

**AprioriSetsAndSequences:
Mining Association Rules from Time Sequence Attributes**

by

Keith A. Pray

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 6, 2004

APPROVED:

Professor Carolina Ruiz, Major Thesis Advisor

Professor Matthew O. Ward, Thesis Reader

Professor Michael A. Gennert, Head of Department

Abstract

We introduce an algorithm for mining expressive temporal relationships from complex data. Our algorithm, `AprioriSetsAndSequences` (ASAS), extends the Apriori algorithm to data sets in which a single data instance may consist of a combination of attribute values that are nominal sequences, time series, sets, and traditional relational values. Datasets of this type occur naturally in many domains including health care, financial analysis, complex system diagnostics, and domains in which multi-sensors are used. `AprioriSetsAndSequences` identifies predefined events of interest in the sequential data attributes. It then mines for association rules that make explicit all frequent temporal relationships among the occurrences of those events and relationships of those events and other data attributes. Our algorithm inherently handles different levels of time granularity in the same data set. We have implemented `AprioriSetsAndSequences` within the Weka environment and have applied it to computer performance, stock market, and clinical sleep disorder data. We show that `AprioriSetsAndSequences` produces rules that express significant temporal relationships that describe patterns of behavior observed in the data set.

Contents

1	Introduction	1
1.1	Context of This Research	1
1.2	Motivation	1
1.3	Problem Definition	3
1.4	Contributions of This Work	4
2	Background	6
2.1	Association Rules	6
2.2	Apriori Algorithm	7
2.3	Pattern Matching	8
2.4	Weka and ARFF	8
2.5	Apriori Sets Algorithm	9
3	Related Work	10
4	Event Attributes and Event Identification	14
4.1	Description of Sequence Attributes	14
4.2	Description of Event Attributes	15
4.3	Event Identification	16
5	Data Representation	19
5.1	Extension of ARFF	19
5.2	ASAS Item Sets	20
5.3	Item Set Data Structures	23
6	ASAS Algorithm	26
6.1	Input	26
6.2	Frequent Item Set Generation	26
6.2.1	Level 1 Candidate Generation	28

6.2.2	Level 1 Counting Support	29
6.2.3	Level 2 Candidate Generation	30
6.2.4	Level 2 (and up) Counting Support	31
6.2.5	Level 3 (and up) Candidate Generation	33
6.2.6	Multiple Events of The Same Type	35
6.2.7	Duplicate Item Sets	36
6.3	Rule Generation: Calculating Confidence	39
7	ASAS Implementation	44
7.1	Data Preprocessing Filters	44
7.2	Implementing Rule Generation	46
7.3	Support Counting Prune	46
7.4	Other Features	47
8	Experimental Evaluation of ASAS	49
8.1	Evaluation on The Computer Performance Domain	49
8.1.1	Data Collection	49
8.1.2	Rules	50
8.1.3	Summary of Results	52
8.2	Evaluation on The Stock Market Domain	53
8.2.1	Stock Market Data	53
8.2.2	Rules	54
8.2.3	Summary of Results	60
8.3	Evaluation on The Human Sleep Domain	60
8.3.1	Clinical Sleep Data	60
8.3.2	Rules	61
8.3.3	Summary of Results	63
8.4	Performance Evaluation of ASAS	63
8.4.1	Time to Mine	64
8.4.2	Effect of Increasing Maximum Number of Same Type Events Allowed in a Rule	67
9	Conclusions and Future Work	71
9.1	Conclusions	71
9.2	Future Work	72
9.2.1	ASAS Extensions	72
9.2.2	Improve Candidate Generation	72

<i>CONTENTS</i>	iii
9.2.3 Improve Support Counting	76
9.2.4 System Features	77
A Appendix Readme File	86
B Appendix ASAS Log Metrics	94
C Appendix Computer Performance Metrics	97
D Appendix Computer Performance Data	100
E Appendix Stock Market Data	105
F Appendix Sleep Data	113

List of Figures

1.1	A sample of a data set containing complex instances. Here, sequential values are represented in a graphical manner only for the first row to save space.	3
4.1	A Sequence Example. Here the top row represents a basic time line without units. The bottom row contains the values of the sequence for the CPU attribute.	14
4.2	Generating Item Types	15
4.3	Here is the sample data set with the CPU sequence attribute replaced with the CPU-Increase event attribute.	16
4.4	Increase Event	17
4.5	Decrease Event	17
4.6	Sustain Event	18
5.1	An item set containing 3 events. Disk Increase $[t_0, t_1]$, Disk Sustain $[t_2, t_3]$, Disk Sustain $[t_4, t_5]$	21
5.2	Adding the Increase Event 2 to item set in Figure 5.1	22
5.3	Relative Time Line Array. DI = Disk Increase, CI = CPU Increase, b = begin, e = end.	24
5.4	Relative Time Line Array. DI = Disk Increase, CI = CPU, Increase, MS = Memory Sustain, b = begin, e = end.	24
5.5	Relative Time Line Array. DI = Disk Increase, CI = CPU, Increase, MS = Memory Sustain, b = begin, e = end.	25
6.1	Level 1 Mapping. DI = Disk Increase, CI = CPU Increase, b = begin, e = end.	30
6.2	All Possible Temporal Relationships Between Two Events. Each event is depicted by a rectangle (one dark and one light). Numbers in the rectangles denote relative times t_0, t_1, t_2 , and t_3	31

6.3	Level 2 Mapping. DI = Disk Increase, CI = CPU Increase, b = begin, e = end.	32
6.4	Combine Item Sets	34
6.5	Selective candidate generation by joining events A and B from Figure 6.4	35
6.6	Selective Candidate Generation Result	35
6.7	Candidates from Combining Item Sets 1 and 2	37
6.8	Candidates from Combining Item Sets 1 and 3	38
6.9	Candidates from Combining Item Sets 2 and 3	39
6.10	Prefix Tree	40
6.11	Apriori Calculate Confidence	41
6.12	ASAS Calculate Confidence	42
6.13	Finding Antecedent	42
6.14	Extending Antecedent to Include Consequent	43
7.1	Several Time Granularities	44
8.1	Partial Sample of A Single Performance Data Instance. See Appendix C and Appendix D for a complete listing.	51
8.2	Idle Process Time Increases $[t_0, t_1] \Rightarrow$ Total Process Time Decreases $[t_0, t_1]$	52
8.3	Total Memory in Use Decreases $[t_0, t_1] \Rightarrow$ Total Process Time Decreases $[t_0, t_2]$	53
8.4	Performance Instance. See Appendix C and Appendix D for a complete listing.	54
8.5	A sample of a company and stock market dataset containing complex instances. Here, sequential values are represented in a graphical manner only for the first row to save space.	55
8.6	AMD Ascending Triangle $[t_0, t_1]$ and SUNW Sustain $[t_2, t_3] \Rightarrow$ CSCO Expand Merge $[t_4, t_5]$	57
8.7	INTC Sustain $[t_0, t_1] \Rightarrow$ INTC Sustain $[t_2, t_3]$	59
8.8	A sample of a dataset containing complex instances. Here, sequential values are represented in a graphical manner only for the first row to save space. G stands for Gender with values M (male) and F (female). S_i denotes sleep stage i . Medication names have been abbreviated.	61
8.9	Sleep Stage 1 $[t_0, t_2]$ and Granule 700-749 $[t_1, t_3] \Rightarrow$ Mild OSA	62
8.10	Sleep Stage 2 $[t_1, t_3]$ and Granule 150-199 $[t_0, t_2] \Rightarrow$ Moderately Severe OSA	62
8.11	Sleep Stage 3 $[t_1, t_3]$ and Granule 300-349 $[t_0, t_2] \Rightarrow$ No OSA	63

8.12	Sleep Stage REM $[t_1, t_2]$ and Granule 750-799 $[t_0, t_3] \Rightarrow$ No OSA	63
8.13	Data Set Events and Total Mining Time	65
8.14	Frequent Item Sets and Total Mining Time	66
8.15	Data Set Time Line and Mining Time per Frequent Item Set	67
8.16	Data Set Events and Mining Time per Frequent Item Set	68
8.17	Diminishing Returns With Higher Number of Same Type Events Allowed In a Rule	69
8.18	Diminishing Returns With Higher Number of Same Type Events Allowed In a Rule and Lower Support	70
9.1	Candidates from Combining Item Sets Level 1	74
9.2	Candidates from Combining Item Sets Level 2	75
9.3	Candidates from Combining Item Sets Level 1a Improved	76
9.4	Candidates from Combining Item Sets Level 1b Improved	77
9.5	Candidates from Combining Item Sets Level 2 Improved	78
E.1	Rounded Top	106
E.2	Selling Climax	106
E.3	Ascending Triangle	107
E.4	Broadening Top	107
E.5	Descending Triangle	108
E.6	Double Bottom	108
E.7	Double Top	109
E.8	Head and Shoulders	109
E.9	Inverse Head and Shoulders	109
E.10	Panic Reversal	110
E.11	Rounded Bottom	110
E.12	Triple Bottom	110
E.13	Triple Top	111
E.14	Sustain	111
E.15	Increase	111
E.16	Decrease	112

Chapter 1

Introduction

1.1 Context of This Research

This paper expands the general use of association rules [AIS93] to complex temporal relationships essential to many domains. Our association rule mining approach discovers patterns in data sets whose instances consist of any combination of standard, set-valued, and sequential attributes. These data sets occur naturally in several scientific, engineering, and business domains, and are generally richer than the transactional, the relational, and the sequence data sets to which association rule mining has been applied. To date, our mining approach has been applied to computer system performance, stock market analysis [HL03], and clinical sleep disorder data [Lax04]. Complex system diagnostics, network intrusion detection, and medical monitoring are some related domains to which this work can also be applied.

1.2 Motivation

A main motivation for mining these temporal rules from complex data comes from our previous experience working on the performance analysis of a hardware and software backup/restore product. The time it takes to complete a backup can be of great importance. One of the most labor consuming tasks is to predict this time. A rule based expert system was built as a way to disseminate the performance knowledge used to predict backup time. This expert system quickly grew to include over six hundred rules. The need for automating the discovery of new rules was apparent.

To create a rule to handle a new feature or new technology, an estimation is made about how performance will be affected. Tremendous amounts of information are col-

lected to verify that the estimation is accurate. For a small scenario including a backup server, a single client, the Ethernet network that connects them, and a single SCSI connected tape drive, an average of 2 MB of data can be collected every ten minutes that a test is run. This includes CPU and memory usage for all processes running on both server and client, I/O metrics for storage sub-systems and the tape drive and the connections between them and the client and server machines as well as network traffic. With a short test run of an hour, analyzing this amount of data manually is very impractical. The reasoning behind the initial estimation serves to focus analysis efforts but is usually inadequate. It is often the case that the estimation is not sufficiently accurate and more analysis must be done to identify the factors behind the observed performance. This process can take weeks or months and often involves running new tests and collecting even more data to be analyzed. With our approach to mine association rules from these data, the resulting rules can be used to focus analysis efforts and possibly explain the observed behavior.

Figure 1.1 depicts a small sample of the type of complex data set that our mining approach applies to. In this computer performance data set, each instance (row) corresponds to a single test in which a process was run until it completed a task. The process is a machine learning task using one of five techniques: C4.5, Instance Based learning, Naive Bayes, or Neural Network Back Propagation. The attributes describe the conditions under which the test was run and state information collected during the test including “standard” attributes such as processor speed and physical memory size; set-valued attributes such as the options or flags that were specified to the process and which algorithms the process implemented; and sequential attributes such as the CPU utilization percentage, memory usage, and the total number of processes running over time. Other such numeric and nominal sequential attributes not shown in Figure 1.1 include CPU usage of all other processes, memory usage of all other processes, I/O activity on main storage devices, memory paging, and process state (running, exit normally, exit without output). All time sequence attributes in a single data instance share the same time line.

Events of interest in the sequential attributes in this data set include among others *increases* and *decreases* in the utilization of a resource (CPU, memory), going above/below a certain usage threshold, and whether or not a process terminates normally.

A sample interesting temporal pattern in this domain is: “Java processes running a machine learning algorithm that are not given the -P option, that exhibit an increase in its memory usage, and that during this increase its memory paging also increases tend to, with likelihood 82%, exit prematurely”. This temporal pattern is captured by our

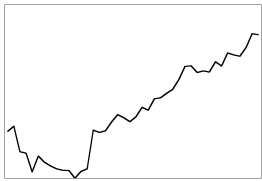
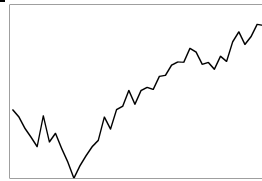
ID	flags	CPU %	CPU (MHz)	memory %	memory (kB)	algorithms
1	{-R, -H}		600		523760	{neural net, backpropagation}
2	{-H}	40, 52, 67, 80, ...	600	10, 26, 46, 69, 86, ...	523760	{C4.5}
3	{-U, -R}	10, 39, 87, 96, ...	300	19, 50, 82, 80, 70 ...	260592	{naive Bayes}
...

Figure 1.1: A sample of a data set containing complex instances. Here, sequential values are represented in a graphical manner only for the first row to save space.

association rule:

$$\begin{aligned} &\text{flag}=-\text{P-missing} \ \& \ \text{process(java)-mem-usage-increase} \ [t_0, t_2] \ \& \\ &\text{process(java)-page-increase} \ [t_1, t_2] \ \Rightarrow \ \text{process(java)-exit-without-output} \ [t_3, t_4] \\ &\text{Conf: } 82\%. \end{aligned}$$

Here, t_0, t_1, t_2, t_3 , and t_4 are *relative* time indices that are used to express the relative order in which these events are observed.

1.3 Problem Definition

The algorithm presented here to mine these temporal relationships, *AprioriSetsAndSequences*, takes as input a complex temporal data set as described, a set of predefined event types, and the standard Apriori minimum support and minimum confidence parameters. An event type is a template of a subsequence of interest in a time sequence attribute. Our mining algorithm starts by identifying occurrences of these event types in the corresponding sequence attributes as described in Section 4. Then, it generates all the frequent relationships among the temporal occurrences of these events and among these and the other non-temporal attributes. Section 6 describes the algorithm in detail. Since there are 13 ways of sorting just two events in time (*before, after, overlaps, etc.*) [All83], and the number of possible orderings of a set of events grows exponentially with the number of events in the set, central issues in the design and implementation of our mining algorithm are the strategy used to prune unnecessary orderings from consideration, and the data structures used to effectively handle potentially frequent orderings of events. We describe these in Chapter 6. This chapter also describes our extensions of the basic notions of support and confidence needed to handle multiple occurrences of an

event or a pattern in a complex data instance. Section 8.4 presents an evaluation of our mining algorithm in the stock market domain. Section 9.1 summarizes the contributions of this paper and discusses future work investigating alternative measures of item set interestingness and alternative search techniques for item set generation in the context of complex data.

1.4 Contributions of This Work

This thesis work introduces AprioriSetsAndSequences (ASAS), an algorithm for mining associations rules containing complex temporal relationships. In addition to sequential attributes, ASAS also handles set valued attributes as well as traditional relational values. ASAS also has the capability of mining temporal relationships over several time granularities simultaneously. ASAS uses two data structures specific to event items. The first indexes time intervals during which the event items in an item set occur. The second indexes item sets containing event items.

The ASAS algorithm was implemented as an add on to a publicly available machine learning and data mining tool known as Weka [FW00]. The file format Weka uses, ARFF, was extended to handle sequence attributes and event set attributes.

A filter for detecting simple events in sequences was implemented as a data preprocessing filter for Weka. Also implemented as a filter was the ability to create different time granularities for data sets containing events. A granularity refers to a unit of time such as a second, minute, hour, month, etc. It can also refer to an interval of time such as a weekday or weekend, holidays, etc.

Tools for constructing data sets containing sequences from multiple data sets were built. These tools allowed a single data instance to be create from an entire data set where each ordered instance in that set represent a single unit of time along a time line specific to that data set.

A tool for automatically collecting performance data on Windows machines during the execution of a specific process was built. This created a source of preliminary data with which to test ASAS during development. This tool automatically transforms output from the Windows Performance Monitor into an ARFF file.

Also during the course of this work an add on package for Weka consisting of WPI KDDRG implemented features and algorithms was compiled and maintained. Full documentation of instructions on use of the add on package and of the all the classes were also created.

For this thesis ASAS was applied to data sets containing computer system performance, stock market and company financial, and clinical sleep study data. Also in the course of this work ASAS was used by other students working with stock market data [HL03], clinical sleep study data [Lax04], motif and gene expression data [BFG⁺03], and intelligent tutoring system logs [Fre04]. In support of these projects, demonstrations, presentations, individual help sessions, and implementation of additional features were provided.

Chapter 2

Background

2.1 Association Rules

[AIS93] introduced the application of association rules to market basket data. Different data representations such as attribute value pairs and relational data have since been translated into items as originally used in the market basket paradigm.

An association rule simply states that there exists a statistical relationship between two or more items in a data set. An item is a literal or attribute value pair occurring in the data set. This relationship is generally shown in the form $X \Rightarrow Y$ where X and Y are sets of items. The statistic measures commonly used with association rules are confidence and support. *Support* is the percentage of the data instances containing both X and Y . *Confidence* is the percentage of instances containing X that also contain Y . Other measures used to describe an association rule are Lift and Chi-Squared. *Lift* is the ratio of the percentage of instances containing X that also contain Y and the percentage of the instance that contain Y . In other words, the confidence of the rule divided by the support of Y . *Chi-Squared* is a test often used to determine if the difference between observed frequency and expected frequency, if any, is due to some difference in behavior or by chance. For association rules if the antecedent of the rule makes the consequent more likely to happen the Chi-Squared value will reflect this. The expected frequency of the consequent is simply the support of the consequent. The observed frequency is the percentage of instances containing the antecedent that also contain the consequent. If the antecedent makes the consequent more frequent then the rule may be more interesting.

2.2 Apriori Algorithm

AprioriSetsAndSequences extends the Apriori algorithm introduced in [AS94]. Specifically the implementation builds upon AprioriSets [Sho01] which was incorporated into the machine learning and data mining system Weka [FW00] by [SS02].

In order to understand the development of an algorithm that successfully mines association rules from time sequence attributes, it is helpful to examine how Apriori works and why it is unsuited for this data set. Apriori was introduced in [AS94] with an application to market basket analysis, and with a transactional perspective of the input data.

As input the Apriori algorithm takes a data set which consists of transactions. Each transaction consists of a list of items which are simply literals. Relational data, or attribute value data can be easily transformed into a transactional form. This is done by creating an item for each attribute value pair that exists in the data set. Minimum support and minimum confidence thresholds are also input to the Apriori algorithm.

As an intermediate step, Apriori produces frequent item sets. An item corresponds to an attribute-value pair in the case of tabular data. An item set is frequent if its support is greater than the user specified minimum support. An item set of size k is called a k -item set. Apriori's final output is a list of all association rules whose confidence and support are above the minimum thresholds.

The Apriori algorithm is based on the Apriori principle, which states that every subset of a frequent item set is also frequent. It first calculates the support of every individual item by counting the instances that contain the item. Apriori then finds larger and larger frequent item sets, step by step. The general steps of Apriori's frequent item set finding algorithm are shown below.

1. Given a data set create a list of candidate item sets of size $k = 1$, one item set for each item appearing in the data set.
2. For each candidate item set of size k count the number of instances in the data set containing it. This number is the support count of the item set.
3. Remove candidate item sets of size k not meeting the required minimum support count. The remaining item sets are the frequent item sets of size k .
4. Generate candidate item sets of size $k+1$ by joining together each pair of frequent item sets of size k that share $k - 1$ item in common.

5. Prune candidate item sets that are known to be infrequent because of the Apriori principle.
6. Increment k by one.
7. Repeat steps 2 through 6 until there are no more frequent item sets or candidate item sets of size k .

Apriori then generates association rules from the frequent item sets. Association rules must meet the minimum confidence criteria. The minimum confidence is specified by the user. The confidence of an association rule is the Probability of the Antecedent given the Consequent. Other metrics used to describe association rules are Lift and Chi-square. We use the procedure introduced in [Alv03] to compute these values.

2.3 Pattern Matching

There are many methods available for matching patterns against a sequence. For numeric sequences, these methods include similarity search using Discrete Fourier Transform [Hal00, AFS93], bounding boxes for clustering over a Fourier Transform feature space [FM94], and dynamic programming time warping [BC95].

To use time warping, patterns are represented as templates. The time axis of a time series can be stretched or compressed to match the template's. This makes the length of the time series negligible in finding a match. By normalizing along the other axis, one can match template patterns regardless of scale as well.

For nominal sequences, work has been done to discover common patterns in transactional sequences [SA96]. These are all valid methods for specifying and identifying events for AprioriSetsAndSequences.

2.4 Weka and ARFF

Weka is a machine learning and data mining system from the University of Waikato, New Zealand [FW00]. It provides an infrastructure for developing machine learning and data mining algorithms. It provides filters for preprocessing of data. Weka is an open source system written in Java. AprioriSetsAndSequences is implemented as an add on for Weka. Identification of events in sequences is implemented as add on filters for Weka.

Weka uses the Attribute-Relation File Format (ARFF) to store and retrieve data sets. It consists mainly of a header section and a data section. the header section lists

each attribute and its data type. Supported data types are numeric, nominal, string, and date. In the data section each line of text, denoted by a carriage return, represents a single instance in the data set. Values for each attribute in the header appear in the order specified in the header and are separated by commas. A sample of ARFF is below.

```
@relation 'A sample ARFF file'

@attribute x numeric
% this is a comment, following is a nominal attribute
@attribute color <red,blue,green>
@attribute name string

@data
2, green, keith
4.5, red, carolina
```

2.5 Apriori Sets Algorithm

AprioriSets [Sho01] is an extension of the Apriori algorithm to mine association rules from set valued data. Instead of an attribute having a single value it has a value that is a set of elements. Set valued attributes are transformed into items for use in Apriori by defining an item for each attribute and set element pair. AprioriSets was added to the Weka system [SS02]. AprioriSetsAndSequences adds the ability to mine associations from time sequence attribute and event attribute types to AprioriSets.

Chapter 3

Related Work

There has been a great deal of interest in devising approaches to mine associations from sequential data. This interest has taken the form of work which aims to find associations using temporal information and finding associations which contain temporal information. Dunham [Dun03] and Han and Kamber [HK01] offer excellent summaries of temporal mining terms and algorithms currently known. Roddick and Spiliopoulou [RS02] provide an excellent survey of temporal knowledge discovery.

