dynamics of `Light` and `Stop` are preserved by randomization (the sizes of intervals during which these propositions are true and false are not altered) the randomized sampling distribution of c^* also estimates probabilities of c.

Let m denote the number of values of c^* greater than or equal to c, and let k be the number of pseudosamples that went into the distribution of c^*. Then $p = m/k$ is an estimate of the probability of getting at least c occurrences of $overlaps(\texttt{Light}, \texttt{Stop})$. If p is very small, then we will be inclined to believe that occurrences of $overlaps(\texttt{Light}, \texttt{Stop})$ are too numerous to be accidental. Interestingly, if $1 - p$ is very small, then we will be inclined to believe that *non*-instances of $overlaps(\texttt{Light}, \texttt{Stop})$ are not accidental. Said differently, very small values of $1 - p$ indicate fewer instances than one would expect by chance.

[1]The reason for this constraint is that two consecutive "true" (or "false") intervals will have a length not observed in the original sample, which amounts to changing one's model of the dynamics of the proposition.

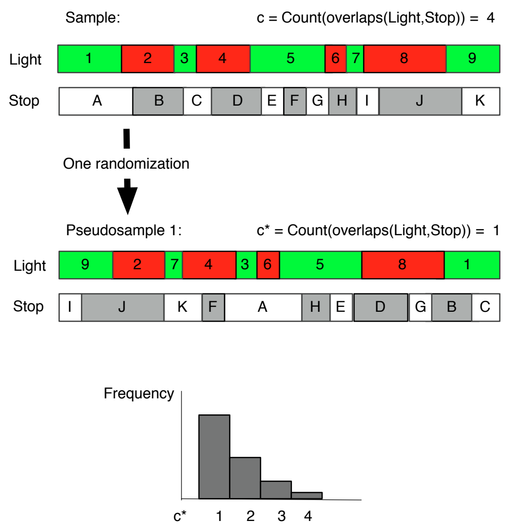

Figure 4: The sample configuration is randomized by shuffling the intervals in `Light` and `Stop` independently, to yield a pseudosample with statistic value c^*. Repeating this process produces a distribution of c^*. The sample value, c, is compared to this distribution to estimate its probability under the hypothesis of independence.

3.1 An Algorithm for Testing a Pattern $\mathcal{A}$

An algorithm that generalizes and extends the procedure of the previous section is shown in Table 2. This algorithm tests the significance of a *set* of Allen relations, and is written in this more general form because we will eventually want to find patterns of changes to graphs that involve more than two propositions (see Sec. 4.). Let us describe the main steps of the algorithm:

1. Characterizing the sample Although the algorithm tests a set of Allen relations, $\mathcal{A}$, for now it is easier to think of the set as comprising a single relation, $R(p_i, p_j)$. The sample statistic is $c_{\mathcal{A}} = Count(\mathcal{A})$, the number of occurrences of $\mathcal{A}$ in the bit array.

2. Randomized Sampling Distribution Randomization is done as shown in Figure 4. For efficiency we randomize all the rows of the bit array, not just those involved in $c_{\mathcal{A}}$. Each time we randomize, we re-count occurrences of $\mathcal{A}$ and add the result, $c^*_{\mathcal{A}}$, to the randomized sampling distribution, D. Also, if $c^*_{\mathcal{A}} \geq c_{\mathcal{A}}$, we increment a counter, m.

This is as far as we need to go to test the hypothesis that $c_{\mathcal{A}}$ occurs significantly more or less often than expected by chance. The probability of $c_{\mathcal{A}}$ under the hypothesis that the dynamics of the propositions in $\mathcal{A}$ are independent is estimated by $p = m/k$.

If p or $1-p$ is very small, we can reject the hypothesis of independent dynamics, with residual uncertainty measured by p.

However, we will often want to test many sets of Allen relations, $\mathcal{A}_1, \mathcal{A}_2, \ldots$, simultaneously. The remaining two steps of the algorithm facilitate this.