Ale et al. [AR00] use the time of a transaction as a rule pruning method allowing the use of lower support to find more interesting rules. It also uses this information to declare rules obsolete if the transactions that support a rule are from a certain distance in the past. It does not produce rules that express temporal relationships between the items in a rule as does AprioriSetsAndSequences (ASAS).

Agrawal and Srikant [AS95] and Zaki [Zak00] use the notions of *time window* and *max/min gaps* to address the complexity of the mining task. Zaki considers item set constraints for this same purpose. One difference between ASAS and these approaches is that the notion of *event* in ASAS is a non-trivial time interval and theirs is a point in time (instantaneous events). This has a profound impact on the expressiveness of our association rules and on the complexity of the mining process, as in our case the possible orderings of two single events is 13 while for them that number of orderings is only 3. Another important difference is that in the ASAS approach data instances that are combinations of several attribute types are considered, while their instances are sequences of transactions.

A notion similar to time windows is used by Chen and Petrounias [CP99] to identify periods of time during which a particular association rule is valid. The rules themselves contain no temporal information otherwise.

Mannila et al. [MTV95, MT96] present a method for finding episodes. An episode is a collection of events occurring inside the same window of time. Here events refer to instantaneous occurrences occupying a single unit of time. The temporal relationships possible between two events with this representation is limited to three. The two events can occur at the same time, one before the other, or one after the other. AprioriSetsAndSequences uses a representation where events occur during an interval. Hence there can be thirteen possible temporal relationships between events. This is more expressive and enables AprioriSetsAndSequences to utilize more information in the data set.

A common method for mining patterns from time sequence data sets is using a template to narrow down the many possible patterns to mine for. Although not explicitly stated, work by Oates et al. [OSGC95] can be thought to be using a template of sorts. Dependencies between events occurring in multiple streams are found. A dependency taking the form $A \Rightarrow t B$ with a certain probability. A is a set of events occurring at time i . B is a set of events occurring at $i + t$. The size of A and B are fixed as is time t . These restrictions effectively form a template which narrows down the search space of dependencies. The algorithm is not directly comparable to an Apriori-like algorithm such as Apriori Sets And Sequences since it does not find all possible dependencies (rules) fitting the criteria specified by the user. The data set accepted by the algorithm presented by Oates et al. would constitute a single instance of an AprioriSetsAndSequences data set. The events are instantaneous instead of ASAS interval based events.

Bettini et al. [BWJL98] use a template consisting of event variables and the time between events. The time can be expressed in multiple granularities. The sequence of events could be defined along a time line with seconds as the unit of measure. Possible granularities include minutes, hours, days, business days, weeks, etc. In [BSJ96, BWJL98] timed automata are used to count all the possible solutions to the template specified by the user. Also specified by the user is the first occurring event type in the template. This is necessary for deciding when to create another automata to determine if an occurrence of a specific instantiation of the template is present and as a basis for calculating confidence of the frequent patterns found. Later work with granularities [LWJ00] propose three algorithms that follow an Apriori-like generate and count method for finding temporal patterns. Requiring a template from the user assumes the user has a great deal of knowledge about the data set. It also precludes the finding of novel, unexpected patterns. The ASAS approach is more general than theirs in that the user is not restricted to use just one temporal template for each mining task, as ASAS considers all possible temporal patterns that are frequent. Also, several time-granularities can be explored during the same mining task, just by defining an event-based attribute for each relevant time-

granularity and letting them “intersect” with other events of interest. The events in these time–granularity attributes would simply define intervals of the time granularity it represents. For a real time line whose units are days the time–granularity attribute for weeks would identify an event for every seven days along the real time line.

An interesting approach of using templates to mine temporal information can be found in [FLYH99, LFH00, TLHF03]. by Feng et al. Rules that associate items between transactions where transactions represent successive units of time are mined. A template for a rule specifying temporal constraints on items is used to restrict the search for association rules. If each data set in this approach were used to create a single instance of a data set, that resulting data set would be of the same form as used in AprioriSetsAndSequences. ASAS does not require the use of a template to produce temporal association rules.

There has been work that produces association rules containing events occurring during intervals that does not use templates. An algorithm presented by Rainsford and Roddick [RR99] takes as input a data set of regular transactional instances where some of the items may contain temporal information specifying the time or interval during which the item is valid. It does not allow for multiple occurrences of events in the same instance. The rules produced have, in addition to the common association rule confidence measure, a temporal confidence factor. For pairs of events in an association rule there may be a frequently observed temporal relationship. The temporal confidence factor is the percentage of instances that support the regular association rule that also support the temporal relationship. All pairs of events may not have a frequent temporal relationship. This leaves some ambiguity to the rules found and may make it difficult to apply those rules.

Another example of mining for rules without the use of a template is presented by Mannila et al. [DLM⁺98]. The input data set is a numeric sequence or set of numeric sequences. All of these sequences share the same time line. A whole data set here could be translated into a single instance for AprioriSetsAndSequences. Each sequence is partitioned into windows, the size of which is a user specified parameter. The windows are clustered based on similarity and assigned an identifier. The original sequences are then partitioned and these partitions are labeled with the cluster identifiers. ASAS relies on events being discovered in the sequences as a preprocessing step. This is more flexible allowing any event detection method to be used. Rules found from this data are in the form $A \Rightarrow B$ where B follows A in T time units. ASAS finds more general rules that include any temporal relationship between A and B.

There is another track of work to find interesting patterns in sequences. Mining

for patterns in sequences is related to mining temporal association rules. Srikant and Agrawal [SA96] use a modified Apriori algorithm to mine patterns from transactional sequences. These interesting patterns could be used to identify events in data sets for algorithms accepting events occurring over intervals.

Chapter 4

Event Attributes and Event Identification

4.1 Description of Sequence Attributes

	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
→ Disk	5	6	7	8	9	10	9	8	7	5	3	2	2	5	5	5	2	2	3	4	5	6	7	7	7	7	7	7	7	7

Figure 4.1: A Sequence Example. Here the top row represents a basic time line without units. The bottom row contains the values of the sequence for the CPU attribute.

AprioriSetsAndSequences deals with two data types not commonly found in data mining systems, time sequences and events. A time sequence is an ordered list of values, each associated with a time of occurrence along a time line specific to the data instance the sequence is a member of. Figure 4.1 shows such a sequence. The top row of numbers is the time line. The bottom row of numbers are the values in the sequence.

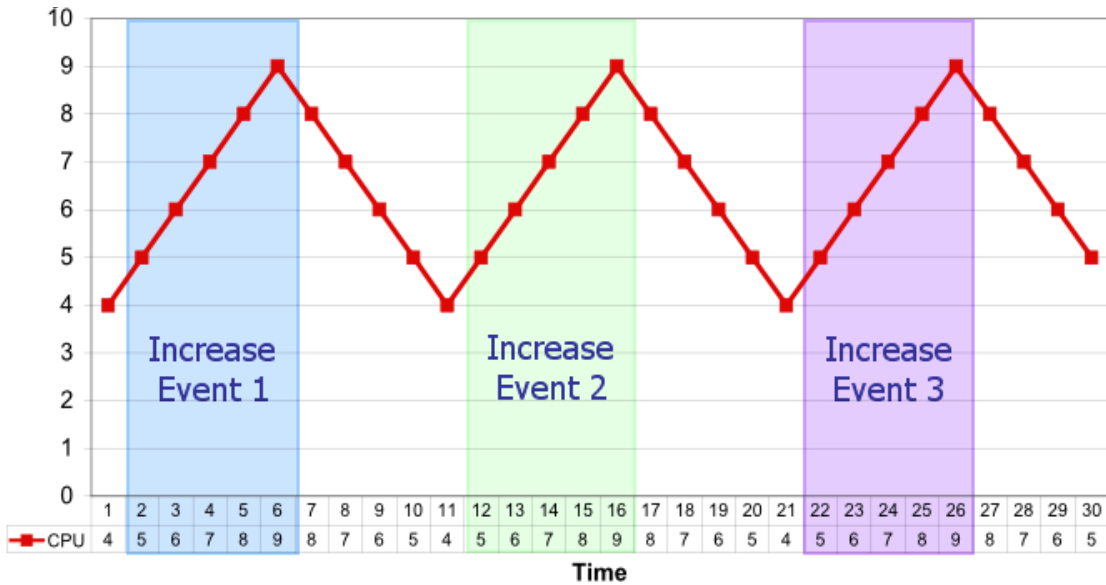
A time sequence has a real time line associated with it. This time line can be explicit or implicit. The values in an explicit real time line can be in any units and the distance between each point on the time line does not need to be uniform. This time line represents the actual time at which a value in a time sequence was observed. The events that are identified in a time sequence use the values along the real time line for their begin and end times. If an explicit time line is not present an implicit time line can be assumed. Units are ambiguous unless known to the user and the length between each point on the line is assumed uniform.

4.2 Description of Event Attributes

AprioriSetsAndSequences mines temporal associations directly from events. This keeps intact the temporal information represented in the data set while eliminating much of the work involved in scanning the actual sequences during mining. The events are akin to indexes into the sequences.

$$\text{CPU-Increase} = [t=2, t=6]$$

An example event is shown above. An event consists of three pieces of information. First is the time sequence attribute in which the event occurred. In this case the event can be found in the CPU sequence attribute. Second is the name of the event. An Increase event was found in the CPU sequence. the Increase event is described in Section 4.3. Third is the period of time during which the event occurred. This CPU-Increase event begins at time $t=2$ and ends at time $t=6$. The time sequence attribute name and the event name together define the event item type. the event type for this example is "CPU-Increase". Time sequence attributes can also be contained in the data set but they are not used directly during association rule mining.



$$\text{CPU-Increase} = \{ [t=2, t=6], [t=12, t=16], [t=22, t=26] \}$$

Figure 4.2: Generating Item Types

Figure 4.2 shows the time sequence attribute CPU. Highlighted are the Increase events found in this sequence. To represent these events as input for AprioriSetsAndSequences

an event set attribute is created. Above is an example of a CPU-Increase event set attribute. Instead of specifying the begin and end time for a single event as in the previous example, here there are a set of begin and end time pairs. This event set attribute will be named for the event type, CPU-Increase. In Figure 4.2 the value of the CPU-Increase attribute for the instance shown is written as it would appear in an ARFF file. Figure 4.3 shows the data sample first shown in Figure 1.1 but with the CPU sequence attribute replaced with the newly formed CPU-Increase event attribute. Details on the ASAS extension to ARFF can be found in Section 5.1.

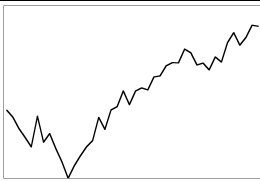
ID	flags	CPU-Increase	CPU (MHz)	memory %	memory (kB)	algorithms
1	{-R, -H}	{[t=2, t=6], [t=12, t=16], [t=22, t=26]}	600		523760	{neural net, backpropagation}
2	{-H}	{[t=4, t=26]}	600	10, 26, 46, 69, 86, ...	523760	{C4.5}
3	{-U, -R}	{[t=22, t=106], [t=202, t=207]}	300	19, 50, 82, 80, 70 ...	260592	{naive Bayes}
...

Figure 4.3: Here is the sample data set with the CPU sequence attribute replaced with the CPU-Increase event attribute.

4.3 Event Identification

An event has a begin and end time that annotates its occurrence along a time line specific to the data instance the event is a member of. All the events of a single type that occur in a single instance are stored as an event set attribute. An event set is simply a set attribute where the elements are all events of the same type. A type is a time sequence attribute and event type pair.

All the time sequences from the sample domain, computer system performance, are numeric. In order to demonstrate AprioriSetsAndSequences, events must be detected in these numeric time sequences. The events used are Increase, Decrease, and Sustain. They are described below and shown in Figures 4.4, 4.5, and 4.6. These are very simple. They are simple so the results can be easily understood. They can be used in conjunction to describe any numeric time sequence.

Increase event is defined as a sequence of values where each subsequent value is larger than the one before it.

Decrease event is defined as a sequence of values where each subsequent value is smaller than the one before it.

Sustain event is defined as a sequence of values where each subsequent value does not differ from the mean of the values before it by a user set tolerance.

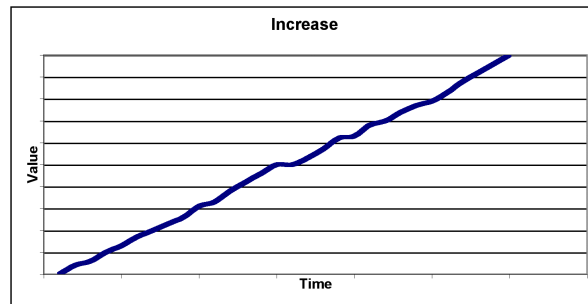


Figure 4.4: Increase Event

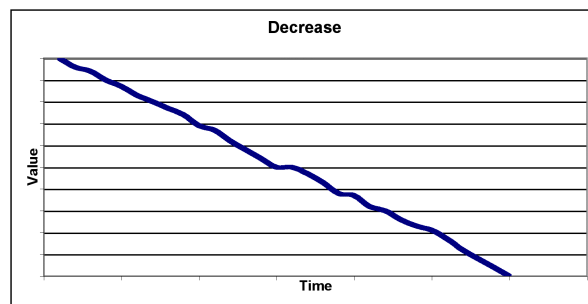


Figure 4.5: Decrease Event

It is important to note that the events detected should have some meaning to the user and to the domain from which the data set comes. These events will be used in the association rules produced. If their meaning is not clear to the user the rule will not benefit the user.

The simple event detection was implemented as a Weka filter. Filters are the main means for preprocessing data in Weka. The event filter reads in a data set containing time sequence attributes. For each time sequence attribute and each type of event pair, it adds an event attribute if that event type is detected in the time sequence. The individual events' begin and end time is noted according to the real time line for a particular instance.

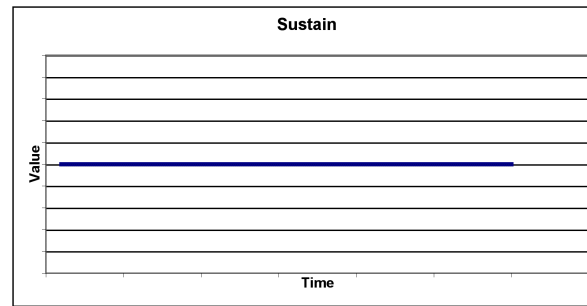


Figure 4.6: Sustain Event

More advanced event detection based on methods discussed in Section 2.3 can be implemented. An event detection tool was developed by Holmes and Leung [HL03] which extended the dynamic programming approach to time warping presented by Berndt and Clifford [BC95]. That tool is used to identify events in the stock market data appearing in Section 8.2.

Chapter 5

Data Representation

5.1 Extension of ARFF

Weka uses the ARFF data file format. There is no provision for time sequence attributes, event attributes, or set attributes in ARFF or Weka. Work done at WPI [Sho01] has incorporated set attributes into the ARFF file format by overloading the string data type. Weka ignores the set specific notation and treats the value as a string with no other meaning. This same approach was used to store time sequence attributes and events in ARFF.

```
@relation 'A sample ARFF file with sequences'

@attribute 'CPU(MHz)' numeric
@attribute algorithm <back-propagation, naive-Bayes, C4.5>
% the following attribute will be a sequence
@attribute CPU string

@data
600, back-propagation, '4:5:6:7:8:9:8:7:6:5:4:5:...7:6:5'
...
```

Above is an ARFF sample containing a sequence attribute CPU. The full sequence is the same present in Figure 4.1. A time sequence is stored as a list of values each delimited by a colon ' : '. Each value corresponds to the time value in the same position as in the time line sequence. This detail is only important to event detectors that

concern themselves with the absolute time at which certain values occur when detecting events. For many purposes the relative order of the time sequence values is enough. This format allows for more complicated representations such as a nonuniform time line where increments in time are not consistent.

An event uses the same delimiter as a sequence but there are only two values in the sequence. The first is the begin time of the event and the second is the end time of the event. In this work the convention of using a caret '^' has been adopted to delimit values in a set attribute. An event attribute would then have zero or more events represented as sequences of two values each delimited by carets.

$$\text{CPU-Increase} = \{ 3:8 \wedge 13:18 \wedge 23:28 \}$$

Shown above is the example of an event attribute for the CPU time sequence attribute shown earlier. The CPU time sequence attribute is the percentage of CPU usage in the overall computer system. This event attribute specifies when the CPU usage increases. Increases occur from time 3 to time 8, 13 to 18, and 23 to 28. Below is the sample ARFF file with the CPU-Increase attribute.

```
@relation 'A sample ARFF file with sequences'

@attribute 'CPU(MHz)' numeric
@attribute algorithm <back-propagation, naive-Bayes, C4.5>
% the following attribute will be a sequence
@attribute CPU string
@attribute CPU-Increase string

@data
600, back-propagation, '4:5:6:...7:6:5', '{3:8 ^ 13:18 ^ 23:28}'
...
```

5.2 ASAS Item Sets

The attribute value format of the data set is common among data mining tools. In order to mine association rules from this data it is translated into an item representation. Each possible attribute value pair that exists in the data set is defined as an item and given

an integer number called the item number. In AprioriSetsAndSequences there are two kinds of items. Regular items are formed from attribute value pairs where the value is not a sequence or an event set. The regular items are uniquely identified by their integer number. Event items are formed from attributes whose values are event sets. While event items are given an item number it is still necessary to interpret the item's value during the association rule mining process.

In the item sets of ASAS a second time line is used in addition to the real time line. This is the relative time line. The relative time line always begins at time 0. There are no units implied. The distance between each point on the relative time line is always 1. Each point on the relative time line corresponds to the begin times or end times of one or more events in the item set.

All the event items in an item set have a begin and end time. The begin and ends times of events are often treated as individual pieces of information. The begin and end times are sorted in ascending order with no regard to which event the begin or end times belong to. They are not treated as pairs for sorting purposes.

The relative time for each begin and end time are simply the position each holds in the sorted list. If the user requires absolute time to be used in mining the temporal associations events can be detected that are specific to the length of events or time between events.

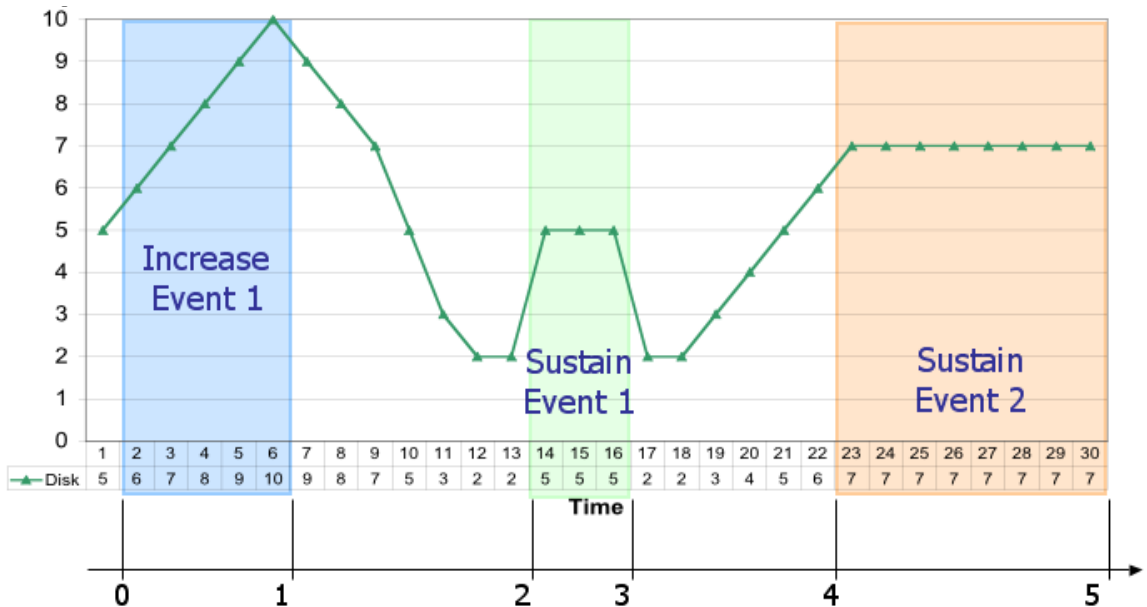


Figure 5.1: An item set containing 3 events. Disk Increase $[t_0, t_1]$, Disk Sustain $[t_2, t_3]$, Disk Sustain $[t_4, t_5]$

Figure 5.1 shows the events in an item set and its real and relative time lines. The item set depicted textually is shown below.

Real Time

{ Disk Increase [t=2, t=6], Disk Sustain [t=14, t=16], Disk Sustain [t=23, t=30] }

Relative Time

{ Disk Increase [t₀, t₁], Disk Sustain [t₂, t₃], Disk Sustain [t₄, t₅] }

Here there are three events in the item set, a Disk-Increase Event, and two Disk-Sustain Events, 1 and 2. On the real time line the first event, Disk-Increase Event begins at time t=2 and ends at time t=6. The real begin and end times for Disk-Sustain Event 1 are t=14 and t=16, respectively. The real begin and end times for Disk-Sustain Event 2 are t=23 and t=30, respectively. On the relative time line Disk-Increase Event begins at time t₀ and ends at time t₁. Note, real times are noted by t= to some real value. Relative times are denoted by t₋ where the digit following is the relative position. The relative begin and end times for Disk-Sustain Event 1 are t₂ and t₃, respectively. The relative begin and end times for Disk-Sustain Event 2 are t₄ and t₅. Adjusting the relative time line when adding an event item to an item set containing event items is a simple procedure.