3. Calculate critical values Suppose we want to pick a *critical value* in D such that $c_{\mathcal{A}}$ exceeds it with probability less than α. The procedure is simply to sort D from largest to smallest value and pick the $(\alpha \times k)^{th}$ value as the critical value. For example, if $\alpha = .05$ and $k = 500$ then the $.05 \times 500 = 25^{th}$ value of the sorted distribution is the critical value. Similarly, we estimate that $c_{\mathcal{A}}$ is *smaller* than the $((1-\alpha) \times k)^{th}$ with probability α. These critical values are denoted c_U and c_L, respectively.

4. Standardize statistics Critical values for D and sample statistics $c_{\mathcal{A}}$ will depend on many factors, such as the average lengths of intervals during which propositions are true, and the number and the particular Allen relations in $\mathcal{A}$. This means critical values for different $\mathcal{A}$'s cannot be compared, and we cannot say one $\mathcal{A}$ is more significant than another. The solution is to put all critical values and all sample statistics on one scale by standardizing them. Standardizing a distribution means subtracting the mean from each value and dividing by the standard deviation. The relevant standardized critical values are $Z_c = (c - \bar{c}^*)/s^*_c$,$Z_{c_L} = (c_L - \bar{c}^*)/s^*_c$ and $Z_{c_U} = (c_U - \bar{c}^*)/s^*_c$. For any $\mathcal{A}$, we will say it is significantly frequent if $Z_c \geq Z_{cU}$ and significantly infrequent if $Z_c \leq Z_{cL}$. We will say $\mathcal{A}$ is significant if either of these is true. Because Z_c scores are on the same scale, they can be used to rank patterns.

3.2 Demonstration: First Look at Wubbles

We built a bit arrays from the pattern in Figure 1 combined with five noise propositions. Instances of the pattern were distributed throughout the bit array, and the noise propositions were of varying length and independently distributed through the bit array. One bit array was dense, with 200 occurrences of the pattern and 200 intervals of each noise proposition distributed in a bit array of length 12000. The other was less dense, with half as many patterns and instances of noise propositions.

We tested every Allen relationship $\mathcal{A} = R(p_i, p_j)$ that occurred in the bit array. Because the order of the arguments in Allen relations matters, there are actually 13 possible relations (two versions of the the six asymmetric relations in Table 2.2 and *equal*). There are 36 pairs of propositions in the bit array. So we might observe $13 \times 36 = 468$ Allen relations in our sample bit array. In fact, 270 of them appeared in the bit array. As noted in the previous paragraph, we put six of these into the data; all others are accidental.

We tested all Allen relations and ranked them by their Z_c scores. The top 25 relations are shown in Figure 5. The dashed lines in the graph indicate a Z_c score of 5 or lower. No spurious relationship (shown as dotted lines) had a Z_c score above 4, and no true relationship (shown as solid lines) had a score below 32, in either the dense or less dense condition.

```
1. Characterizing the sample:
Let $S$ be a sample (a bit array)
Let $\pi$ be the propositions in $S$ (the rows in the bit array)
Let $\mathcal{A}$ be a set of Allen relations, each of the form $R(p_i, p_j)$
Let $c_{\mathcal{A}} = Count(\mathcal{A})$

2. Randomized sampling distribution, $D$
and calculate $p$ values associated with $c_{\mathcal{A}}$:
Let $m = 0$
Repeat $k$ times
    For each proposition $P \in \pi$
        Shuffle the intervals in $P$
    Push $c^*_{\mathcal{A}} = Count(\mathcal{A})$ onto $D$
    If $c^*_{\mathcal{A}} \geq c_{\mathcal{A}}$, increment $m$
$p_{\mathcal{A}} = m/k$ is the $p$ value for $\mathcal{A}$

3. Calculate critical values:
Sort the values in $D$ from largest to smallest
Find the $(k \times \alpha)^{th}$ and $(k \times (1-\alpha))^{th}$ values in sorted $D$
Call them $c_U$ and $c_L$, repectively.

4. Standardize statistics:
Mean of $D$: $\bar{c}^* = (\sum_{c^* \in D} c^*)/k$
Std. of $D$: $s^*_c = ((\sum_{c^* \in D} (c^* - \bar{c}^*)^2)/(k-1))^{0.5}$
$Z_c = (c - \bar{c}^*)/s^*_c$
$Z_{c_L} = (c_L - \bar{c}^*)/s^*_c$
$Z_{c_U} = (c_U - \bar{c}^*)/s^*_c$
```

Table 2: The algorithm for calculating a p value, a standardized c, and standardized critical values for one set of Allen relations, $\mathcal{A}$.