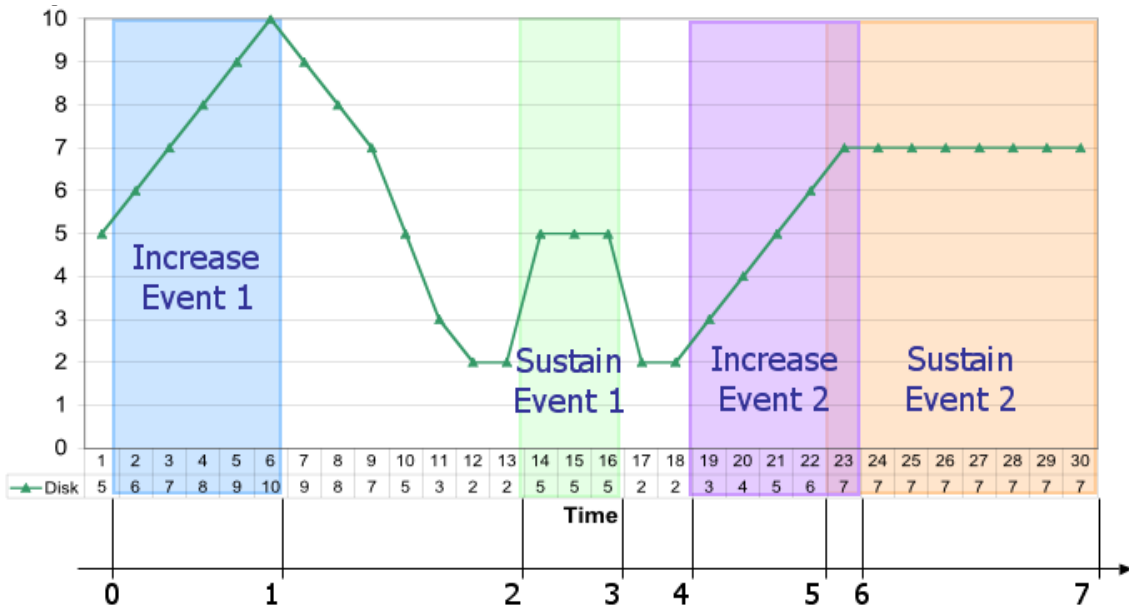


Figure 5.2: Adding the Increase Event 2 to item set in Figure 5.1

Let us add a second Disk-Increase event to this item set. Disk-Increase Event 2 has real begin and end times of $t=19$ and $t=23$. The begin and end times for the new event item are inserted into the sorted list of begin and end times according to the real time line. The begin and end time for each event now in the item set is then renumbered with the relative time according to their order in the list. Figure 5.2 shows the item set after an event is added along with the renumbered relative time line. As shown the relative times for Disk-Increase Event 1 and Disk-Sustain event 1 are the same. The relative times for the new event, Disk-Increase Event 2 are t_4 and t_6 . The new relative times of Disk-Sustain Event 2 are t_5 and t_7 . These relative times clearly show the temporal relationship between these two events. Disk-Increase Event 2 overlaps Disk-Sustain Event 2.

The data set instances share the same data representation as the item sets in ASAS. They both contain the same meta data about event items and the same code is used to store each. Functionality described for item sets is the same for data set instances.

5.3 Item Set Data Structures

In Apriori an item set is simply a list of integers. Each integer is a reference to a specific literal found in the data set. The list is commonly stored in an array.

In AprioriSetsAndSequences an item set is also a simple array of integers representing a set of items. However, some of these items may be events. Events occur during intervals of time and these intervals are used to determine if an item set is frequent. Rather than retrieving the literal for an event item and parsing it to discover the time it begins and ends every time the item set is compared to another item set or to an instance from the data set, the begin and end times are stored in an item set when the event is added to the item set.

Take the item set A shown below.

$$A: \{ \text{Disk Increase } [t=5, t=25], \text{CPU Increase } [t=10, t=40] \}$$

The Disk Increase event beginning is the first thing to happen in this item set. Secondly, the CPU Increase event begins. Thirdly, the Disk Increase event ends. Fourthly, the CPU Increase event ends. If we number, starting from 0, in chronological order, the begin and end of the events in item set A we obtain the relative times shown below.

$$A: \{ \text{Disk Increase } [t_0, t_2], \text{CPU Increase } [t_1, t_3] \}$$

ASAS uses an array to represent an item set's relative time line. Each element in the array represents a unit of time along the relative time line as denoted by its index. Stored

at each index of the relative time line array is a list of the begin and end of events that occur during that time. The relative time line array for item set A is shown in Figure 5.3.

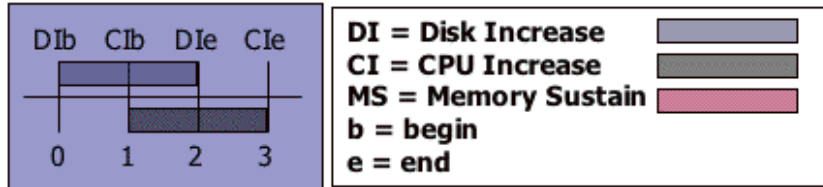


Figure 5.3: Relative Time Line Array. DI = Disk Increase, CI = CPU Increase, b = begin, e = end.

Item set A with its new relative times is shown below.

A: { Disk Increase $[t_1, t_3]$, CPU Increase $[t_2, t_4]$, Memory Sustain $[t_0, t_4]$ }

Let us see what happen to the relative time line when we add the event item Memory Sustain $[t=3, t=40]$ to item set A.

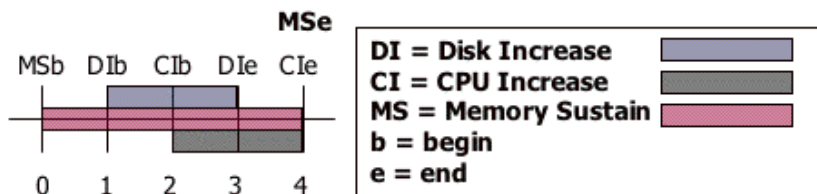


Figure 5.4: Relative Time Line Array. DI = Disk Increase, CI = CPU, Increase, MS = Memory Sustain, b = begin, e = end.

Since the Memory Sustain event begins at real time 3, before the real begin time of the Disk Increase event, the relative time line which always begins at 0, labels the Memory Sustain begin as relative time 0. The Memory Sustain event ends at the same real time that the CPU Increase event ends. Notice how the list of labels for relative time 4 contains both the end label for the CPU Increase event and the end label for the Memory Sustain event. Lastly, let us add another Disk Increase event to item set A. We will number the Disk Increase event so we don't confuse it with the first. This event will be Disk Increase 2 $[t=1, t=45]$.

Item set A is shown below.

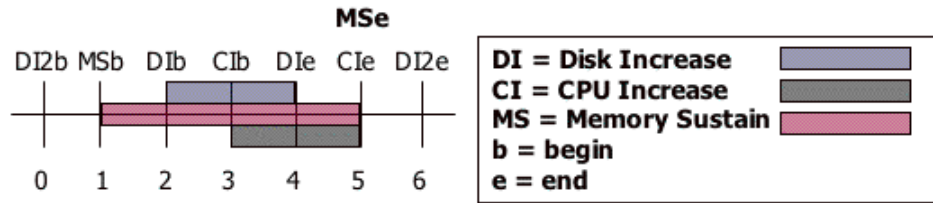


Figure 5.5: Relative Time Line Array. DI = Disk Increase, CI = CPU, Increase, MS = Memory Sustain, b = begin, e = end.

A: { Disk Increase $[t_2, t_4]$, CPU Increase $[t_3, t_5]$, Memory Sustain $[t_1, t_5]$,
Disk Increase 2 $[t_0, t_6]$ }

Although in this example we used the term “Disk Increase 2” to refer to the second disk increase event in the ASAS implementation the two disk Increase events are of the exact same type. at the beginning of this section it was said that the items in an item set are stored as an array of integers. The integers representing Disk Increase and Disk Increase 2 are different. The string literal attribute name for the items are the same, namely “Disk Increase”. Rather than parse the string literal each time to reason temporally or even discern if an item is indeed an event, this information is saved inside the item set itself when the event item is added. The type or attribute name of the event is stored in a hash table with the integer value as the key. To tell if an item is an event in a particular item set one has simply to ask the hash table if that item integer key is present.

Chapter 6

ASAS Algorithm

6.1 Input

AprioriSetsAndSequences takes as input a data set consisting of instances. Each instance has a set of attributes. Each attribute can be of type nominal, set, or event set. If temporal associations are to be mined the data set must contain event sets. An event set is simply a set whose elements are events. The standard minimum support and minimum confidence parameters are also specified as input. ASAS also takes a variety of other parameters controlling the form of the rules shown as output to the user. These include the attribute(s) required to appear in the antecedent or consequent of the rules, the size of the antecedent and consequent, the minimum number of rules, and the option to iteratively decrease support if the minimum number of rules are not found. Details on these options can be found in Appendix A.

6.2 Frequent Item Set Generation

- 1: Given a data set of instances DS , and minimum weight $minW$
 {Size $k = 1$ Candidate Generation (see Section 6.2.1)}
- 2: **for all** regular items i in DS **do**
- 3: create candidate item set c of size $k = 1$
- 4: add i to c
- 5: add c to candidate list C
- 6: **end for**
- 7: List of event types already seen $ET = \emptyset$
- 8: **for all** event items e in data set **do**

```

9:   if event type of  $e$  not present in event type list  $ET$  then
10:     create candidate item set  $c$  of size  $k = 1$ 
11:     create a new event item  $ei$  with event type of  $e$  and begin time = 0 and end time
        = 1
12:     add  $ei$  to  $c$ 
13:     add  $c$  to  $C$ 
14:     add event type of  $e$  to  $ET$ 
15:   end if
16: end for
    {Size  $k = 1$  Candidate Support Counting (see Section 6.2.2)}
17: for all instances  $I$  in  $DS$  do
18:   for all  $c$  in  $C$  do
19:     if  $I$  contains the item in  $c$  then
20:       increment weight of  $c$ 
21:     end if
22:   end for
23: end for
24: for all  $c$  in  $C$  do
25:   if weight of  $c \geq minW$  then
26:     add  $c$  to frequent item sets of size  $k$  list
27:   end if
28: end for
29: remove all from  $C$ 
30: while frequent item set of size  $k$  list is not empty do
31:    $k = k + 1$ 
    {Size  $k \geq 2$  Candidate Generation (see Section 6.2.3)}
32:   for all pairs of item sets  $f1$  and  $f2$  in the frequent item sets of size  $k-1$  list do
33:     if  $f1$  and  $f2$  contain event items then
34:       generate 13 candidates, 1 for each possible temporal relationship between the
        event items  $f1$  and  $f2$  do not have in common
35:     else
36:       generate 1 candidate by combining  $f1$  and  $f2$ 
37:     end if
38:     add generated candidate(s) to  $C$ 
39:   end for
    {Size  $k \geq 2$  Candidate Support Counting (see Section 6.2.4)}

```

```

40:  for all instances  $I$  in  $DS$  do
41:    for all  $c$  in  $C$  do
42:      if all regular items  $i$  in  $c$  are included in  $I$  then
43:        if mapping exists between all event items  $ei$  in  $c$  to event items in  $I$  such
          that all temporal relationships are the same then
44:          increment weight of  $c$ 
45:        end if
46:      end if
47:    end for
48:  end for
49:  for all  $c$  in  $C$  do
50:    if weight of  $c \geq minW$  then
51:      add  $c$  to frequent item sets of size  $k$  list
52:    end if
53:  end for
54:  remove all from  $C$ 
55: end while

```

6.2.1 Level 1 Candidate Generation

In regular Apriori the first level of candidate item sets is generated by simply creating an item set of size one for each of the items appearing in the data set. These item sets are added to a list of candidate item sets. The number of instances in the data set that contain the item sets in this list will be counted. This count is used to calculate the support of each item set. Only those item sets that have a support equal to or higher than the minimum support specified by the user will be kept to generate the next level of candidates.

This basic process is used in `AprioriSetsAndSequences` with some important differences. For each regular item in the data set an item set of size one is still created and the item added to it. This newly generated item set is then added to the list of candidate item sets.

Event items found in the data set are handled differently. A candidate item set is not generated for each event item. Event items have a type, usually a name that combines the original time sequence attribute name and the event detected in the sequence. There are usually multiple event items of a single type in a data set. Each event item may have a unique begin and end time. If a candidate item set were created with these event

items with unique begin and end times they would be too specific and a frequent item set containing them would not likely be found. Instead, a representative event item for each event type is created. This new item is added to a candidate item set.

For the data set consisting of a single instance shown below we will generate the level 1 candidates.

Data Set:

CPU Increase = { [t=3, t=8], [t=13, t=18], [t=23, t=28] }

Disk Increase = { [t=2, t=6], [t=12, t=23] }

Disk Sustain = { [t=14, t=16], [t=23, t=30] }

Memory Increase = { [t=3, t=8] }

Memory Sustain = { [t=11, t=26] }

Level 1 Candidate Item Sets:

1. { CPU Increase [t₀, t₁] }
2. { Disk Increase [t₀, t₁] }
3. { Disk Sustain [t₀, t₁] }
4. { Memory Increase [t₀, t₁] }
5. { Memory Sustain [t₀, t₁] }

6.2.2 Level 1 Counting Support

The support of an item set is the percentage of instances in the data set that contain the item set. The weight of an item set is the number of instances that contain it. The weight of an item set is counted and then used to calculate support.

In regular Apriori determining if an item set is included in an instance is a straight forward procedure. The list of items in the item set is compared to the list of items in each data set instance. If all items in the item set are found in the instance that instance counts towards the weight of the item set.

Determining if an item set is included in a data set instance in AprioriSetsAndSequences is not as straightforward. In a data set instance there may be many event items of one type. The event items in the instance that match the event items in the item set must be chosen so the temporal relationships between them are the same. When an item set containing event items is included in an instance there exists a mapping between the matching event items from the item set to the instance.

Mapping event items is trivial when the item set contains a single event item. This is because the relative begin and end times are always 0 and 1, respectively. As long

as the instance contains an event item of the same type a mapping is guaranteed. This is the case for all item sets of size one containing one event item. Figure 6.1 illustrates a mapping between an item set containing a single Disk Increase event and a data set instance.

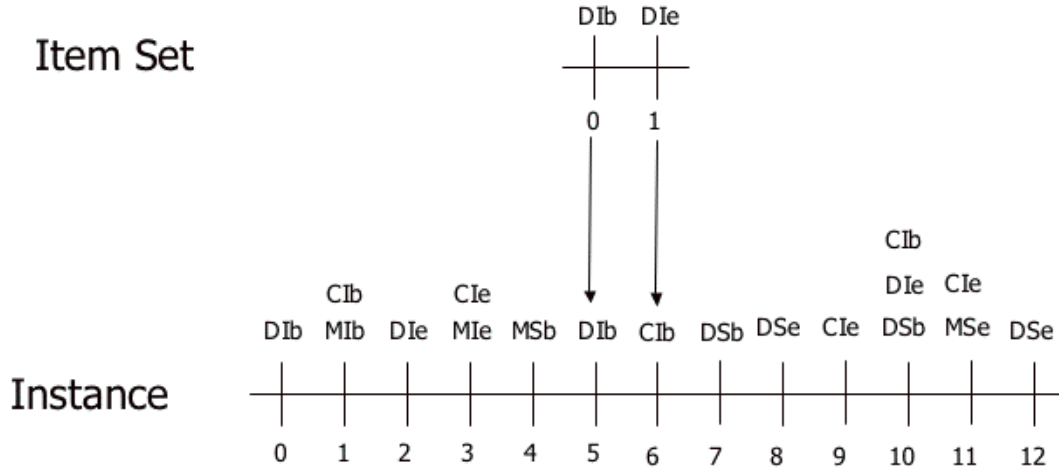


Figure 6.1: Level 1 Mapping. DI = Disk Increase, CI = CPU Increase, b = begin, e = end.

Even though the example in Figure 6.1 shows a mapping to a specific pair of times in the instance any pair of times in which a Disk Increase event begins and ends would suffice. This mapping is not actually done during Level 1 since a single event item is guaranteed a map to an instance which contains at least one event of the same event type.

6.2.3 Level 2 Candidate Generation

In regular Apriori, candidate item sets of size two are generated by combining each pair of frequent item sets of size one. Combining item sets of size one is a simple matter since each pair results in a valid candidate. The same is true for AprioriSetsAndSequences. The difference is that for each pair of event items there exists thirteen different item sets that represent the different temporal relationships [All83] those two event items can have together. Figure 6.2 illustrates these relationships and corresponding relative times.

There being thirteen possible temporal relationships between any two events is a very important issue when mining for temporal association rules. The search space in this problem is much larger than regular Apriori and every effort must be made to make it efficient as possible to obtain results in a reasonable amount of time.

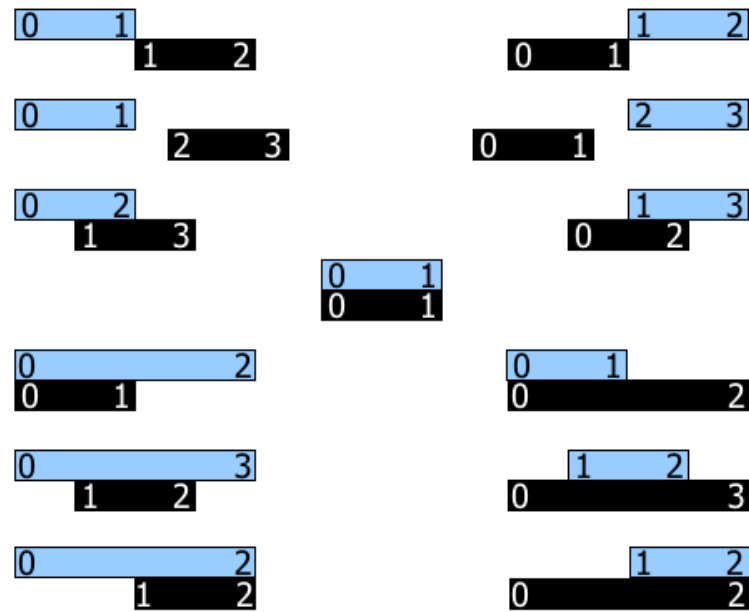


Figure 6.2: All Possible Temporal Relationships Between Two Events. Each event is depicted by a rectangle (one dark and one light). Numbers in the rectangles denote relative times t_0 , t_1 , t_2 , and t_3 .

6.2.4 Level 2 (and up) Counting Support

Candidate item set weight is counted in Apriori in the same manner as described in Section 6.2.2 Level 1 Counting Support. For AprioriSetsAndSequences the process is different. Each instance in the data set is still compared to each candidate item set in the candidate list. First, the instance must contain all the regular items in the item set. If they are all present, inclusion of the event items are checked. If the item set contains only one event item the checking is trivial as described in Section 6.2.2 Level 1 Counting Support. If the one event item is found the weight of the item set is incremented.

If the candidate item set contains two event items (or more in subsequent levels) then a mapping must be made from event items in the item set to event items in the instance. The relative begin and end times of event items in item sets and instances are stored for quick reference in a item set's or instance's relative time line as described in section 5.3. A mapping between relative times in the item set to those in the instance represents a partial possible mapping between the begin and end labels of the event items, and hence a mapping between the event items themselves.

Let us map an item set containing a Disk Increase event and a CPU Increase event into the instance used in the Level 1 Support mapping example. This mapping is shown

in Figure 6.3.

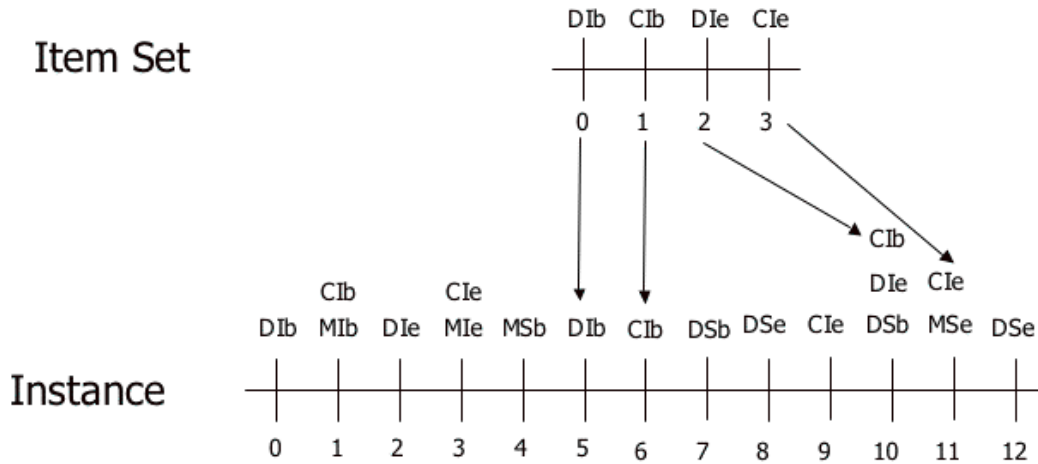


Figure 6.3: Level 2 Mapping. DI = Disk Increase, CI = CPU Increase, b = begin, e = end.

Note how the event items in the instance which are mapped to from the item set have the same temporal relationships to each other as the corresponding event items in the item set.

A mapping is made between an item set’s relative times and an instance’s relative times by first matching relatives based on begin times of events in the item set. First the list of begin labels is retrieved from the item set’s relative time line array. Starting at time 0 in the instance, the earliest time index containing a list of begin labels for the same type of events as those in the item set is found. Once this corresponding time index is found it is noted as a possible map for that relative time 0 in the item set. The remaining time indexes in the item set’s relative time line are mapped in the same way, always starting at the next relative time in the instance.

Once a possible map is created it is tested using the end time labels. Begin and end time labels are paired together, belonging to a single event item in the item set. The corresponding begin and end labels in the instance must belong to a single event item as well. Furthermore, an event item in the instance can only be mapped to a single event item in the item set as the begin and end labels must.

Once the map is validated the weight of the item set is incremented. If the map is found to be invalid the last relative time used in the instance is skipped and another possible map of relative times is created and tested. This continues until there are no more relative times left to try in the instance.

6.2.5 Level 3 (and up) Candidate Generation

At the third level of candidate item set generation, generating possible frequent item sets of size 3, there are ways to eliminate some of the candidates before counting support. This saves time. The items in an item set are sorted in ascending order according to their item number. To combine two item sets to form a new item set one size larger there are a list of conditions that must be met. They must have the same number of items. The list of items in both item sets must be the same except for the last item which must be different. Furthermore, the last item in the second item set must have a higher item number than the last item in the first item set. If two item sets do not meet these criteria they are not combined to generate a candidate item set. The candidate item set formed by combining two item sets simply has the superset of the items in each. This is easily done by adding the last item of the second item set to the first item set.

Combining item sets has been modified for AprioriSetsAndSequences. All of the conditions still hold for regular items. The event items have to be handled as a special case. Since there is the possibility of there being more than one temporal relationship between the same set of event items the item number alone cannot be used to decide if two item sets should not be joined. Rather, a mapping must exist from all the event items in the first item set to all the event items in the second item set. This excludes event items that are the last item in an item set. Once this mapping is found the item sets can be combined. For an example of such a mapping see Figure 6.3.