4. Higher-order Patterns

What if a pattern involves three or more relations between propositions? The pattern in Figure 1, for example, involves four propositions, six Allen relations between pairs of propositions, and many conjunctions of three or more Allen relations. We want to find higher-order patterns that occur more often in the bit array than would be expected by chance. How do werepresent such patterns, how do we find them without being overwhelmed by combinatorics, and how do we test them? Of the three, the easiest question to answer is the last: The algorithm in Table 2 lets us test any pattern, $\mathcal{A}$, not only individual Allen relations. The greater challenges are representing and finding higher-order patterns.

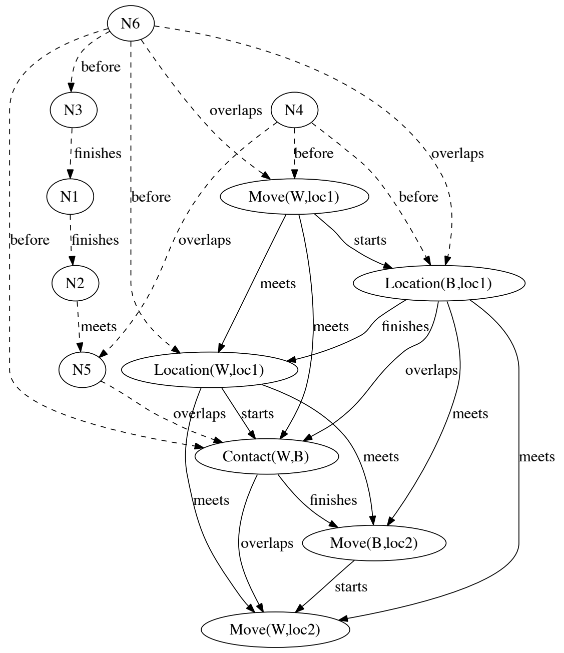

Figure 5: A graph of significant Allen relations from Demonstration 2

4.1 Represention

Some sets of Allen relations are *incomplete* and have two or more qualitatively different completions. For example, $\mathcal{A} = starts(\mathtt{b}, \mathtt{a}) \wedge finishes(\mathtt{b}, \mathtt{c})$ doesn't specify the relationship between a and c. Presumably some relationship exists, but the data doesn't tell us which it is.

This is a well-known inference problem for representations based on Allen relations (Allen, 1983, Alspaugh, 2005, Shih et al., 1998, Gennari, 1998), but we would like to dodge it. So when we test a pattern such as $\mathcal{A} = starts(\mathtt{b}, \mathtt{a}) \wedge finishes(\mathtt{b}, \mathtt{c})$, we test *all* the higher-order patterns it represents.

To do this, we need an unambiguous representation of the relationships between all of the propositions in a pattern. We call this representation *bit patterns.* Bit patterns are easily extracted from compressed bit arrays, introduced in Section 2.1. For example, here are the three bit patterns that are consistent with $\mathcal{A} = starts(\mathtt{b}, \mathtt{a}) \wedge finishes(\mathtt{b}, \mathtt{c})$:

	meets	*overlaps*	*none*
a	100	110	100
b	111	111	111
c	011	011	001

It is not necessary to enumerate all possible bit patterns that are consistent with any pattern, $\mathcal{A}$. All we need is a representation of the pattern that includes wild cards, such as

	$\mathcal{A}$
a	1*0
b	111
c	0*1

and then we can simply count all the patterns in a sample or pseudosamples that match this pattern.