As for level two candidate generation, it is possible for more than one candidate item set to be generated for each pair of frequent item sets that are combined. The algorithm for generating these possibilities is more involved than the straight forward iteration of thirteen possible temporal relationships. When more than two event items are present the number of temporal combinations grows. This number is limited during candidate generation by using the relative temporal information contained in each item set.

Consider combining the item sets A and B shown in Figure 6.4. These can be combined since they have the same number of items, a mapping exists between the Disk Increase event in item set A and the Disk Increase event in item set B , and the event items listed last in A and B are different. The temporal relationship between the CPU Increase event in A and the Memory Sustain event in B is not known. Some of the possible relationships can be eliminated by inferring information from the fact that the Disk Increase event in both A and B is the same event. Figure 6.5 illustrates this inference.

Combining the two item sets is done by adding the last item from the first item set to the second item set. In this example the event item Memory Sustain will be added to

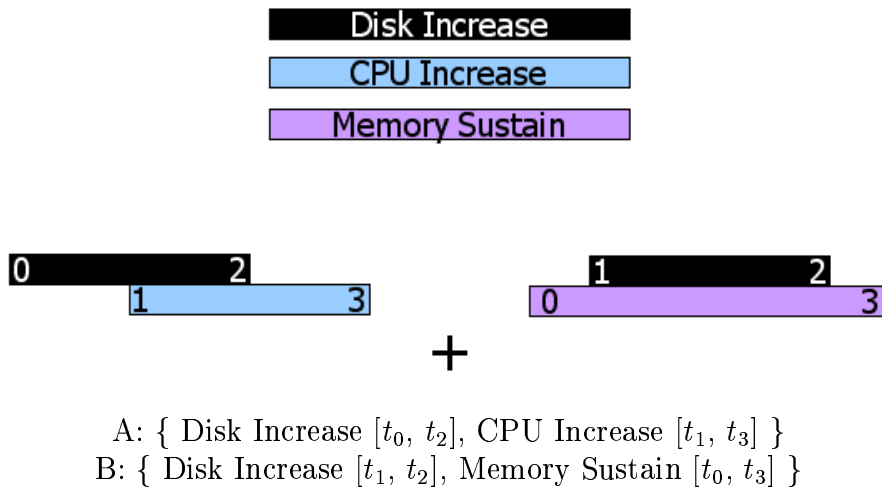


Figure 6.4: Combine Item Sets

item set A . First, the begin time of Memory Sustain event to be added to item set A must be determined. In item set B the Memory Sustain event began before the Disk Increase event began. This temporal relationship must hold for the item sets being generated. In the item set A there is nothing before the begin time of Disk Increase. Therefore the only relative time Memory Sustain can begin is one time line value before time 0.

Next the end time of the Memory Sustain event must be determined. From item set B it can be seen that the Disk Increase event ended before the Memory Sustain event ended. This means that each relative time in the item set A that occurs after the end of the Disk Increase event is potentially when the end of the Memory Sustain event occurs. This includes the times noted in the time line for A and each time in between those marked times. Figure 6.2 illustrates how two events can be related in time.

The possible begin and end time pairs determined are in relation to item set A 's existing relative time line. A candidate item set is generated for each pair. In these candidate item sets the relative time line will be renumbered starting from 0 to include the Memory Sustain event as shown in Figure 6.5. The resulting candidate item sets are shown in Figure 6.6

By inferring temporal relationships from the event items in common between the item sets that are being combined many candidate item sets representing different temporal relationships can be eliminated and hence save time during mining. An important observation to make is that this reasoning is only necessary when each item set has more than one event item. If each item set contains exactly one event item each the generation of candidates follows the procedure described in Section 6.2.3 Level 2 Candidate

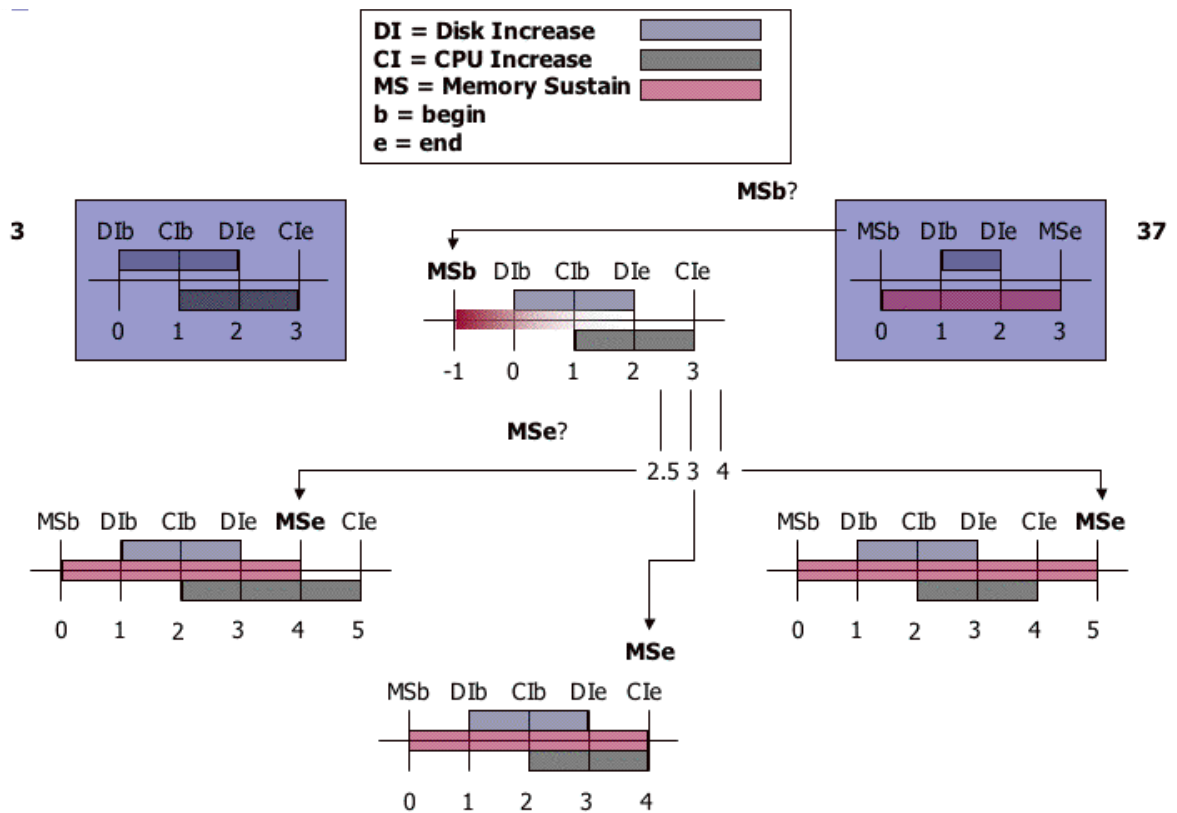


Figure 6.5: Selective candidate generation by joining events A and B from Figure 6.4

- {Disk Increase $[t_1, t_3]$, CPU Increase $[t_2, t_5]$, Memory Sustain $[t_0, t_4]$ }
- {Disk Increase $[t_1, t_3]$, CPU Increase $[t_2, t_4]$, Memory Sustain $[t_0, t_4]$ }
- {Disk Increase $[t_1, t_3]$, CPU Increase $[t_2, t_4]$, Memory Sustain $[t_0, t_5]$ }

Figure 6.6: Selective Candidate Generation Result

Generation.

6.2.6 Multiple Events of The Same Type

It may be that a user wishes to find association rules that contain multiple events of the same type. As explained in Section 6 rules of this form will not be found. This is due to a single event item being created to represent all event items in the data set of one specific type. when frequent item sets are mined only this representative event item can appear and hence only that representative event item will ever make its way into an association rule.

To overcome this limitation of the Basic ASAS algorithm we allow the user to select the maximum number of representative event items to be created during the process described in Section 6.2.1. For example, if a user wants to find association rules that contain two Memory-Sustain events, the maximum events of the same type parameter must be set to 2 or higher.

Instead of just: item set # 1. { Memory-Sustain $[t_0, t_1]$ }
we create:

item set # 1. { Memory-Sustain $[t_0, t_1]$ }
item set # 2. { Memory-Sustain $[t_0, t_1]$ }

so that later when these item sets are combined to form Level 2 we can get item sets that contain two Memory Sustain events:

{ Memory-Sustain $[t_0, t_1]$, Memory-Sustain $[t_2, t_3]$ }
{ Memory-Sustain $[t_0, t_2]$, Memory-Sustain $[t_1, t_3]$ }

...

to represent the 13 temporal relationships two events can have as depicted in Figure 6.2.

The number of event items of each event type are counted. For each event type a number of new event items are created. This number is the maximum number of event items of the same type specified by the user or the number of event items found in the data set of that type, whichever is smaller.

The more event items of the same type allowed, the more event items there are to count support for and use to generate new candidate item sets. The higher the maximum is set the more work there is to mine, impacting performance negatively. However, the higher the maximum is set the more potentially expressive the rules found could be.

Since the implementation of ASAS incorporating multiple event of the same type a more efficient method has been devised but not implemented. Please see Section 9.2.2 for details.

6.2.7 Duplicate Item Sets

Having multiple event items of the same type creates conceptually identical item sets during candidate generation and in the frequent item set list. Take the item sets 1, 2 and 3 shown below.

1: { CPU Increase $[t_0, t_1]$ }
2: { Disk Increase 2 $[t_0, t_1]$ }
3: { Disk Increase 1 $[t_0, t_1]$ }

Let us combine these to create candidate item sets of size 2. The results are shown in Figure 6.7, Figure 6.8, and Figure 6.9.

Level 1 Frequent	Level 2 Candidates
1. CPU Increase $[t_0, t_1]$	1. Disk Increase 1 $[t_0, t_1]$ & CPU Increase $[t_1, t_2]$
2. Disk Increase 1 $[t_0, t_1]$	2. Disk Increase 1 $[t_0, t_1]$ & CPU Increase $[t_2, t_3]$
3. Disk Increase 2 $[t_0, t_1]$	3. Disk Increase 1 $[t_0, t_2]$ & CPU Increase $[t_1, t_3]$
	4. Disk Increase 1 $[t_0, t_2]$ & CPU Increase $[t_0, t_1]$
	5. Disk Increase 1 $[t_0, t_3]$ & CPU Increase $[t_1, t_2]$
	6. Disk Increase 1 $[t_0, t_2]$ & CPU Increase $[t_1, t_2]$
	7. Disk Increase 1 $[t_1, t_2]$ & CPU Increase $[t_0, t_1]$
	8. Disk Increase 1 $[t_2, t_3]$ & CPU Increase $[t_0, t_1]$
	9. Disk Increase 1 $[t_1, t_3]$ & CPU Increase $[t_0, t_2]$
	10. Disk Increase 1 $[t_0, t_1]$ & CPU Increase $[t_0, t_2]$
	11. Disk Increase 1 $[t_1, t_2]$ & CPU Increase $[t_0, t_3]$
	12. Disk Increase 1 $[t_1, t_2]$ & CPU Increase $[t_0, t_2]$
	13. Disk Increase 1 $[t_0, t_1]$ & CPU Increase $[t_0, t_1]$

Figure 6.7: Candidates from Combining Item Sets 1 and 2

Clearly the candidate item sets depicted in Figure 6.7 and those in Figure 6.8 are conceptually the same. Let us refer to those item sets 1 through 13 as the originals and item sets 14 through 26 as the duplicates. In order to minimize the time spent counting support for these duplicate item sets we put them aside and do not count them. We can do this because those duplicate item sets are guaranteed to have the same support as the item sets 1 through 13, respectively. All we need is a way to look up which item set the uncounted item set is a duplicate of. This can be done by using a hash table where the duplicate item set is the key and the original item set is the value.

To make sure the duplicate item sets do not get counted they are stored in a holding list called *all - candidates*. The original item sets are stored in a list called simply *candidates*. When a candidate is generated it must be determined if it is a duplicate of an item set previously generated. During the first phase of ASAS implementation this was done brute force searching through the candidate list for a duplicate. This was very expensive and did not save enough time overall.

An improvement was made by making the observation that the candidate item set generated from an item set that itself was a duplicate increased that chances that the

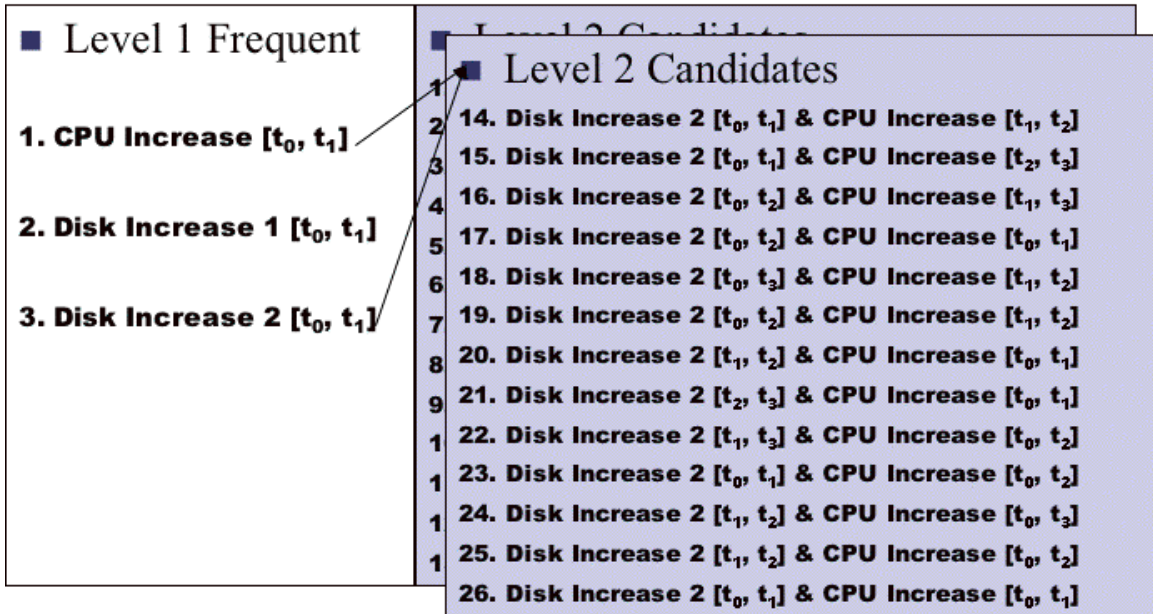


Figure 6.8: Candidates from Combining Item Sets 1 and 3

generated candidate item set would be a duplicate as well. The candidate list would only be searched when one or both of the parent item sets was a duplicate. Even though performance improved, it was still not enough.

Now we introduce the Item Set Prefix Tree data structure. This tree is like any other prefix tree in that the first item in a list is taken, removed from the list, and a branch is traveled based on that item’s value. This continues until there are no more items left and a leaf of the tree is reached. In our case, the leaf holds a reference to an item set. The one change made to this idea for the Item Set Prefix Tree is that for event items of the same type the same branch is followed even though the item number may be different. This means that once an item set is added to the Item Set Prefix Tree if a duplicate of that item set is created in the future it can be detected with a number of steps equal to the number of items in the item set. This greatly reduces the time to find duplicate item sets.

Let us take the candidate item sets 1 and 14 from our example above. They are shown below.

- 1. { Disk Increase 1 $[t_0, t_1]$ & CPU Increase $[t_1, t_2]$ }
- 14. { Disk Increase 2 $[t_0, t_1]$ & CPU Increase $[t_1, t_2]$ }

Let us add item set 1 to an Item Set Prefix Tree. The result is shown in Figure 6.10

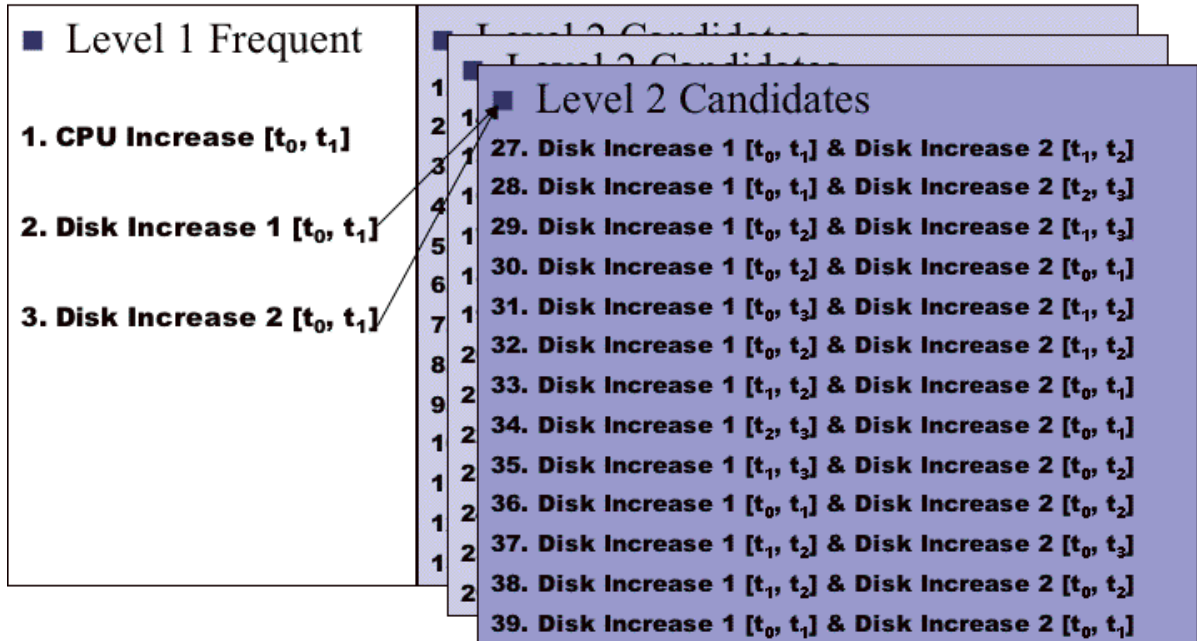


Figure 6.9: Candidates from Combining Item Sets 2 and 3

Let us now transverse the tree with item set 14. The first item in item set 14 is the Disk Increase 2 event. This is a Disk Increase event so we follow the Disk Increase branch. It has a relative begin time of 0 so we follow the 0 branch. It has a relative end time of 1 so we follow the 1 branch. The next item is the CPU Increase event. Clearly we can follow the CPU Increase event branch, begin time branch, and end time branch shown. This leads us to the leaf containing a reference to item set 1. Item set 14 has been determined to be a duplicate in two steps without having to compare it to all 13 item set generated before it.

6.3 Rule Generation: Calculating Confidence

Traditionally confidence is defined for a rule $A \Rightarrow B$ as the percentage of instances that contain A that also contain B. This is usually calculated as:

$$\text{Support } (AB) / \text{Support } (A)$$

Figure 6.11 shows an example of the association rule,

$$a[t_0, t_1] \Rightarrow b[t_2, t_3],$$

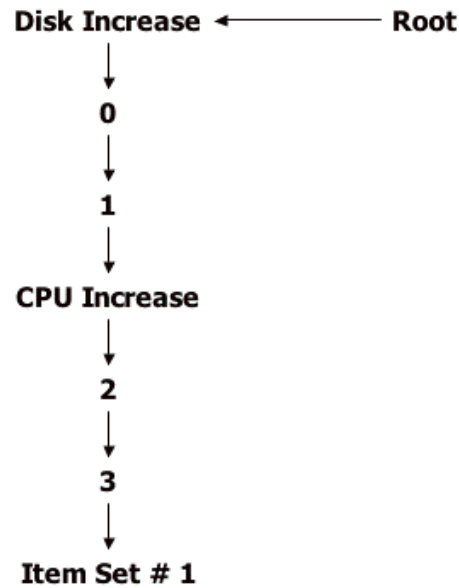


Figure 6.10: Prefix Tree

and a data set containing a single instance. The data set instance has six events all occurring in succession with no overlap. If the confidence of the association rule is calculated using the standard method, then the confidence for this rule is calculated as:

$$\text{Support (a } [t_0, t_1] \text{ \& b } [t_2, t_3]) / \text{Support (a } [t_0, t_1])$$

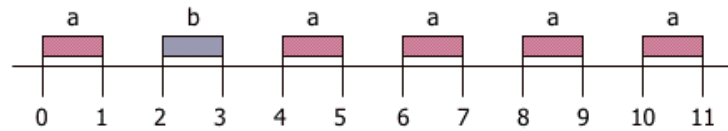
To get the support we simply count the number of instances that contain the item set and divide that count by the total number of instances in the data set. For the support of the antecedent we get 1, or 100%. for the support of the confidence we also get 1. this leads to a confidence of 100%. This means that the event a is followed by the event b 100% of the time in our data set. Looking at the instance in Figure 6.11 is it clear that only one a is followed by a b once. Since there are five a events, b follows a only 20% of the time. Clearly the standard measure of confidence is not adequate for complex temporal association rules.

Figure 6.12 illustrates the ASAS method of calculating confidence. The same association rule and the same instance are used as in the previous example. The goal of this method was to make confidence an accurate measure of the rule's success in predicting the presence of the consequent in an instance. Once the antecedent has been detected, the consequent, as represented temporally in the rule, should also be detected.

ASAS confidence introduces the concept of event weight. *Event weight* is simply the number of times the events in an item set occur in all the instances of the data set.

Confidence = **Support (Antecedent & Consequent) / Support (Antecedent)**

The only instance
in data set



a [t₀, t₁] => b [t₂, t₃]

Support (a [t₀, t₁] & b [t₂, t₃]) = 1

Support (a [t₀, t₁]) = 5

Confidence (a [t₀, t₁] => b [t₂, t₃]) = 1/5 = 20%

b follows a only 20% of the time

Standard calculation of Confidence is not adequate

Figure 6.11: Apriori Calculate Confidence

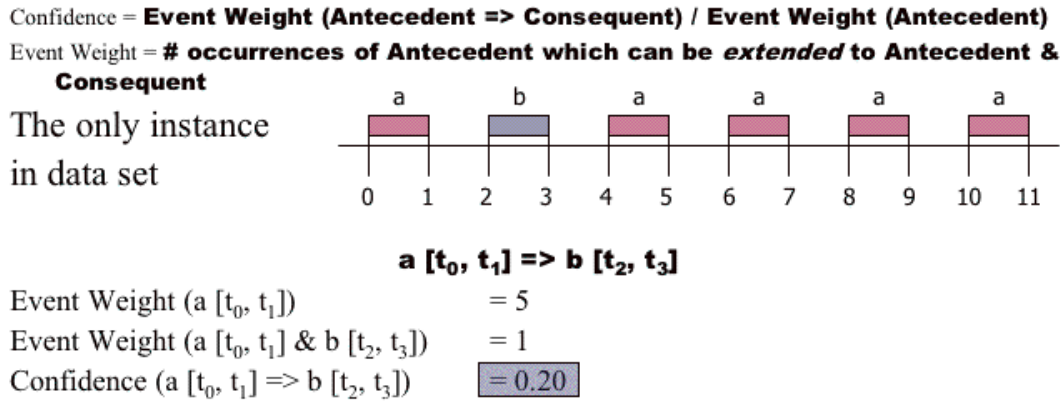
An important caveat comes into play if the item set contains non event items. Those non-event items, or regular items, must be present in an instance before we count the event weight that that instance contributes to an item set's event weight. If the regular items are not present we do not count the event weight of the event items in the item set from this instance.

Using event weight, the calculation for obtaining confidence is:

Event Weight (Antecedent \Rightarrow Consequent) / Event Weight (Antecedent)

Event Weight (Antecedent \Rightarrow Consequent) means we first find all occurrences of the Antecedent in the data set. We then take this list of occurrences, essentially a list of specific item sets, and try to extend each one to include the consequent. That is, from the instance in which the antecedent item set occurs we try to add other items to the item set until it contains not only the items in the antecedent but also the consequent. This newly formed item set must also represent the exact temporal relationships expressed in the association rule. The number of antecedent item sets we can extend to include the consequent is the event weight of (Antecedent \Rightarrow Consequent). In other words, the confidence of a rule is the percentage of occurrences of the antecedent that can be extended to the entire rule.

Let us calculate the confidence of our example rule. First we need to find the event weight of the antecedent. There are 5 occurrences of *a* as shown in Figure 6.13. Next we



ASAS calculation of Confidence correctly represents the data

Figure 6.12: ASAS Calculate Confidence

need to see how many of these can be extended to include the consequent. The temporal relationship expressed in the rule is that *b* must occur after *a* with no overlap. The first *a* does have a *b* that follows it as shown in Figure 6.14. The other 4 do not. Therefore we have an event weight of 1. This makes the confidence of the example rule 20%. Clearly this way of measuring confidence more accurately represents the data set, and hence the strength of the rule.

a [t₀, t₁] => b [t₂, t₃]

Find Antecedent:

Figure 6.13: Finding Antecedent

Since event weight is counted in relation to the antecedent of a rule being extended to encompass the consequent of a rule, the event weight counting is done during rule generation and not frequent item set mining.

a [t₀, t₁] => b [t₂, t₃]

**Extend to include
Consequent:**

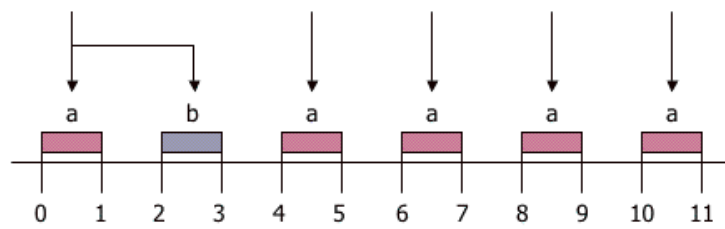


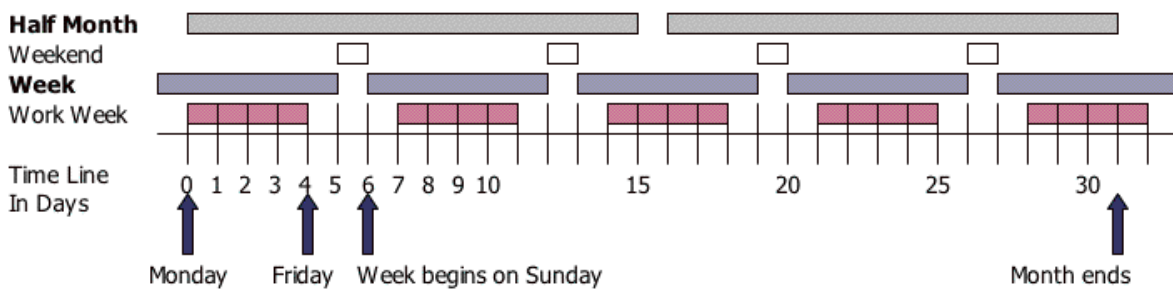
Figure 6.14: Extending Antecedent to Include Consequent

Chapter 7

ASAS Implementation

7.1 Data Preprocessing Filters

To detect events in sequential attributes an Event Filter was implemented and made available through the Weka System. The increase, decrease, and sustain events described in Section 4 are found using this filter. For details on this filter and its options please see Section A.



Time line serves as smallest granularity

Figure 7.1: Several Time Granularities

A Time Granularity Filter was added for creating different time granularities attributes for data sets containing events. Figure 7.1 shows several different time granularities. The time line has units of days. This forms the smallest granularity possible for this example. Days can be grouped into weeks, seven consecutive days starting on a par-

ticular day of the week, in this example Sunday. Granularities can be defined with gaps between them such as work week and weekend. Granularities can signify a percentage of time on the time line as does the half month granularity. For examples of association rules containing time granularity events please see Section 8.3.

There are two major types of granularities the filter currently supports. The first type of granularity is where granules are indistinguishable from each other. That is a granule's the absolute (real) position along the time line is not a defining characteristic. Only that granule's relative time can be used for temporal reasoning. Say we had an item set containing a single minute granule of this type and a CPU Increase event which occurs during the minute granule. When calculating confidence for a rule in which this item set was the antecedent, it would not matter where along the time line this granule appeared. Each occurrence of a CPU Increase and minute granule during which the CPU Increase happened would count towards the item set's event weight.

The second type of granularity supported is when each granule is uniquely identifiable. For example, each minute along the time line is numbered. There is the first minute, the second minute, and so forth. Take the example from above and make the minute granule of this type. Say it is the second minute along the time line. Only CPU Increase events occurring during the second minute of each instance would count towards the event weight of the item set. For an explanation of how event weight is counted see Section 6.3.

There are two current ways to decide the length of a granule in a granularity in the filter. The first is number of unit of time along the time line. A granule size of 60 would create a granule for each 60 units along the time line. If the time line was in units of seconds, each granule would correspond to a minute. If the time line was in units of days each granule would roughly correspond to two months.

The second way of dividing a time line into granules using this filter is specifying a percentage of the time line. this results in each instance having the same number of granules per granularity. Currently only a 50% granularity of this type is implemented. Once need arises the implementation can be easily converted to a generic percentage.

Two other filters created during this thesis were for constructing data sets. The first added a new attribute to a data set and allowed an attribute value to be specified for all the instances in that data set.

The second filter takes a data set where the instances are ordered chronologically and are assumed to be related. The filter outputs a data set containing a single instance. For each attribute in the input data set a sequence attribute is created in the output data set. The value for this sequence attribute for the single instance in the output data set

is comprised of the ordered values from the instances in the input data set.

7.2 Implementing Rule Generation

Rule generation as implemented in [Sho01, SS02] used a method similar to that of candidate item set generation to generate possible association rules which are then evaluated according to user specified parameters such as confidence, required items, minimum and maximum number of items, etc. This method being grossly inefficient was not apparent until event items were introduced to the system by ASAS. The reasoning necessary to manipulate item set containing events compounded this inefficiency to a point where rule generation never completed even after a week of run time, hence there being no comparisons of time between this method and the one implemented for ASAS.

The rule generation implementation used in ASAS starts with the maximal frequent item sets. Let us think an item set as a simple array of a certain length n . From these n indexes the algorithm will choose some number to be included in the antecedent of a rule and some number to be included in the consequent of a rule. This choosing is simply a permutation of all the number of items contained in the antecedent and of all the numbers of items contained in the consequent. For example if we had an item set of size $n = 3$ the possible respective sizes of the antecedent and consequent would be: $\{ (1,1), (1,2), (2,1) \}$. This says 1 item must be chosen for the antecedent and 1 item for the consequent. Then 1 item for the antecedent and 2 items for the consequent and so forth.

Choosing which items form the item set is also a simple permutation. For example, for the item set of size 3 there are 3 possible antecedents of size 1, first being the item as index 0, the second the item at index 1, and the third as index 2.

It is very easy and quick to generate this permutation of corresponding sizes of antecedents and consequents. It is just as easy to generate the index choices for those antecedent and consequent sizes. This can be thought of as generating a tree of rules. This methodology also has the added benefit of being able to prune rule generation according to the user specified criteria such that entire branches of the tree can be pruned and even more effort saved.

7.3 Support Counting Prune

An item set pruning step was implemented in ASAS during support counting whose benefit is not specific to ASAS alone. The number of instances in the data set and the

minimum support is known during mining. Therefore it is possible to prune a candidate item set before all instances are seen. This can potentially save many comparisons of item sets and instances. Let us consider a data set containing 100 instances and a minimum support of 60%. If a given item set is not found to be included in any of the first 41 instances, it is clear it will never meet the minimum support threshold. There is no need to compare it to the other 59 instances. In general, if at any point during support counting the minimum support count minus the current support of the item set is larger than the cardinality of the remaining set of instances, then the item set cannot be frequent.

7.4 Other Features

During the course of this thesis other useful features were implemented that were not directly related to ASAS. These were to improve the usability of the WPI Weka system and sometimes upon request of people using the system in their own research.

The ability to load frequent item sets from a cache file so rules may be generated from them directly was added. This allows for many different confidence values, required items, and other user specified criteria to be used to find rules without mining for frequent item sets each time.

The application of ASAS or any data mining technique can be lengthy. Giving the user information on the progress made so far by an algorithm might be very useful. Added to the system in this regard is the run-time reporting of system status. The status of mining and of rule generation are reported in both standard mining and during rule generation directly from a cache file of frequent item sets.

The metrics Lift and Chi-square, as described in [Alv03], were added to the system. They are now reported with each association rule found. However, these metrics are not available for association rules containing events.

In order to characterize and improve upon ASAS and research that will use ASAS a metric reporting feature was added to the system. A general logging mechanism was added for association rule mining. It logs various metrics to an ARFF file describing the behavior of the ASAS algorithm. These include the time in seconds for counting support at each level, the time to generate candidates at each level, etc. Also logged are statistics about the data set including the number of events found, average length of the time lines, etc. A complete list of these metrics can be found in Appendix B.

A Readme file containing complete instructions on the use of WPI's Weka add-on

package, of which ASAS and the associated filters and utilities are a part, was created. The Readme file also contains directions on making new filters and association rule mining algorithms available from the Weka graphical user interface. See Appendix A.

Chapter 8

Experimental Evaluation of ASAS

The associations rules presented here are in the form $A \Rightarrow B$, where A and B are each a set of items. Furthermore, events are annotated with their begin (t_1), and end (t_2) times as such: $[t_1, t_2]$. This is followed by the confidence, support, and event weight of the rule. The minimum and maximum possible length of each event type appearing in the rule is specified.

8.1 Evaluation on The Computer Performance Domain

8.1.1 Data Collection

The original motivation for this work was to assist a software engineering performance group with very complicated estimation and investigation tasks. The example performance paradigm is not as complicated.

The computer system performance domain was used because there exists many basic and easy to understand behaviors that are fairly easy to measure. Such relationships as “processor utilization increasing as memory utilization increases” are common, understood and easy to reproduce. By mining associations of this sort AprioriSetsAndSequences can be shown to produce valid, expected results.

Another advantage of working with the computer system performance domain is it is easy to collect such data. This was key. Publicly available data sets in a format where several sequences appear as attribute values for a single data instance were not found despite this being a natural representation for many domains. This required the compilation of original data sets. Computer performance was an obvious choice.

For this purpose several performance metrics were captured on a single computer system. The load on the computer system during performance data collection was the execution of a machine learning algorithm implemented in Weka. Collection of data would begin for a single data set instance before a machine learning algorithm would begin. After the machine learning algorithm had completed, the data collection was stopped. All the machine learning algorithms had the same basic process of building a model, training the model over the training data set, and testing the accuracy of the model over the test data set.

The machine learning algorithms that were used include Neural Networks with Back Propagation, Instance Based classification, Naive Bayes, and C4.5 Decision Trees. The training data set used was the census income data available at the UCI Machine Learning Repository's website [BM98]. The data used took several forms including normalized numeric values, discretized numeric values, and nominal attributes. All instances that included missing values were filtered out.

The experiments with machine learning algorithms were run on a Windows 2000 machine. Metrics were captured at the system level and at the process level. The same metrics were captured for each process running on the machine. A list of the Windows 2000 performance metrics captured during each experiment can be found in Appendix C. The descriptions shown were retrieved from Microsoft's MSDN web site [Cor] and are also available from the Performance Monitor Utility provided with Windows 2000.

Figure 8.1 shows a partial sample of the sequence data collected from a single experiment. This data along with all the other time sequences not shown comprise a single instance of the data set. Also see Figure 1.1.

8.1.2 Rules

Shown below is a rule which describes an obvious, general relationship between two attributes. this rule is illustrated in Figure 8.2.

$$\text{Idle Process Time Increases } [t_0, t_1] \Rightarrow \text{Total Process Time Decreases } [t_0, t_1]$$

$$[\text{Conf: 1.0, Sup: 0.94}]$$

The Idle process time is always inversely related to the Total process time. While the rule is not novel and simply points out a redundancy in the performance data that was collected, it is a valid rule. It is easy to verify. For this simple case ASAS found a valid temporal relationship. The algorithm did not know beforehand that the two attributes were related.

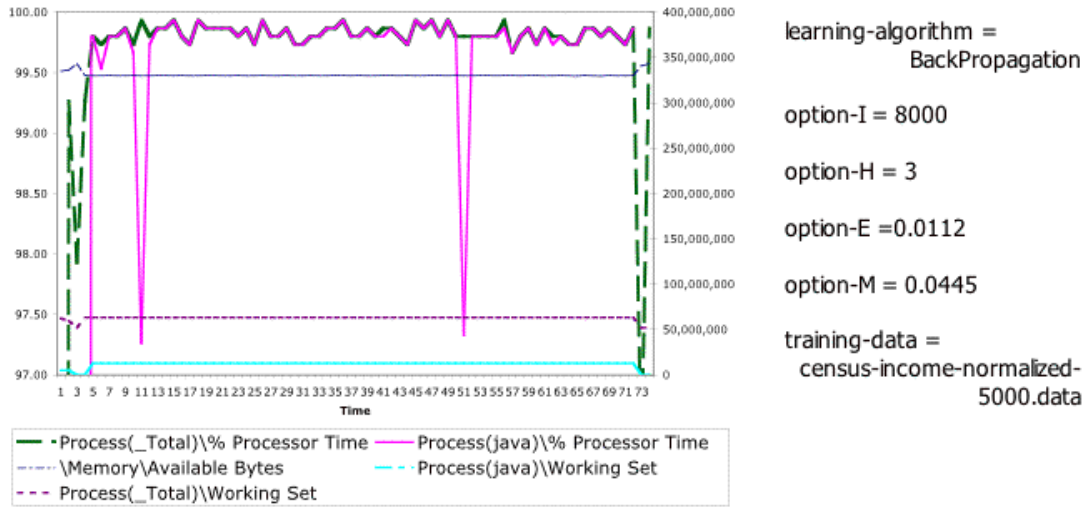


Figure 8.1: Partial Sample of A Single Performance Data Instance. See Appendix C and Appendix D for a complete listing.

The behavior described in the next rule, shown below and illustrated in Figure 8.3, is slightly more complicated and more subtle than the last example. It occurs when the main computation task, machine learning, is finished executing and memory resources are returned to the operating system.

Total Memory in Use Decreases $[t_0, t_1] \Rightarrow$ Total Process Time Decreases $[t_0, t_2]$
[Conf: 0.97, Sup: 0.85]

As soon as the machine learning task completes CPU utilization drops of course. It still takes some CPU cycles to handle the memory being released. While this rule cannot validate the correctness of ASAS and the system it is implemented in, to be able to obtain such rules was a significant milestone for this thesis.

The rules obtained from this domain were fairly uninteresting and not novel. This is due to the lack of interesting data collected. Figure 8.4 shows all the sequence attributes from a sample instance from this domain. The sequences are fairly unchanging. There seem to be no temporal patterns other than the type shown to be found in this data set.

This is due to the nature of the computation task selected. All the machine learning algorithms used during data collection followed the same three steps. First, a model is built. Second, the model is trained until it reaches some fitness criteria. Third, the model is evaluated over some test data. In the Figure 8.4 you can see where the model

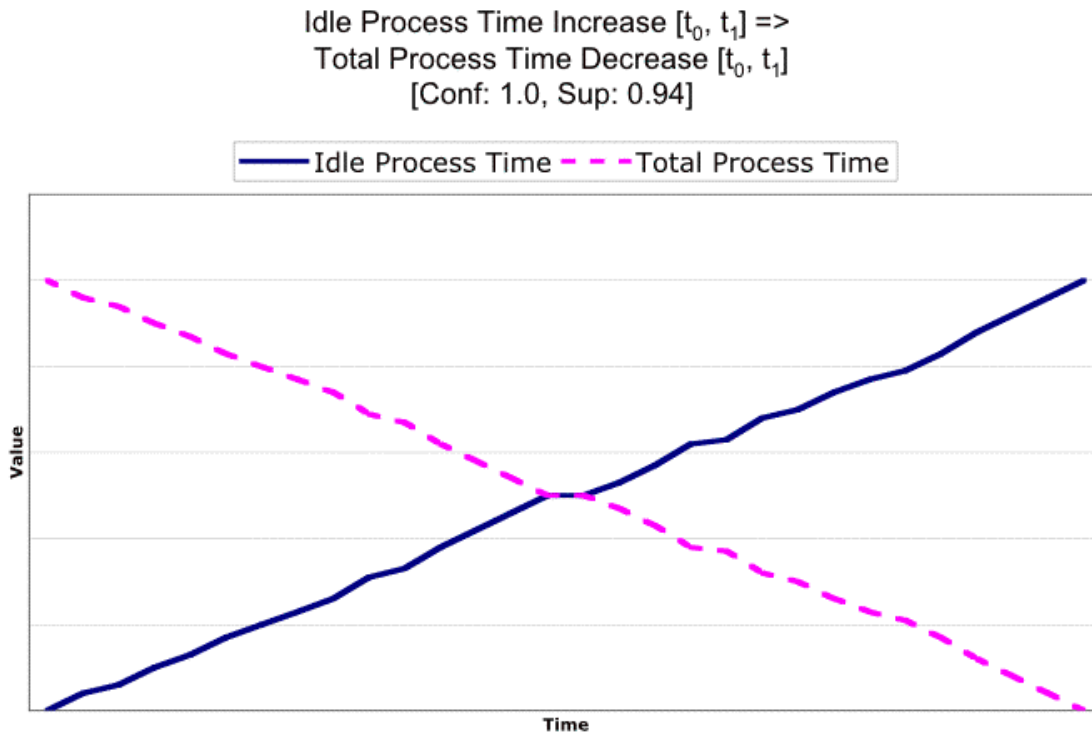


Figure 8.2: Idle Process Time Increases $[t_0, t_1] \Rightarrow$ Total Process Time Decreases $[t_0, t_1]$

building phase and the training phase end. The sequence for the Java process time shows a slight decrease at each end point. In this respect and seemingly all other respects, each machine learning algorithm used looks the same as far as resource use goes. Execution time did differ a great deal but this information was not usable in ASAS at the time these experiments were run.

8.1.3 Summary of Results

The evaluation of ASAS over computer system performance data showed ASAS found the temporal patterns that there were to find in the data set. It provided a link to the original motivation for this work. It served as an excellent source of readily collectible data to use for experimentation while developing the algorithm.

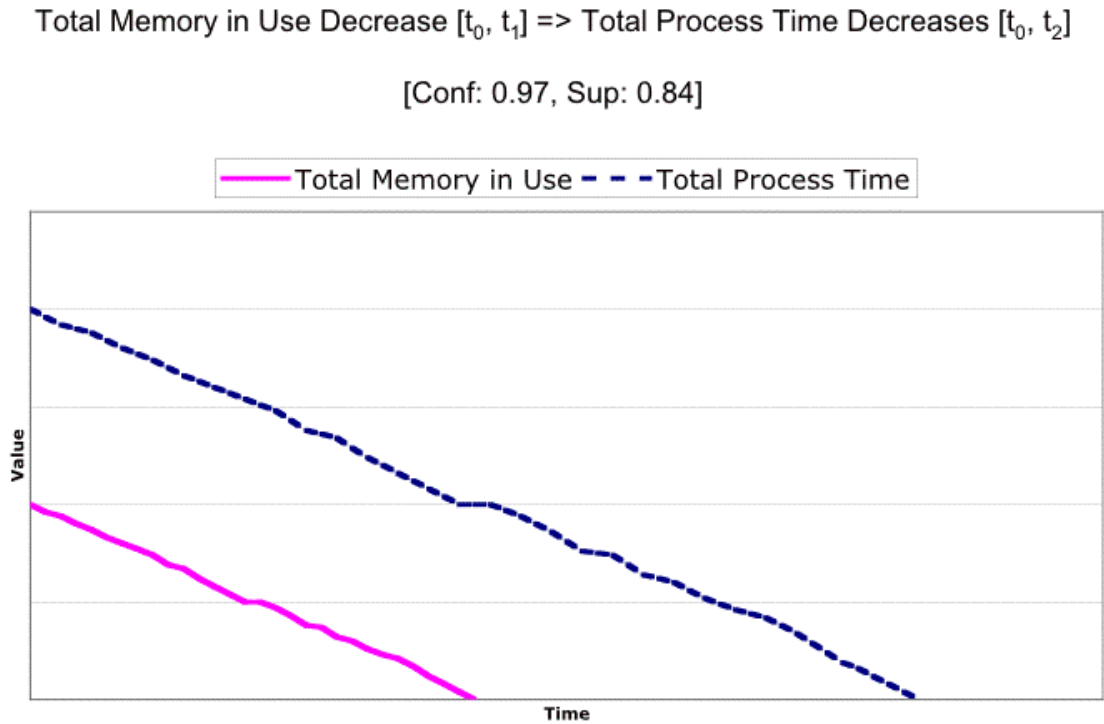


Figure 8.3: Total Memory in Use Decreases $[t_0, t_1] \Rightarrow$ Total Process Time Decreases $[t_0, t_2]$

8.2 Evaluation on The Stock Market Domain

8.2.1 Stock Market Data

A full description of each attribute in these data sets can be found in Appendix E.