4.2 Searching for Higher-order Patterns

In outline, this will be our approach: A graph of significant Allen relations between pairs of propositions, such as the one in Figure 5, will serve as the starting point for the search for higher-order patterns. We define *significant connected subgraph* to mean a set of nodes connected (in the usual sense) by arcs that represent significant Allen relations. The order of a significant connected subgraph is the number of nodes it contains. Our approach is to "grow" significant connected subgraphs of higher orders, testing each by the randomization procedure in Table 2.

This approach reduces the combinatorics of the search considerably. Approximately $(13^{k-1}n!)(k!(n-k!))$ patterns can be formed by selecting k propositions from n propositions; for instance, given ten propositions, more than a quarter of a million order 3 patterns are possible. However, there are only twenty significant connected subgraphs of order 3 in Figure 5.

Nevertheless, we do not advocate fully automated search for higher-order patterns. Even if the search is limited to significant connected subgraphs, it will eventually be overwhelmed by the combinatorics as n and k increase. Moreover, fully automated search is likely to turn up large numbers of uninteresting patterns. This will happen whenever we track more propositions than causally related propositions. For example, in the Wubble World experiment in Section 5., some of the most significant Allen relations were of the form $equal(\mathtt{Away}(\mathtt{wubble}, \mathtt{wall1}), \mathtt{Away}(\mathtt{wubble}, \mathtt{wall2}))$, even though walls had no causal role in the experiment. One cannot dodge this kind of problem by being more selective about propositions if the whole point of mining is to discover the causally relevant propositions and patterns.

For these reasons, we prefer a semi-automated approach in which a human user iteratively nominates propositions and sets of propositions, and the machine generates patterns of Allen relations around them and tests them.

5. Experiment: Wubble World

Wubble World is a virtual environment with simulated physics in which softbots interact with objects (Hewlett et al., 2007). Wubble World is instrumented to collect distances, velocities, locations, colors, sizes, and other sensory information and represent them with propositions such as `Above(wubble,box)` (the wubble is above the box) and `PVM(wubble)` (the wubble is experiencing positive vertical motion).

```
Collision(wubble,floor)      111111-----------11-----1111
On(wubble,floor)             111111-----------11-----1111
Jump(wubble)                 ----11----------------------
InFrontOf(wubble,box)        -111111---------------------
Towards(wubble,box)          ---1111---------------------
PVM(wubble)                  -----11-----------1---------
Away(wubble,floor)           -----1111---------11--------
Towards(wubble,ceiling)      -----1111---------11--------
Away(wubble,box)             --------1-------------------
NVM(wubble)                  -----------1-11111----1-----
Above(wubble,box)            -------1111111--------------
Towards(wubble,box)          ----------11111-------------
Away(wubble,ceiling)         ----------11111111---111----
Towards(wubble,floor)        ----------11111111---111----
Forward(wubble)              --11111111111111111111111---
Motion(wubble)               ---11111111111111111111111--
SurfaceMotion(wubble)        ---11111111111111111111111--
Away(wubble,box)             ----------------1111111111--
InBackOf(wubble,box)         --------------1111111111111-
```

Table 3: An order 19 bit pattern that represents a wubble jumping over a box.

The dataset for this experiment was generated by manually controlling a wubble to jump over a box twenty times. Each jumping episode was unique, in that the wubble would start closer or further from the box, move more or less quickly, and so on. We recorded 154 propositions over a period of 9229 ticks (i.e., the sample bit array was 154 wide and 9229 long). Over 150,000 Allen relations were possible; 25,949 of them were actually observed in the bit array; and 92,844 were observed during randomization. Our analysis is limited to the relations that appeared in the sample.

Although there is no objective gold standard pattern for wubbles jumping over boxes, Table 3 shows an order 19 bit pattern that represents one of the twenty jump-over episodes in the sample. Of course, many other propositions became true or false during this episode, but we chose these 19 propositions as the "core" components of jump-over. (By implication, 135 of the 154 propositions in the dataset are causally irrelevant.) This *core pattern* contains 181 Allen relations, which we call the *core relations*.