The data used consists of ten years worth of closing prices from 7 technology companies from 1992 to 2002 obtained from Yahoo! Finance [Fin]. Additionally, events such as new product releases, awards received, negative press releases, and expansions or mergers from each company were obtained from each respective company's web site. Each instance in this data set represents a single quarter year. There are 24 instances in the data set. All 10 years are not represented because information on the additional events listed above were not available for all years.

Before mining, the sequences of closing prices for a quarter for each company are filtered for events. For each predefined event and closing price sequence, a new attribute is created indexing where this event type occurs in the sequence. The closing price sequence attributes are then removed from the data set. The financial events detected include

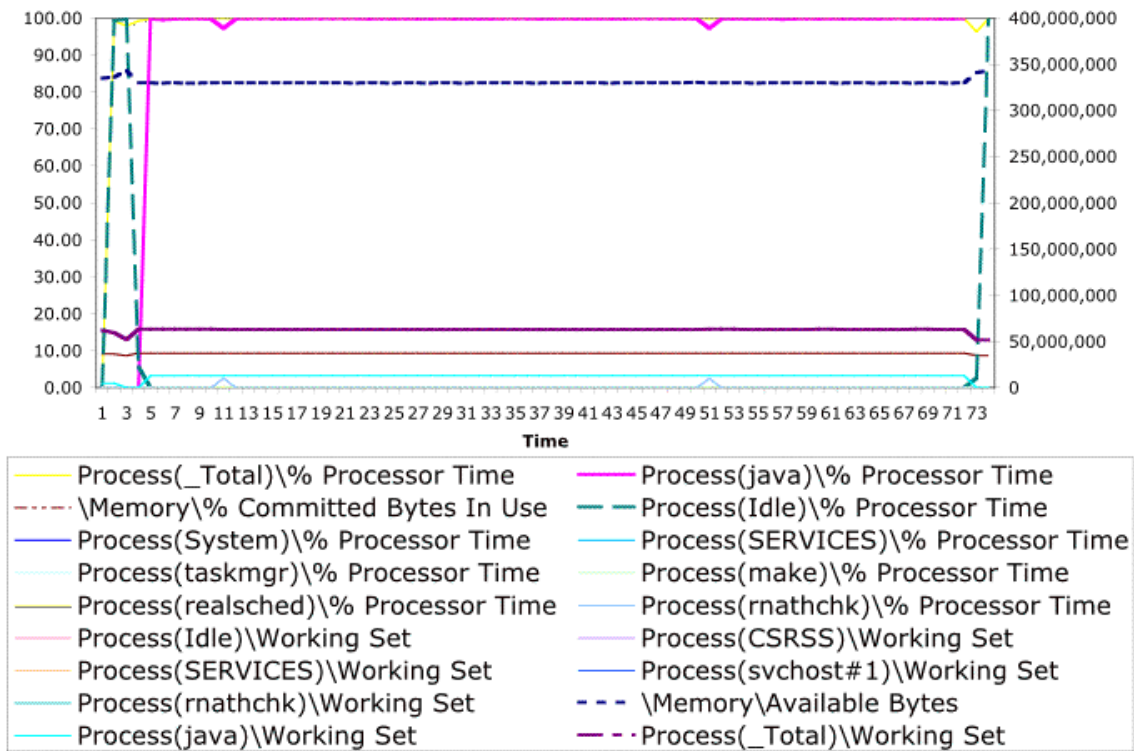


Figure 8.4: Performance Instance. See Appendix C and Appendix D for a complete listing.

rounded top, selling climax, ascending triangle, broadening top, descending triangle, double bottom, double top, head & shoulders, inverse head & shoulders, panic reversal, rounded bottom, triple bottom, triple top, sustain, increase, and decrease [LR91]. See Appendix E for descriptions and illustrations of these financial events. This data was compiled by [HL03].

8.2.2 Rules

This domain provided motivation to examine how ASAS could be used to place the relative temporal relationships it found in a more specific time period. We accomplished this by reporting the interval lengths of the events in the rules. When a user runs ASAS a summary of the minimum and maximum length of each event type is given.

The rules below were obtained by applying AprioriSetsAndSequences to the data set described in Section 8.2.1. Let us look at a fairly simple rule containing events.

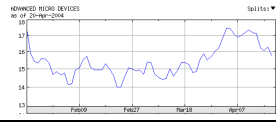
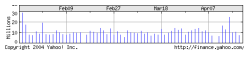
ID	AMD Product Announcement	AMD Stock	AMD Awards	AMD Trade Volume	CSCO Product Announcement
1	{Athlon XP 2200 [Nov 11, 02]}		{Lifetime Achievement [Oct 31, 02]}		{Aironet 1100 Series [Oct. 2, 2002]}
2	none	38, 47, 53, 55, ...	{Athlon XP Processor: Product of the Year [Jun 2]}	{5208400, 5959300, ...}	none
3	{1) AMD Duron 1.3GHz [Jan 21]}	18, 20, 16, 12, ...	none	{2720700, 4875000: ...}	none
...

Figure 8.5: A sample of a company and stock market dataset containing complex instances. Here, sequential values are represented in a graphical manner only for the first row to save space.

$$\text{CSCO Increase } [t_0, t_1] \Rightarrow \text{CSCO Sustain } [t_2, t_3]$$

[Conf: 0.8, Sup: 0.42, Event Weight: 16]

CSCO Increase 6-11 days, CSCO Sustain 6-11 days

This rule reads: The closing stock price of Cisco Systems Inc. increased in value for 6 to 11 days. With a confidence of 80% the closing price of Cisco will remain fairly constant for 6 to 11 days sometime after Cisco’s closing price stops increasing. It will do this during the same quarter year the increase took place. There is no overlap in time between the two events of this rule. With a support of about 42% this happens in about 10 of the quarters represented in the data set. In these 10 instances this behavior is found 16 times as noted by the event weight. Let us look at a rule similar to the one described above.

$$\text{NXTL Selling Climax } [t_0, t_1] \Rightarrow \text{NXTL Sustain } [t_2, t_3]$$

[Conf: 0.8, Sup: 0.67, Event Weight: 24]

NXTL Selling Climax 6-12 days, NXTL Sustain 6-12 days

Here the company of interest is Nextel Communications Inc. The relationship between the events are the same. The support is higher with this rule being found in 16 quarters. In the 16 instances representing those quarters this rule is found 24 times. This rule and the previous one are examples of predicting future values in a single sequence. This is just one form of the rules ASAS can find.

Diagnostic rules concerned with a single sequence can also be found. Rather than predicting an event to begin or end after the events in the antecedent, a diagnostic rule describes an association of events beginning or ending before those in the antecedent. This includes events in the consequent which occur between or during events in the antecedent.

INTC Increase $[t_2, t_3] \Rightarrow$ INTC Selling Climax $[t_0, t_1]$
 [Conf: 0.87, Sup: 0.46, Event Weight: 13]

INTC Increase 6-8 days, INTC Selling Climax 6-13 days

Intel Corporation's closing stock price increases for 6 to 8 days. Before Intel's stock price increases it goes through a selling climax that lasts 6 to 13 days during the same quarter. Let us look at a pair of rules. These have the same events in them but one has a predictive form and the other has a diagnostic form.

AMD Ascending Triangle $[t_0, t_1]$ & CSCO Expand Merge $[t_4, t_5]$
 \Rightarrow SUNW Sustain $[t_2, t_3]$
 [Conf: 0.91, Sup: 0.42, Event Weight: 10]

AMD Ascending Triangle $[t_0, t_1]$ & SUNW Sustain $[t_2, t_3]$
 \Rightarrow CSCO Expand Merge $[t_4, t_5]$
 [Conf: 1.0, Sup: 0.42, Event Weight: 11]

CSCO Expand Merge 1-7 days,
 AMD Ascending Triangle 6-30 days, SUNW Sustain 6-13 days

Advanced Micro Devices Inc's closing stock prices exhibit a pattern known as an ascending triangle for 6 to 30 days. Sometime after but during the same quarter Sun Microsystems Inc's closing stock price remains fairly constant for 6 to 13 days. Sometime after in the same quarter Cisco goes through a period of expansion or merger for 1 to 7 days. The predictive form of the rule has a 100% confidence. In any quarter in the data set, every time AMD and Sun exhibit the behaviors described in these rules, Cisco expands or merges. This rule is illustrated in Figure 8.6.

This example shows rules where events identified in different numeric sequences and symbolic events are related in time. The temporal relationships affect the confidence of the rules, even when they have the same events. This example also shows that ASAS can find diagnostic forms of association rules.

AMD Ascending Triangle $[t_0, t_1]$ & SUNW Sustain $[t_2, t_3] \Rightarrow$ CSCO Expand Merge $[t_4, t_5]$

[Conf: 1.0, Sup: 0.42, Event Weight: 11]

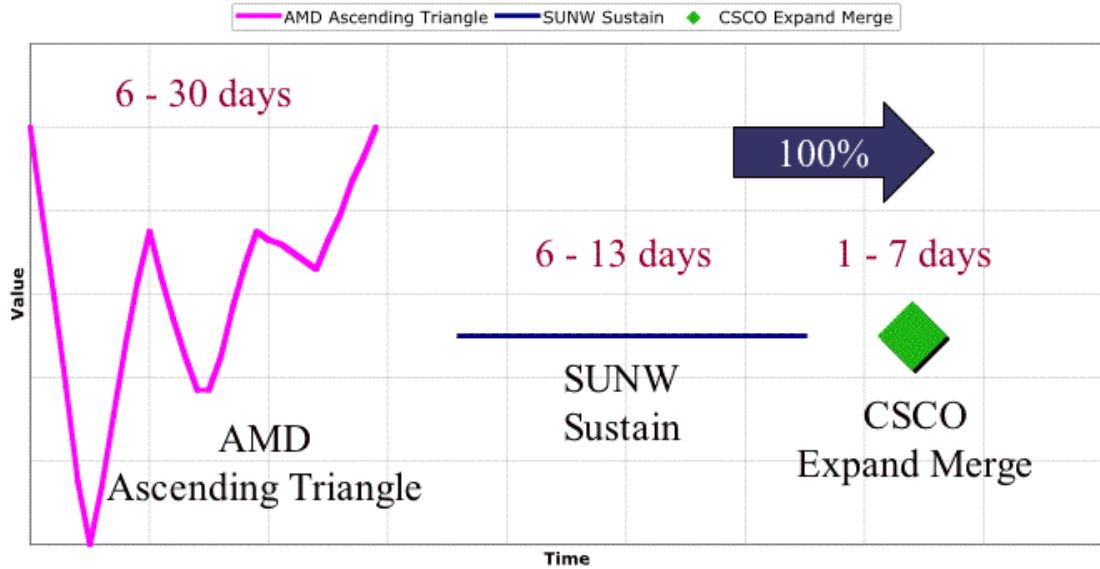


Figure 8.6: AMD Ascending Triangle $[t_0, t_1]$ and SUNW Sustain $[t_2, t_3] \Rightarrow$ CSCO Expand Merge $[t_4, t_5]$

AMD Sustain $[t_0, t_2]$ & INTC Sustain $[t_1, t_3] \Rightarrow$ MSFT Sustain $[t_4, t_5]$

[Conf: 0.78, Sup: 0.42, Event Weight: 7]

AMD Sustain 6-9 days, INTC Sustain 6-10 days,
MSFT Sustain 6-13 days

AMD's closing stock price remains fairly constant for 6 to 9 days. During this 6 to 9 days interval Intel Corporation's closing stock price begins to remain fairly constant. Intel's sustain period last for 6 to 10 days. During this 6 to 10 day period AMD's sustain ends. Sometime after Intel's sustain period ends but in the same quarter year, Microsoft Corporation's closing stock price remains fairly constant for 6 to 13 days.

Events overlapping in time make the temporal relationships between events more precise and may make it more interesting. These rules can be found by ASAS in predictive and diagnostic forms.

An interesting characteristic of some of the rules found in our dataset with overlapping events was that they described the events themselves in terms of other events. A company's closing stock price could exhibit a complicated behavior like a selling climax.

A portion of this complicated behavior may be similar to a much simpler behavior such as a sustain in stock closing price. ASAS can find rules that state a sustain event and selling climax event can overlap in the same sequence as illustrated in the rule below.

AMD Selling Climax $[t_0, t_2] \Rightarrow$ AMD Sustain $[t_1, t_3]$

[Conf: 0.61, Sup: 0.42, Event Weight: 14]

AMD Selling Climax 6-15 days, AMD Sustain 6-9 days

Parts of complicated events can also be found similar.

SUNW Inverse Head & Shoulders $[t_1, t_3] \Rightarrow$ SUNW Descending Triangle $[t_0, t_2]$

[Conf: 0.71, Sup: 0.42, Event Weight: 12]

SUNW Inverse Head & Shoulders 6-14 days, SUNW Descending Triangle 6-24 days

Multiple Occurrences of Event Type

The AprioriSetsAndSequences algorithm allows rules to be found containing multiple occurrences of events with the same type.

CSCO Sustain $[t_0, t_1] \Rightarrow$ CSCO Sustain $[t_2, t_3]$

[Conf: 0.81, Sup: 0.54, Event Weight: 38]

CSCO Sustain 6-11 days

Cisco's closing stock price remains fairly constant for 6 to 11 days. With a confidence of 81% we can say that later in the same quarter Cisco's value will experience another 6 to 11 days period of sustain. It is interesting to compare rules which predict the same repeating patterns for different companies. Below are rules that exhibit repeating sustain events for four other companies.

SUNW Sustain $[t_0, t_1] \Rightarrow$ SUNW Sustain $[t_2, t_3]$

[Conf: 0.87, Sup: 0.58, Event Weight: 39]

MSFT Sustain $[t_0, t_1] \Rightarrow$ MSFT Sustain $[t_2, t_3]$

[Conf: 0.86, Sup: 0.63, Event Weight: 36]

NXTL Sustain $[t_0, t_1] \Rightarrow$ NXTL Sustain $[t_2, t_3]$

[Conf: 0.83, Sup: 0.63, Event Weight: 40]

AMD Sustain $[t_0, t_1] \Rightarrow$ AMD Sustain $[t_2, t_3]$

[Conf: 0.81, Sup: 0.54, Event Weight: 35]

Let us compare two rules. In the first an event repeats twice. In the second the same event repeats three times. These two rules are illustrated in Figure 8.7.

INTC Sustain $[t_0, t_1] \Rightarrow$ INTC Sustain $[t_2, t_3]$

[Conf: 0.92, Sup: 0.63, Event Weight: 44]

INTC Sustain $[t_0, t_1] \Rightarrow$ INTC Sustain $[t_2, t_3]$ & INTC Sustain $[t_4, t_5]$

[Conf: 0.71, Sup: 0.42, Event Weight: 34]

INTC Sustain 6-10 days

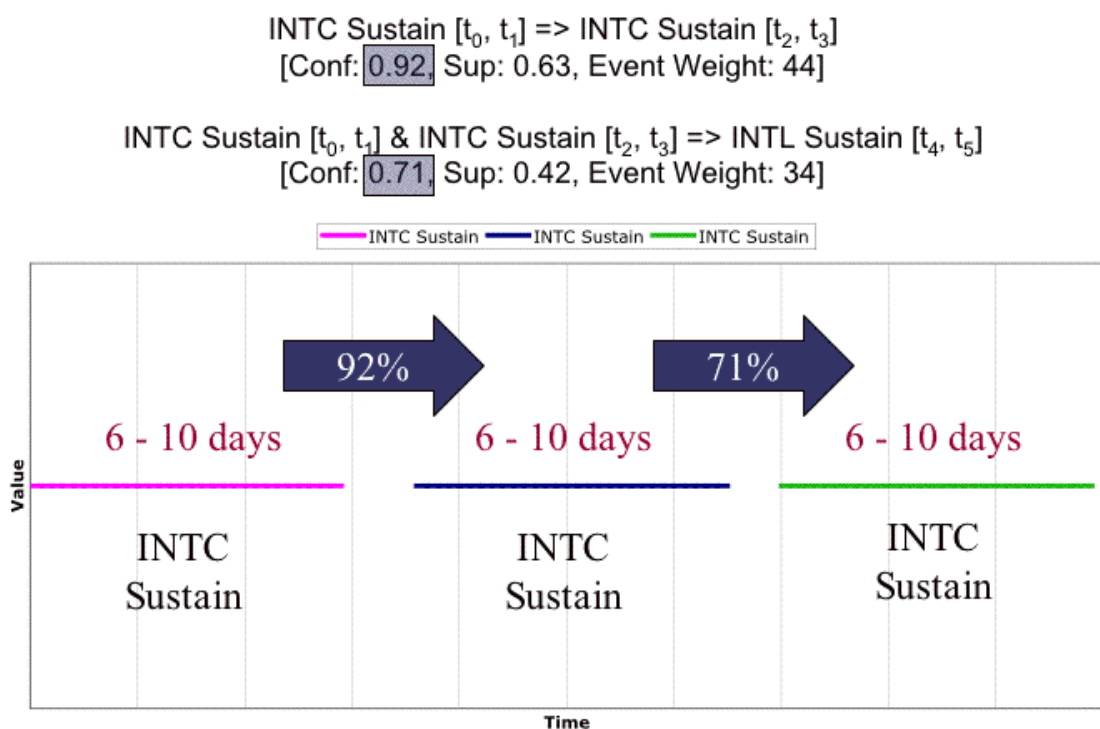


Figure 8.7: INTC Sustain $[t_0, t_1] \Rightarrow$ INTC Sustain $[t_2, t_3]$

Notice the reduced confidence in predicting Intel's stock price will undergo a sustain period twice more in the same quarter. Repeating events also occur in events that were not obtained from a sequence. The rule below shows a repeating Cisco Expand Merge event.

CSCO Expand Merge $[t_0, t_1] \Rightarrow$ CSCO Expand Merge $[t_2, t_3]$

[Conf: 0.9, Sup: 0.46, Event Weight: 36]

CSCO Expand Merge 1-7 days

Rules with repeating events are not limited to just one type of event. Below is an example of a rule containing both a singular event and a repeating event.

CSCO Expand Merge $[t_4, t_5]$ & NXTL Sustain $[t_0, t_1] \Rightarrow$ NXTL Sustain $[t_2, t_3]$
 [Conf: 0.92, Sup: 0.42, Event Weight: 24]

NXTL Sustain $[t_2, t_3]$ & NXTL Sustain $[t_0, t_1] \Rightarrow$ CSCO Expand Merge $[t_4, t_5]$
 [Conf: 0.67, Sup: 0.42, Event Weight: 16]

CSCO Expand Merge 1-7 days, NXTL Sustain 6-12 days

As shown above rules with repeating events can be found in predictive and diagnostic forms.

8.2.3 Summary of Results

Overall, the rules produced from this domain far exceeded our expectations in confidence. Obtaining temporal relationships with confidences in the 70, 80, and even 90 percent ranges from stock market data was not expected. Future work in this domain with a financial expert would be interesting.

Associations were found that contained many different types of complex temporal relationships. Association rules where the events in the antecedent occurs before the events in the consequent could be used for prediction. Rules where the events in the consequent occur before those in the antecedent could be used in a diagnostic capacity, to explain observed behavior. Rules were found where the same event type repeats. This domain also provided data suitable for examining some algorithmic behavior of ASAS. Those results are presented in Section 8.4.

8.3 Evaluation on The Human Sleep Domain

8.3.1 Clinical Sleep Data

Figure 8.8 shows a small sample of the data used for this set of experiments. The data was obtained from Laxminarayan [Lax04]. The sequential data was taken every 30 seconds and each 30 second interval is referred to as an epoch.

A full description of each attribute in these data sets can be found in Appendix F.

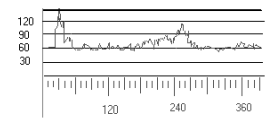
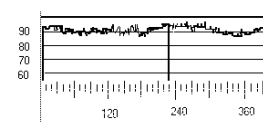
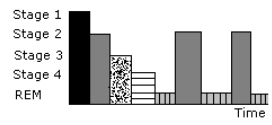
ID	OSA	heart rate	age	oxygen potential %	G	drugs	sleep stage
1	severe		27		M	{pro, flo}	
2	moderate	97, 72, 67, 80, ...	73	90, 92, 96, 89, 86, ...	F	{flo, asp, zol}	S2,S2,REM,S1, ...
3	none	102, 99, 87, 96, ...	49	97, 100, 82, 80, 70 ...	M	{zol}	S1,S1,S3, ...
...

Figure 8.8: A sample of a dataset containing complex instances. Here, sequential values are represented in a graphical manner only for the first row to save space. **G** stands for Gender with values M (male) and F (female). S_i denotes sleep stage i . Medication names have been abbreviated.

8.3.2 Rules

Laxminarayan’s work required that the rules obtained from mining be able to express temporal relationships which were associated with specific times during a patient’s night sleep. Time Granularities as discussed in Section 7.1 was not yet available at the time of Laxminarayan’s project. Laxminarayan implemented a window based preprocessing step to create items which contained temporal information. This temporal information identified where during a patient’s night sleep the item, or windowed event occurred. (See [Lax04].)

Here we apply ASAS with Time Granularities to human sleep data in hopes of obtaining rules similar to those presented in [Lax04]. Although more rules were obtained only a handful of the most significant rules were selected for inclusion in this document.

The rules below were obtained using a data set containing only the sleep stages 1, 2, 3, 4, and Rapid Eye Movement (REM), and level of Obstructive Sleep Apnea (OSA). A time granularity of 50 epochs, or 25 minutes is also part of the data set. Only rules that classify for Obstructive Sleep Apnea (OSA) are shown. Note: Chi-square and Lift values are available for the rules presented in this section. This is because only the antecedents contain event items. It is only when event items appear in both the antecedent and consequent of a rule that these metrics become unavailable.

The rule shown below and illustrated in Figure 8.9 can be read as a patient who exhibits sleep stage 1 which begins sometime before epoch 700 (minute 350) and ends sometime between epoch 700 (minute 350) and epoch 749 (minute 374), with a likelihood of 83%, suffers from mild obstructive sleep apnea. The statistical robustness of this rule is quite high as shown by the chi-square value.

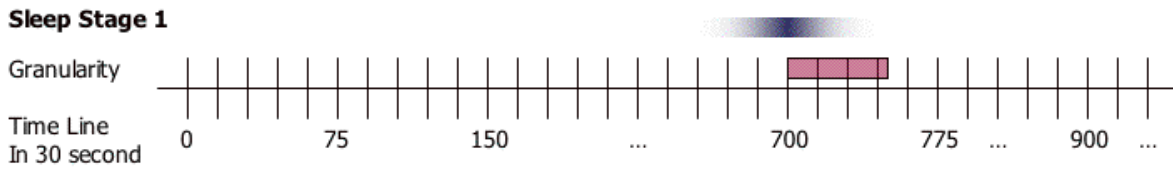


Figure 8.9: Sleep Stage 1 $[t_0, t_2]$ and Granule 700-749 $[t_1, t_3] \Rightarrow$ Mild OSA

Sleep Stage 1 $[t_0, t_2]$ & Granule 700-749 $[t_1, t_3] \Rightarrow$ Mild OSA
 [Conf: 0.83, Sup: 0.063, Lift: 3.51, Chi-S: 12.72, $p < 0.001$]

In the illustration (Figure 8.9) a patient's night sleep is shown as a time line marked by epoch. The sleep stage event is marked as a bar which is faded at either end. The time granularity is also shown as a bar but which appears solid from end to end.

The next rule shown below and illustrated in Figure 8.10 can be read as a patient who exhibits sleep stage 2 which begins between epoch 150 (minute 75) and epoch 199 (minute 99) and ends sometime after epoch 200 (minute 100), with a likelihood of 67%, suffers from moderately severe obstructive sleep apnea. This rule also seems statistically robust.

Sleep Stage 2 $[t_1, t_3]$ & Granule 150-199 $[t_0, t_2] \Rightarrow$ Moderately Severe OSA
 [Conf: 0.67, Sup: 0.075, Lift: 2.96, Chi-S: 11.34, $p < 0.001$]

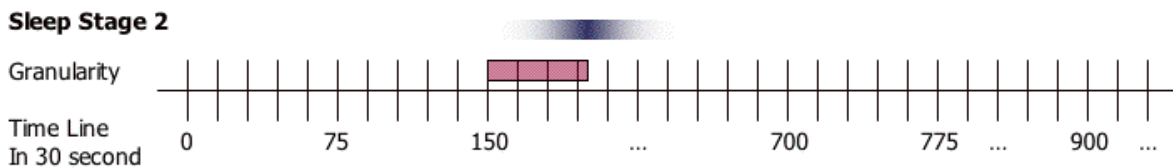


Figure 8.10: Sleep Stage 2 $[t_1, t_3]$ and Granule 150-199 $[t_0, t_2] \Rightarrow$ Moderately Severe OSA

The next rule shown below and illustrated in Figure 8.11 can be read as a patient who exhibits sleep stage 3 which begins between epoch 300 (minute 150) and epoch 349 (minute 174) and ends after epoch 350 (minute 175), do not suffer from obstructive sleep apnea.

Sleep Stage 3 $[t_1, t_3]$ & Granule 300-349 $[t_0, t_2] \Rightarrow$ No OSA
 [Conf: 0.7, Sup: 0.088, Lift: 1.87, Chi-S: 5.15, $p < 0.023$]

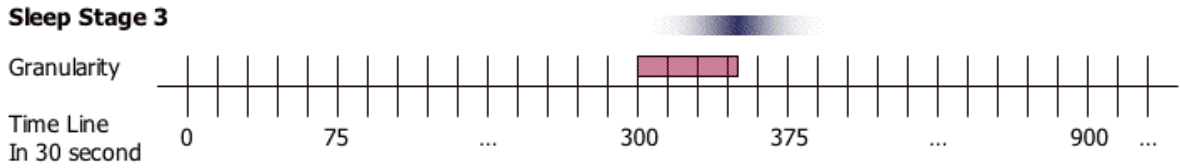


Figure 8.11: Sleep Stage 3 $[t_1, t_3]$ and Granule 300-349 $[t_0, t_2] \Rightarrow$ No OSA

The last rule shown below and illustrated in Figure 8.12 can be read as a patient who exhibits Rapid Eye Movement (REM) occurring between epoch 750 (minute 375) and epoch 799 (minute 399), with a likelihood of 67%, do not suffer from obstructive sleep apnea.

Sleep Stage REM $[t_1, t_2]$ & Granule 750-799 $[t_0, t_3] \Rightarrow$ No OSA
 [Conf: 0.67, Sup: 0.1, Lift: 1.78, Chi-S: 5.12, $p < 0.023$]



Figure 8.12: Sleep Stage REM $[t_1, t_2]$ and Granule 750-799 $[t_0, t_3] \Rightarrow$ No OSA

8.3.3 Summary of Results

This data set and the need expressed in [Lax04] for a system that finds association rules that contain explicit times motivated the use of time granularity events with ASAS. In trying to obtain rules similar to those presented by Laxminarayan we show that using Time Granularities with ASAS is a viable way to mine for association rules expressing general temporal patterns as well as time specific patterns over multiple granularities at the same time. The rules found express statistically robust associations which seem relevant to the study of human sleep.

8.4 Performance Evaluation of ASAS

The machine used to run experiments for the results shown in Section 8.4.1 Time to Mine had 512 MB of physical RAM and a 600MHz Pentium II CPU.

The machine used to run experiments for the results shown in Section 8.4.2 Effect of Increasing Maximum Number of Same Type Events Allowed in a Rule had 512 MB of physical RAM and a 1.5MHz Pentium IV CPU.

For both sets of experiments the Java virtual machine which ran ASAS was configured with a limit of 900 MB.

8.4.1 Time to Mine

There are many factors which, at first, might seem to effect the total time the ASAS mining process will take to finish. Another way of thinking about the performance of ASAS is the time per frequent item set found. That is, how long, on average, did ASAS take to find each frequent item set. In this section we look at both total time and time per frequent item set. We will look at total mining time first.

Total Time to Mine

Figure 8.13 shows the event characteristics of various data sets from the Stock Market Domain described in Section 8.2.1. These characteristics are plotted according to the total time spent mining that data set. The x-axis shows the total time in seconds spent mining along a logarithmic scale. The left hand y-axis has a logarithmic scale and shows the number of event occurrences in a data set. The right hand y-axis shows the number of event attributes in a data set and the average number of event occurrences per event attribute. The results for each of the five experiments run all appear in a single column in the Figure.

ASAS was run with the minimum confidence set to 60%, the minimum number of rules to produce was set to 1, and all other user parameters used the default values. Please see Appendix A for user parameter details. The only difference between the five experiments are data sets used. In increasing total time to mine order, the data sets used for are *csc-msft-events*, *news*, *1-90-evented*, *7-90-evented*, and *simple-events*.

The *csc-msft-events* data set consists of two companies, Cisco and Microsoft, and the stock market value events described in Appendix E. The *news* data set consists of all seven companies appearing in Section 8.2 and the product release, award, bad news, and expand merge events. The *1-90-evented* data set consists of the single company Cisco and the stock market value events. The *7-90-evented* data set consists of all seven companies and their stock market value events. The *simple-events* data set consists of all seven companies and the increase, decrease and sustain events detected over their stock market

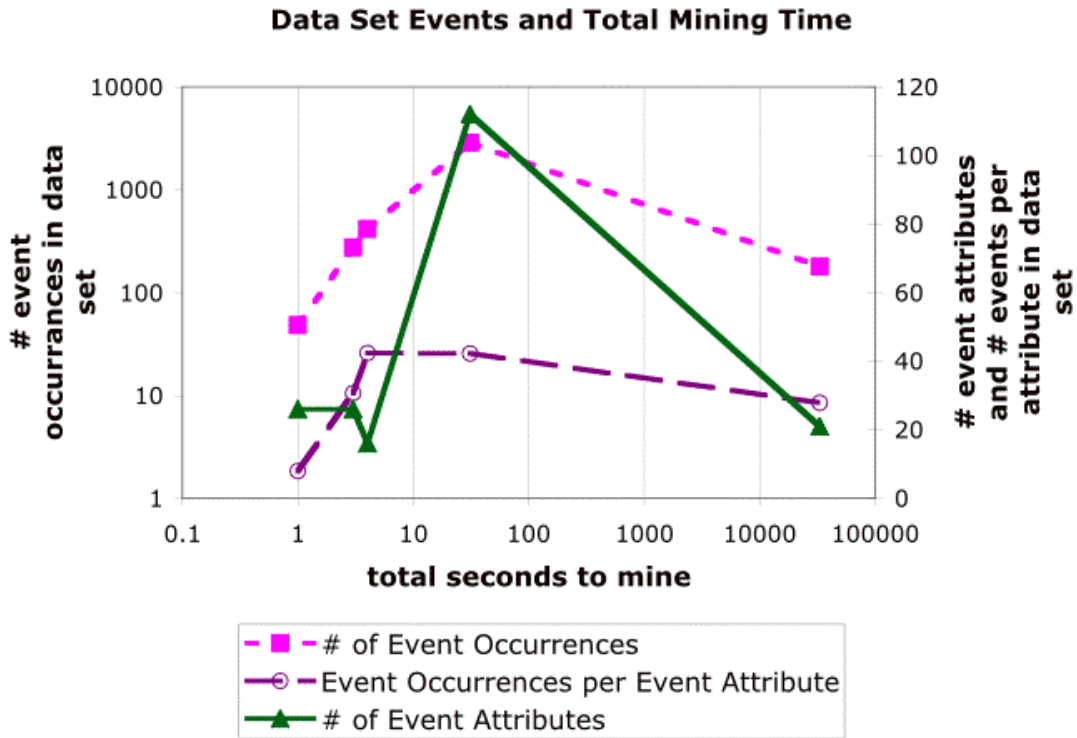


Figure 8.13: Data Set Events and Total Mining Time

value. All data sets consist of 30 data instances. They only contain event attributes. The number of event attributes in each data set is given in Figure 8.13.

The number of event attributes is the total number of attributes in a particular data set that are of the event set type. The number of event attributes dictates the number of event items created in the first level of candidate generation. The more event attributes, the more potential work for ASAS to mine frequent item sets.

The total number of event occurrences is the total count of events contained in each and every instance in a data set. This can be thought of as the number of items which ASAS must compare candidate item sets to. The more there are, the more potential work for ASAS to mine frequent item sets.

The average number of event occurrences per event attribute is a view of how dense the temporal landscape of a data set is. One might expect many frequent item sets to be found the more events that occurred in a single attribute.

Figure 8.14 shows the number of frequent item sets found during mining. This seems to be the only measured parameter which has a strong correlation with the total time ASAS spends mining. The more frequent patterns there are to find in a data set, the

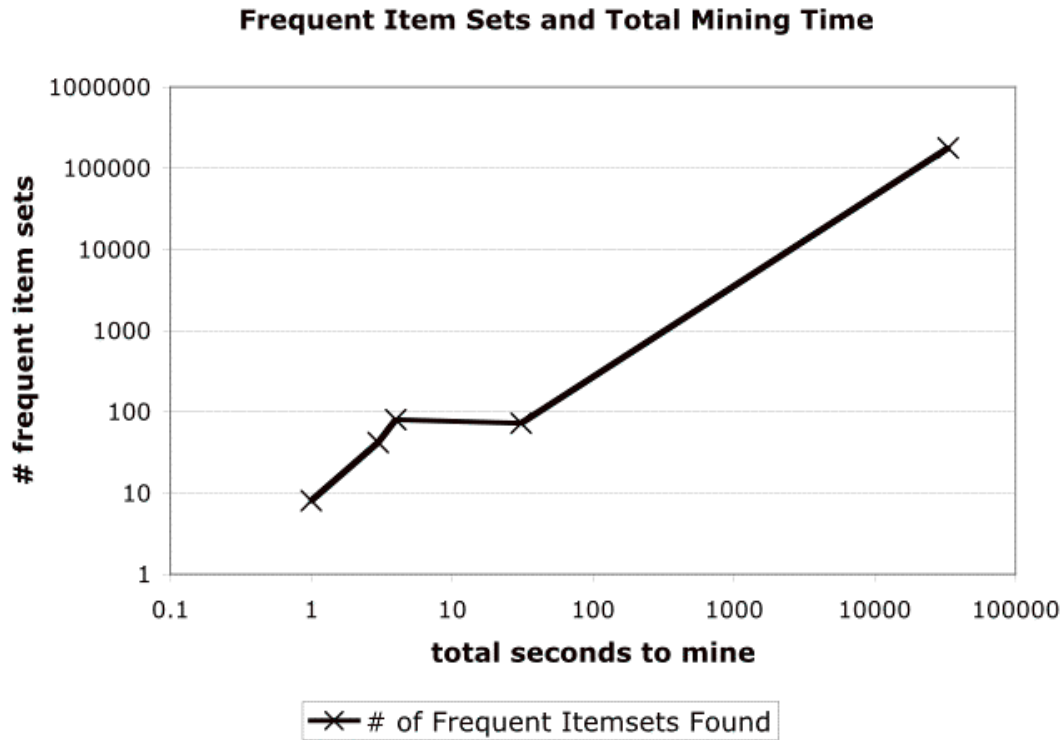


Figure 8.14: Frequent Item Sets and Total Mining Time

more time ASAS spends to find them. No other characteristics of the data set seem to influence the mining time more.

A comparison between Figures 8.13 and 8.14 shed some light on the increase in mining time between the third and fourth experiments. The number of frequent item sets found does not increase. Both the number of event attributes and event occurrences in the data set do increase. While the number of frequent item sets to be found in the data set seems to be the main determining factor of mining time, the amount of information contained in a data set still influences the mining time to some degree.

Time to Mine per Frequent Item Set

Another way to consider the time spent mining is the time it takes to mine each frequent item set. The lower the time per item set, the more efficient the mining process seems. Figure 8.15 shows the average time line length for each data set and how it relates to the time spent mining per frequent item set. The interval of time represented by each instance in a data set does not seem to have an effect on the efficiency of ASAS to mine frequent item sets.

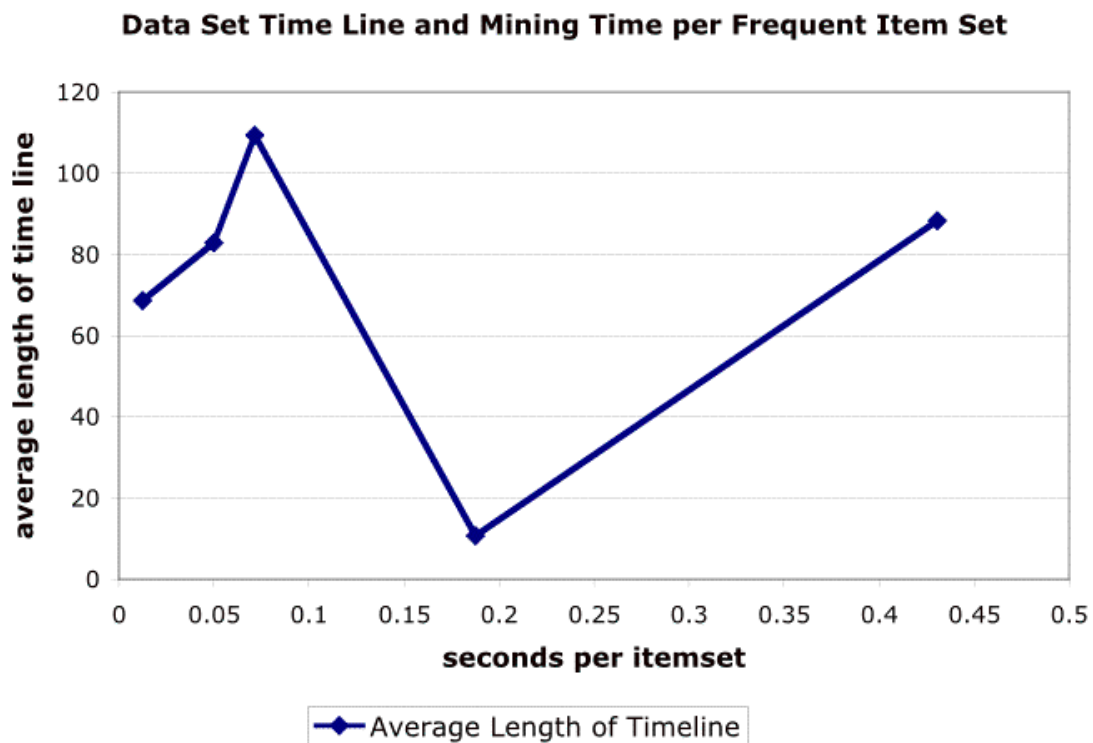


Figure 8.15: Data Set Time Line and Mining Time per Frequent Item Set

Figure 8.16 shows the same data set characteristics as Figure 8.13 but now arranged by mining time per frequent item set.

8.4.2 Effect of Increasing Maximum Number of Same Type Events Allowed in a Rule

The user parameter maximum number of events of the same type allowed in a rule can have a large influence on the overall run time of ASAS and the efficiency with which ASAS finds frequent item sets. To examine this effect, a mining task using a data set containing all the information for all seven companies from the stock market domain described in Section 8.2.1. Four experiments were run with values of 1, 2, 3, and 4 for the maximum same type event items allowed in a rule. The only difference between each mining task is the maximum number of same type events allowed in a rule. The minimum support value used is 49%.

Figure 8.17 shows the number of association rules found, the number of frequent item sets found, and the time per frequent item set found. Along the x-axis is the number of

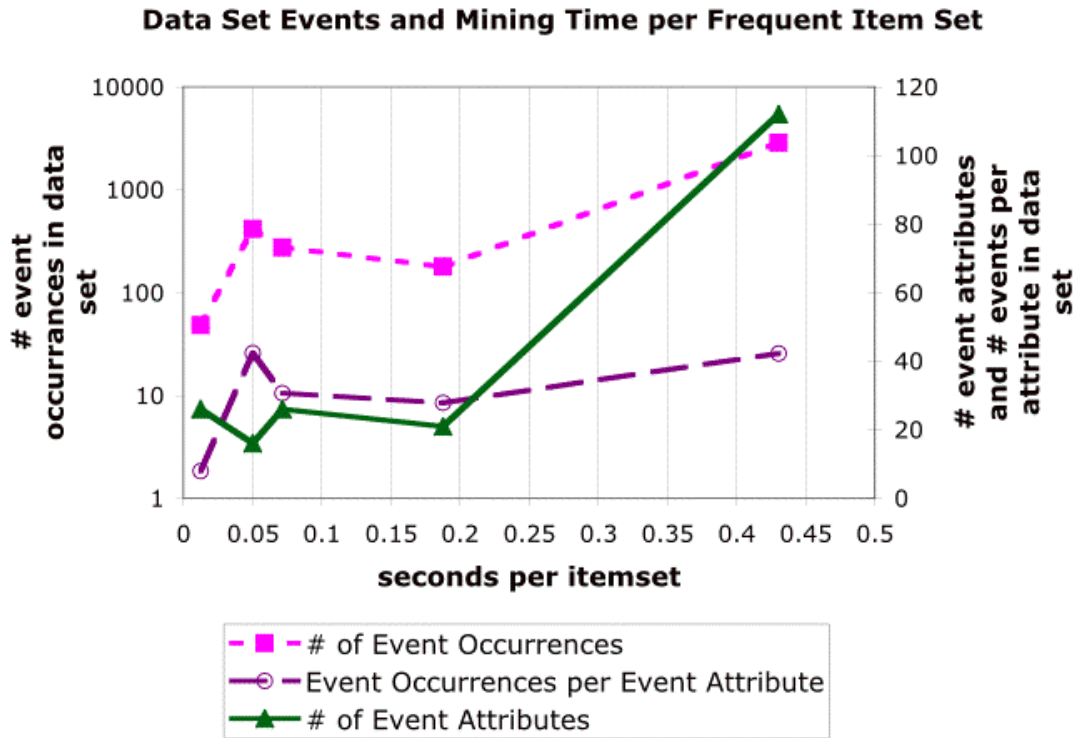


Figure 8.16: Data Set Events and Mining Time per Frequent Item Set

event of the same type allowed in a rule. Along the left hand y-axis is the seconds spent per frequent item set and the number of rules found on a logarithmic scale. The right hand y-axis shows the number of frequent item sets found.

The results from the first experiment in Figure 8.17 is for mining rules with a single event item of each type allowed. These results appear in the first column of values plotted in the Figure. There were 35 such rules found. The second experiment shows results for when the number of event items of the same type allowed in a rule is two. This yielded 51 rules. This is 16 more rules than found in the last experiment. To find this extra 16 rules more candidate item sets were generated. The frequency of those candidates were then counted in the data set. More frequent item sets were found as shown in the Figure. The time per item set spent on this process stays the same.

For the third and fourth experiments we see that there are no more additional rules to be found containing three or four event items of the same type in this data set. ASAS still must do the work creating candidate item sets and counting support. This work is done needlessly since no more additional frequent patterns, as represented by association rules, are present.