5.1 Results

We tested each of the 25,949 observed Allen relations with the algorithm outlined earlier. Before we ranked the relations by their Z_c scores, we removed some relations that hardly ever occurred but that appeared to be significant because their sampling distributions had little or no variance. We ranked all relations by s_c^* and removed those in the lowest 5% of the distribution *and* had $c < 5$ occurrences in the sample. This removed 1119 relations, leaving 24,830 for further analysis.

We constructed an ROC curve for the following discrimination task: Using only Z_c for a relation $\mathcal{A}$, predict whether $\mathcal{A}$ is one of the 181 core relations from our "gold standard" in Table 3. The area under the curve (AUC) was a respectable 86%. We also used c, the count of $\mathcal{A}$ in the sample as a predictor and found, not surprisingly, that it performed well, with $AUC = 93\%$. However, the ROC curve for Z_c rose more steeply; for example, five core relations were found in the first 52 when they were ranked by Z_c scores, while it was necessary to search 310 relations to find five core relations when they are ranked by c. Testing and ranking by Z_c scores finds more correct results sooner than ranking by c scores.

Even so, it is inconceivable that any algorithm would find all and only the 181 core relations in Table 3 without some guidance from a human user. The combinatorics of the problem preclude it. Every statistical test has a p value — a probability that a spurious relationship will appear significant — and when there are 25,949 order 2 patterns and millions of higher-order patterns to be tested, the user will soon drown in spurious results.

A better approach is to use our algorithm to screen lower order relations and test the significance of higher order ones, but let the user grow higher order relationships that he or she finds interesting. This approach is illustrated by Figure 6. The user provides a set of propositions that he or she is interested in, and the system finds all of the significant patterns that include one or more members of the set. In this experiment, we looked for patterns associated with three propositions: `Jump(wubble)`,`Away(wubble,floor)`, and `PVM(wubble)` (PVM means positive vertical motion). Our algorithm was directed to find all Allen relations between these and any other propositions that had Z_c scores greater than 20. The resulting graph has 11 propositions and 26 significant Allen relations. Of these, 21 are core relations, as defined earlier, and the remaining five are not core relations but are perfectly reasonable components of a jump-over pattern, such as $before$(`PVM(wubble)`, `NVM(wubble)`).

The algorithm was then used to test the significance of higher order relations as described in Section 4.. We directed the algorithm to find all significant order 3 relations between `Jump(wubble)`, `Away(wubble,floor)`, and `PVM(wubble)`. Two of the resulting bit patterns, with comments, are:

Jump	1100	*The jump itself overlaps the beginning of*
PVM	0110	*positive vertical motion and moving away*
Away	0111	*from the floor. The latter continues after*
		the end of PVM. $Z_c = 245.27$

Jump	11000	*As above, but after the wubble lands*
PVM	01100	*it bounces, resulting in a second PVM.*
PVM	00001	$Z_c = 37.1$

6. Previous Work

In temporal pattern mining research, there are several critical choices that researchers must make in order to find interesting patterns. First they must

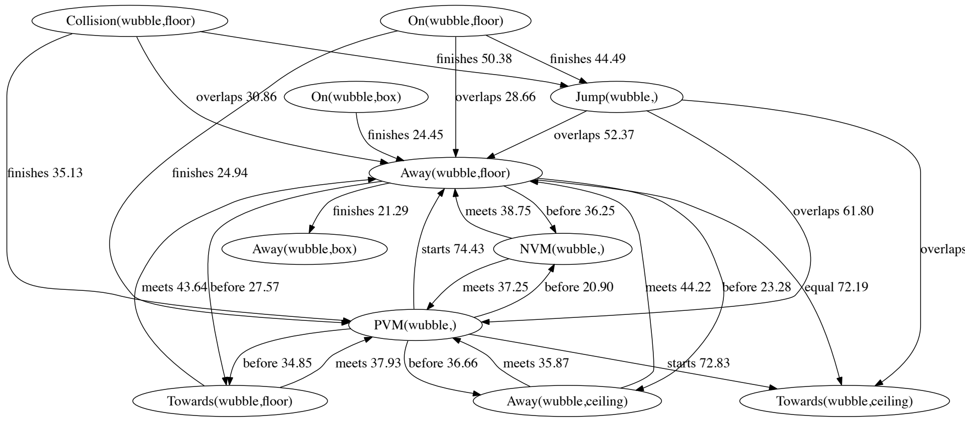

Figure 6: A graph of significant Allen relations from the Wubble World experiment

decide on a measure that determines what makes one pattern better than another (often called support), and our support metric selects patterns that occur more (or less) frequently than expected by chance. Secondly, researchers must select a representation of the temporal relationships between propositions. Any representation must solve two problems: First, a pattern of order 3 or higher can be represented as compositions of Allen relations in several ways (e.g., we could say $meets(\mathtt{a}, (during(\mathtt{b}, \mathtt{c})))$ or $during(meets(\mathtt{a}, \mathtt{b}), \mathtt{c}))$. A canonical form is desired, and is provided by our bit pattern representation. Second, a representation should capture *all* the Allen relations that exist between propositions. A sentence such as $during(meets(\mathtt{a}, \mathtt{b}), \mathtt{c})$ doesn't say whether `c` occurs during `a,b` or both, though one of these must be true. Bit patterns make explicit all the Allen relations between fluents.

Kam and Fu (shan Kam and chee Fu, 2000) present an interesting canonical form, based on right concatenations, for compositions of three or more fluents. However, this representation does not capture all the Allen relations in a composition. In the algorithm presented by Kam and Fu, the frequency of the pattern was used to select interesting patterns. The algorithm presented in (Cohen, 2001, Cohen et al., 2002) admitted a larger set of possible patterns than Kam and Fu, but lacked a canonical representation of patterns. This research is the only other research that we've found that also extracts statistically significant patterns. Other research (Höppner and Klawonn, 2002, Winarko and Roddick, 2007) uses a matrix representation that captures all $\frac{k(k-1)}{2}$ pairwise Allen relations between k fluents, and again uses frequency to determine the support of a pattern. In (Höppner and Klawonn, 2002), support is defined as the frequency within a sliding window, ensuring a degree of locality between the internal relationships of the pattern, whereas in (Winarko and Roddick, 2007) the support metric is similar to (shan Kam and chee Fu, 2000). Although the support metrics differ between our work and the work in (Höppner and Klawonn, 2002, Winarko and Roddick, 2007), it's worth noting that there exists a mapping function from the pairwise representation to our bit pattern representation. The bit pattern representation offers a more human-friendly view of the pairwise relationships than the matrix representation.

In (Mörchen and Ultsch, 2007), the authors developed a technique that doesn't rely on Allen relations. The authors introduce a hierarchical interval language called Time Series Knowledge Representation (TSKR). Like the bit pattern representation, the language was developed to address the shortcomings of concatenation with Allen relations. The TSKR extends the Unification-based Temporal Grammar (UTG) proposed by Ultsch and Mörchen in (Mörchen and Ultsch, 2004), to increase the expressivity of the language. Larger temporal patterns are found using a support metric based on observation frequency. Both UTG and TSKR rely on a smoothing operator to allow fuzzy boundaries to make a more "robust" Allen *equal* operator. It is unclear how this type of smoothing operator would perform with the data sets presented in this paper.

7. Future Work

Four extensions of this work are underway. First, we wish to introduce variables into our representations. Currently, two patterns of graph dynamics that occur at different locations, or involve different colored objects, but are otherwise identical, are treated as different patterns. This prevents us from mining simple generalizations of patterns. It will be necessary to make the counting part of our algorithm recognize grounded instances that match generalizations, perhaps by allowing wildcards in bit patterns as in Section 4.1.

Second, the algorithm in Table 2 yields probabilities of patterns but not conditional probabilities of partial patterns given other partial patterns. For example, we currently randomize propositions `a,b,` and `c` and count occurrences of $\mathcal{A} = R_1(a,b) \wedge R_2(b,c)$ but we don't hold `a,b` fixed as they occur in the sample and randomize `c` with respect to fixed `a` and `b`. If we did this, then we could estimate the conditional probabilities of Allen relations involving some propositions given others. Early results show that this kind of conditional counting is more sensitive and better able to separate spurious patterns from true ones, and it is very much in keeping with the iterative, semi-automated search for higher-order bit patterns described in Section 4.2.

Third, to support this kind of semi-automated search, we are developing an interface based on the kind of graph representation shown in Figure 6. It will be possible for users to click on subgraphs and have them tested for significance.

Fourth, we are developing a causal modeling capability based on counterfactuals. The three conditions for `a` to be a *counterfactual cause* of `b` are 1) `a` occurs before `b`, 2) `a` and `b` are dependent, not accidental, and 3) `a` is necessary for `b`. Consider a simple pattern, $\mathcal{A} = \mathit{meets}(\mathtt{a},\mathtt{b})$, meaning that `b` becomes true right after `a` ends. This establishes the first condition for counterfactual cause. Suppose this pattern is significantly frequent, meaning there are more instances of $\mathcal{A}$ in the sample than would be expected by chance. This establishes the second condition. If we can devise an efficient way to show that `b` does *not* become true *unless* `a` becomes false immediately beforehand, then `a` is a necessary condition for `b`, establishing the third condition for counterfactual cause.

8. Conclusion

We modeled propositional dynamics in terms of Allen relations and showed how to mine statistically significant patterns comprising Allen relations between intervals in which propositions are true and false. Conjunctions of three or more Allen relations can be ambiguous, so we introduced bit patterns, a canonical form for patterns relating three or more propositions. We demonstrated that our procedure could, with heuristic guidance from a user, find higher order patterns with some accuracy.

References

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

Alspaugh, T. A. (2005). Software support for calculations in allen's interval algebra. *ISR Technical Report # UCI-ISR-05-02*, page 6.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.

Cohen, P. R. (2001). Fluent learning: Elucidating the structure of episodes. *Advances in Intelligent Data Analysis*, pages 268–277.

Cohen, P. R., Sutton, C., and Burns, B. (2002). Learning effects of robot actions using temporal associations. *International Conference on Development and Learning*.

Gennari, R. (1998). Temporal reasoning and constraint programming: A survey. *CWI Quarterly*, 11(2&3):163–214.

Hewlett, D., Hoversten, S., Kerr, W., Cohen, P. R., and Chang, Y.-H. (2007). Wubble world. *Proceedings of the Third Artificial Intelligence for Interactive Digital Entertainment Conference*.

Höppner, F. and Klawonn, F. (2002). Finding informative rules in interval sequences. *Intelligent Data Analysis*, 6:237–255.

Kerr, W., Hoversten, S., Hewlett, D., Cohen, P. R., and Chang, Y.-H. (2007). Learning in wubble world. *International Conference on Development and Learning*, pages 330–335.

Mörchen, F. and Ultsch, A. (2004). Mining hierarchical temporal patterns in multivariate time series. *Lecture Notes in Computer Science*, 3238:127–140.

Mörchen, F. and Ultsch, A. (2007). Efficient mining of understandable patterns from multivariate interval time series. *Data Min Knowl Disc*, 15(2):181–215.

shan Kam, P. and chee Fu, A. W. (2000). Discovering temporal patterns for interval-based events. *Lecture Notes in Computer Science*, 1874:317–326.

Shih, T. K., Chang, A. Y., Lin, H.-J., Yen, S.-H., and Chiu, C.-F. (1998). Interval algebra for spatio-temporal composition of distributed multimedia objects. *Proceedings of the 1998 International Conference on Parallel and Distributed Systems*.

Winarko, E. and Roddick, J. F. (2007). Armada – an algorithm for discovering richer relative temporal association rules from interval-based data. *Data & Knowledge Engineering*, 63(1):76–90.