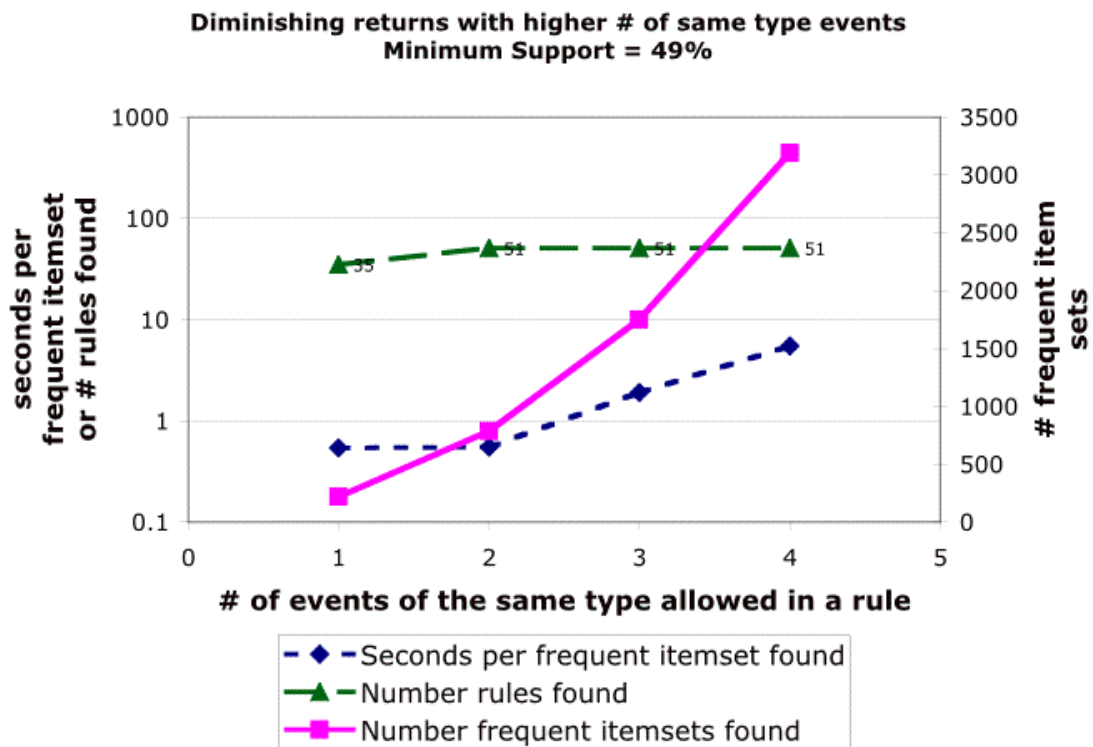


Figure 8.17: Diminishing Returns With Higher Number of Same Type Events Allowed In a Rule

Figure 8.18 shows the first three experiments from above but with a lowered support of 40%. By lowering the support we make it possible to find more frequent patterns in the data set. This allows another view of the behavior illustrated in Figure 8.17.

We can see that although by lowering support more rules are now found for each maximum number of event items of the same type allowed in a rule, the work involved in finding those rules grows at a faster rate. This seems to be caused by the percentage of new patterns found being lower than the previous set of experiments. In the previous experiment, between the first and second experiment there was a 46% increase in the number of rules found. In this experiment the increase was 33%. The less patterns to be found and the more patterns we look for, the more wasted effort.

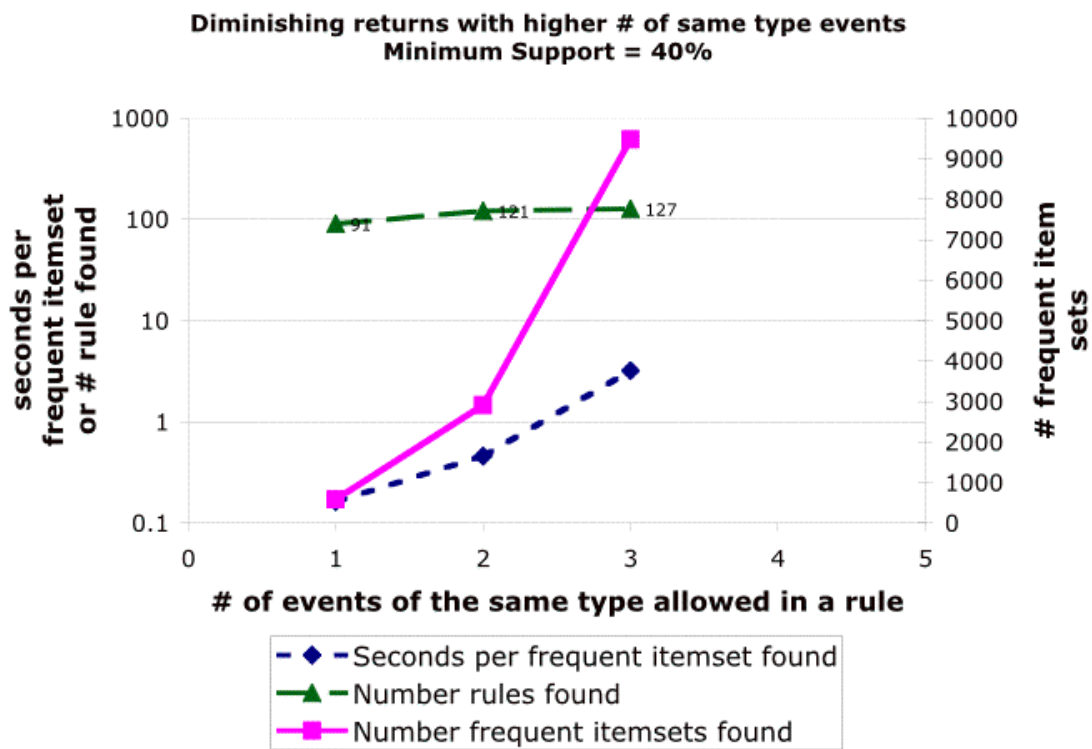


Figure 8.18: Diminishing Returns With Higher Number of Same Type Events Allowed In a Rule and Lower Support

Chapter 9

Conclusions and Future Work

9.1 Conclusions

In this thesis we introduced an algorithm for mining expressive temporal relationships from complex data sets in which a single data instance may consist of a combination of attribute values that are nominal sequences, time series, sets, and traditional relational values. Our mining algorithm is close in spirit to the two-stage Apriori algorithm. Our work contributes the the investigation of prune strategies and efficient data structures to effectively handle the added data complexity and the added expressiveness of the temporal patterns.

Several alternative methods to Apriori's item set generation have been proposed in the literature. Those alternative methods differ from Apriori in their strategy to generate *frequent* item sets, or in the item sets that they consider *interesting*. Some approaches attempt to increase the performance of the mining algorithm over certain types of data [ZPOL97]; compute frequent item sets very efficiently [HPY00, XD99, Lin98, Hid98]; or utilize parallel computing environments [AS96, ZOPL96]. Work by Webb [Web00], Agarwal, Aggarwal, and Prasad [AAP00, AAP01], and Bayardo, Agrawal, and Gunopoulos [BAG00] address some of the efficiency concerns by means of novel and judicious search techniques for item set generation. Modifications and generalizations of association rules that are more suitable to certain application domains are investigated by Brin, Motwani, and Silverstein [BMS98] and by Cohen et al. [CDF⁺00]. Additional work on using statistical measures of item set interestingness other than support and lift are investigated by DuMouchel and Pregibon [DP01] and by Wu, Barbará and Ye [WBY03].

This work focuses on the necessary extensions of the traditional support and confidence framework to handle the desired complex associations. Zhen, Kohavi, and Mason

[ZKM01] show experimentally that although some well-known alternative association rule mining approaches discussed above outperform Apriori over artificial data sets, they do not over real-world data sets. The work described here provides a foundation for future investigation and comparison of alternative measures of item set interestingness and alternative search techniques such as those discussed above but in the context of complex data.

9.2 Future Work

9.2.1 ASAS Extensions

ASAS could be extended to handle one more temporal concept, the instantaneous event. It seems to be a simple implementation problem to include this concept. An instantaneous event would begin and end at the same time. There would be no need to change ASAS's basic representation of events. Local changes to item set matching and candidate generation would have to be made.

The introduction of Time Granularities to ASAS creates many event items which are usually frequent. This makes the time to mine much longer. See Section 8.4 to see how the number of frequent item sets found effects the time to mine. Many rules are produced that one contain these frequent granularity event items. ASAS should be modified to treat granularity attributes as a special case. For example, there is no need to generate rules from an item set containing only granularity event items. In fact those item sets need not be present in the frequent item set list at all. The work involved is taking care of all the places in the implementation where those item sets would be expected in the list.

9.2.2 Improve Candidate Generation

Candidate generation is the part that currently takes the longest during the ASAS mining process, for the majority of the data sets it has been used with. There are two main approaches for improving candidate generation described here. The first involves reducing memory consumption. The second involves more intelligent handling of same type event items.

Improve Candidate Generation: Memory Consumption

To date memory usage has been recorded in the multiple gigabyte range for data sets containing many frequent item sets. To make ASAS viable for large database applications this memory problem must be addressed.

Candidate generation gets very slow when memory is utilized beyond the physical limit of the machine. To reduce this effect we could try counting support intermittently during generation to remove candidates found not frequent earlier, freeing memory resources for the rest of the candidate generation phase. This might have to be balanced with the data set size. If the data set is substantially large enough that a complete scan of it makes the support counting process very long, counting support during candidate generation might not be an option. This option could still be implemented with a switch sensitive to data set size or as a user option.

Another memory reducing strategy might be to take advantage of the fact that only the item sets suspected of being the first in a succession of conceptually duplicate item sets (See Section 6.2.7) have to be kept for quick access. The full candidate list could be cached to a file during candidate generation and until the support counting is done.

Memory might also be saved by reviewing the Item Set class implementation. This class represents the most numerous object type during execution. Even the smallest memory savings in this class will be potentially magnified hundreds of thousands of times.

Improve Candidate Generation: Intelligent Same Type Event Handling

The second approach to improve the speed at which candidates are generated involves treating the event items of the same type more intelligently. In Section 6.2.6 the reason why these multiple events are used is said to be so the standard Apriori method of generating and counting support will work. If the generation part is changed so that event of the same type are used only if need be, much time could be saved.

Using the same example from Section 6.2.7 we get the result shown in Figure 9.1 for the level 2 candidates.

Figure 9.2 shows the candidate item sets generated for level 3. The total number of candidate item sets generated is 45. Let us look at a method that introduces event items of the same type in an as needed fashion.

Let us assume a maximum of three event items of the same type allowed in a rule. For level 1 candidates we will only create a single item set for each event type. Although there are three event items of each type available we will only use the first item of each

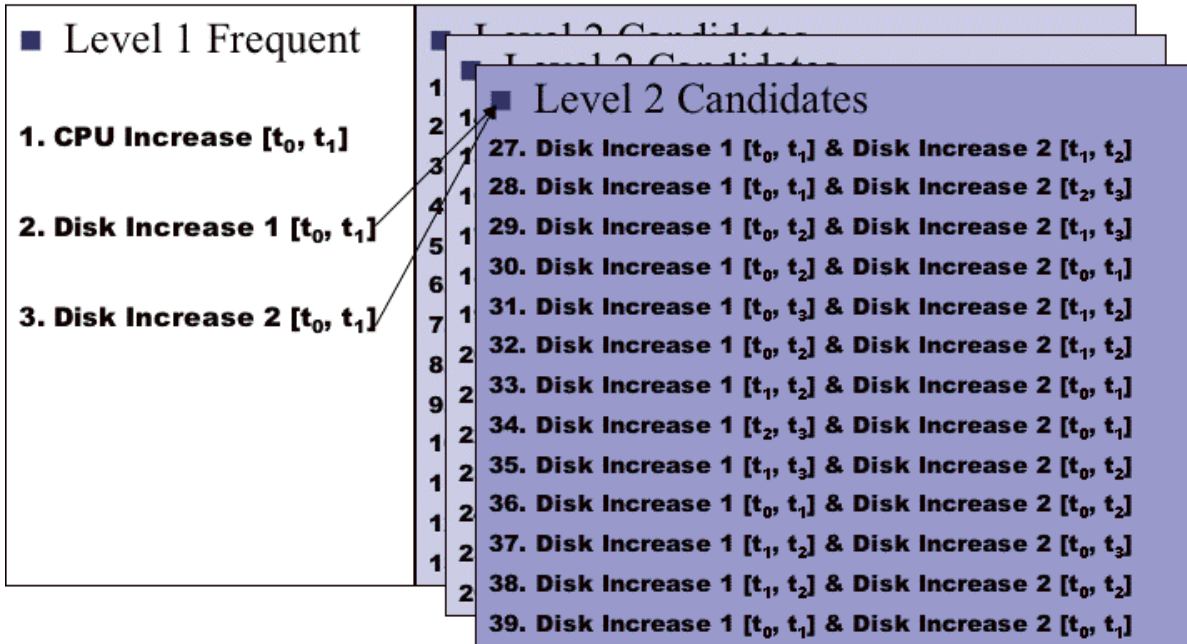


Figure 9.1: Candidates from Combining Item Sets Level 1

type to create the candidate item sets.

Using an example similar to the one above, when generating candidates for level 1 we will not include an item set which contains the second Disk Increase event.

To generate the level 2 candidates we simply combine the frequent item sets from level 1 as we normally would. Now we check if it is time to introduce the second event item of the same type. We check each frequent item set from the previous level to see if it contains enough event items of the same type to warrant adding another. For level 1 this means there must be at least one event of the same type in a frequent item set. This is trivial for level 1. For each frequent item set meeting this criteria we generate candidates by adding the additional event item of the same type to that frequent item set. A candidate item set will be generated for each possible temporal relationship the new event item could have with all the other event items in that item set.

Following along in our example, after the frequencies of the candidate item sets in level 1 are counted we find item set 1 and 2 are frequent. Figure 9.3 shows the result of combining item sets 1 and 2 to create candidates for level 2.

Since we know that Disk Increase 1 is frequent we know that Disk Increase 2 must be frequent. We will combine Disk Increase 2 with all the frequent item sets from level 1 that already contain a Disk Increase event. The level 2 candidates generated in this

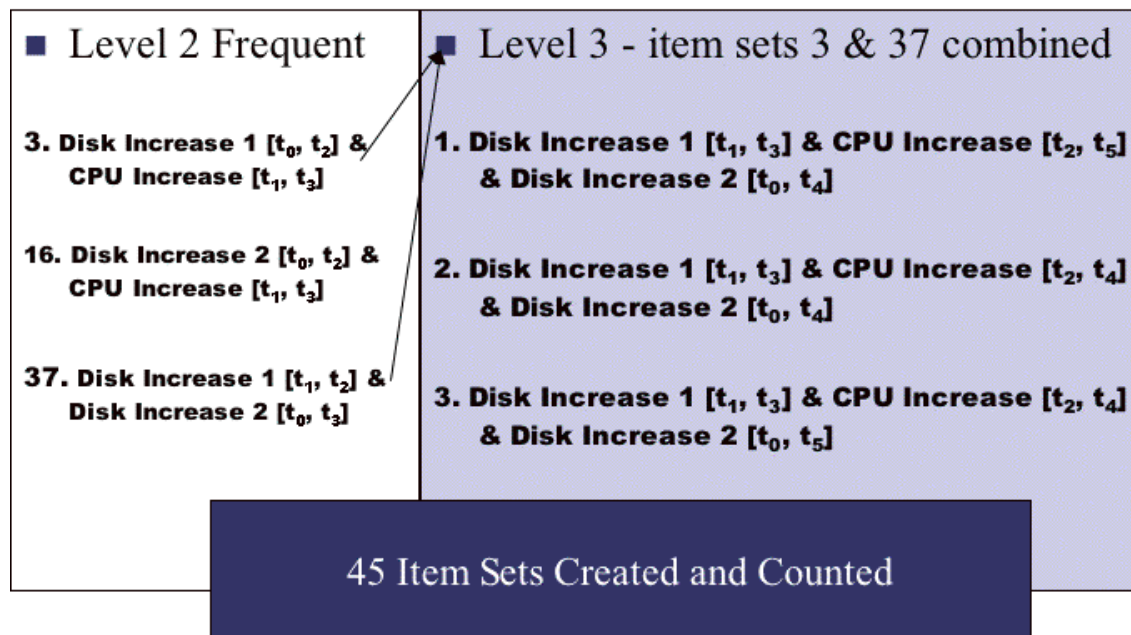


Figure 9.2: Candidates from Combining Item Sets Level 2

manner are shown in Figure 9.4. For simplicity we will not illustrate the second CPU Increase event.

After counting the candidate frequencies we find that item sets 3 and 24 are frequent. Note that item sets matching these were found frequent in the previous example. Figure 9.5 shows the candidates resulting from combining these two item sets. The 3 candidates are the same as the candidates for level 3 in the previous example. We came to this conclusion by generating and counting frequencies for 31 candidates instead of 45. This is a savings of 31%.

From here we would apply our criteria test to the frequent item sets of level 2 and see that item set 24 contains two Disk Increase events. We would then add the third available Disk Increase event to item set 24 to create candidates that contained three Disk Increase events. Each level's candidate generation hereafter would follow the same procedure.

There are considerations using this method. First, when generating candidates, all subsets of the candidates are checked to see if they are frequent. If one subset is not frequent then the candidate item set is known to also not be frequent via the Apriori principle. This subset check must treat event items of the same type as if they were the same item. That is, if item A is an event and was counted and found to be frequent

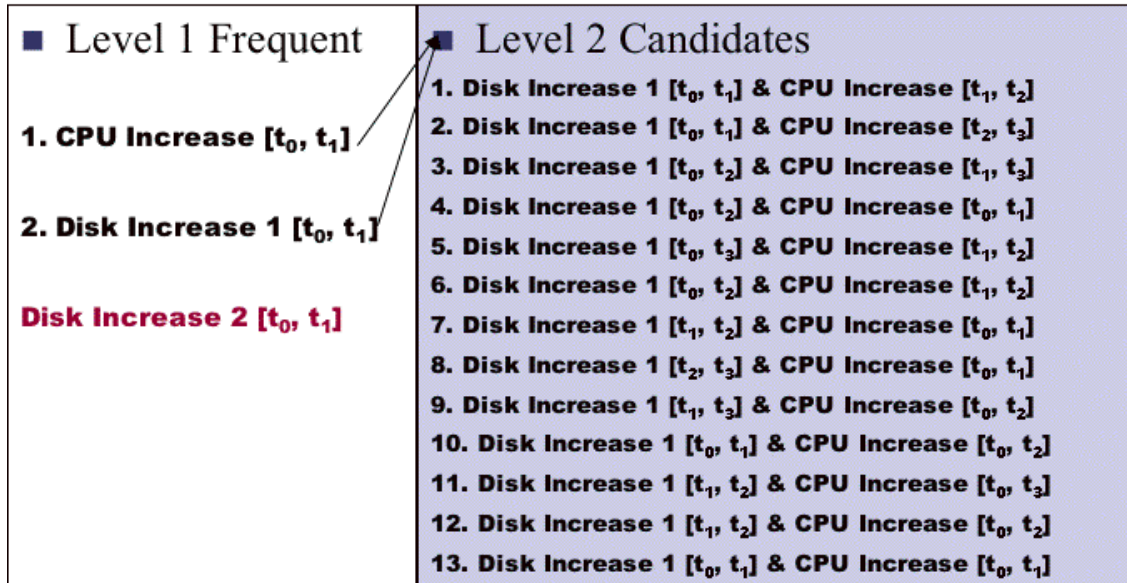


Figure 9.3: Candidates from Combining Item Sets Level 1a Improved

and item B is an event of the same type and was introduced into the candidates and not previously counted, it should also be known to be frequent. This can be easily done by modifying the Item Set Prefix Tree data structure to hold support values instead of references to actual item sets. The same type of problem occurs during rule generation. The support values of the antecedent of a rule and the consequent of a rule must be available during rule generation.

9.2.3 Improve Support Counting

To speed up support counting, a bit array could track if an instance counted toward the support of any item set. Instances that did not contain any item sets are guaranteed to not contain any item sets generated in the future and using them during support counting is a waste of time. They should be removed from the working data set. Keep in mind that the original total number of item sets should be stored so the support counting prune discussed in Section 7.3 can still be used.

Furthermore, keeping track of which data set instances contribute towards the support of an item set would save even more work for the algorithm. If an instance does not count towards the support of an item set it will not contribute towards the support of a superset of that item set. We can thus skip comparing that instance to future candidate item sets

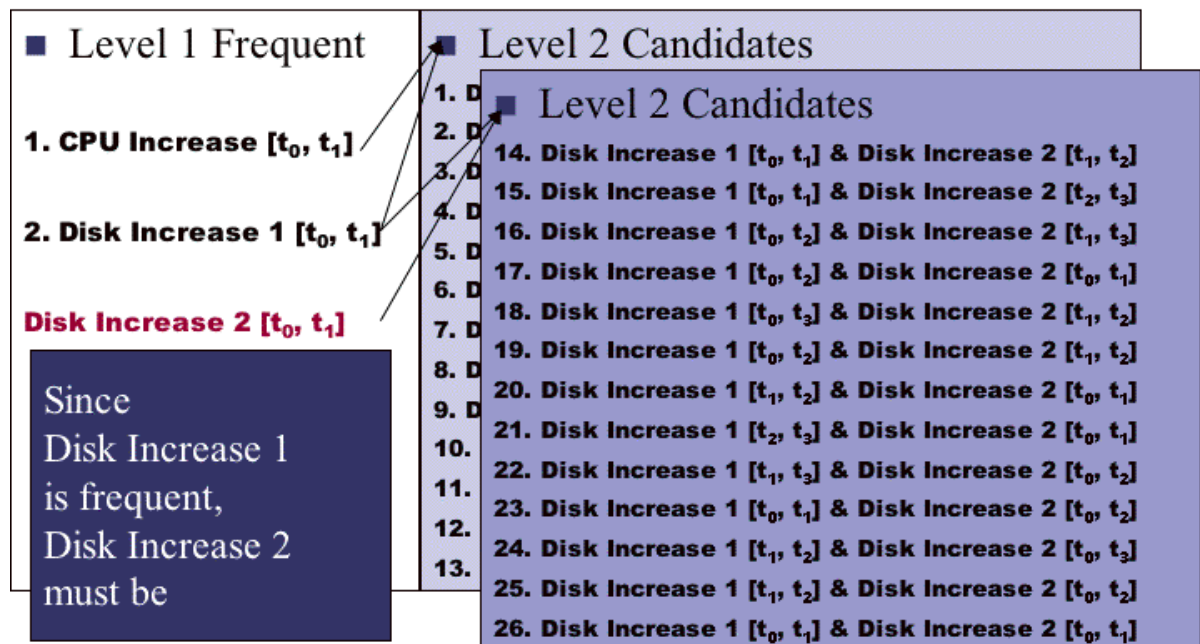


Figure 9.4: Candidates from Combining Item Sets Level 1b Improved

generated from those item sets.

9.2.4 System Features

Features not related directly to ASAS have been identified as useful and have been requested to be in the WPI Weka system. Here are those features which have not been implemented in the system to date.

Limit level of mining to maximum antecedent size plus maximum consequent size if both are specified. This would avoid mining for larger item sets which would not be used during rule generation anyway. If the maximum size of the antecedent is chosen to be 3 and the maximum size of the consequent is chosen to be 1, there is no need to mine for frequent item sets larger than 4. If there are larger frequent item sets to be found in a data set the time spent finding them will be wasted since only rule of maximum size 4 will be reported to the user.

Continue mining using cache file of frequent item sets to avoid counting support again for item sets with known support. Many times it is necessary to rerun a data mining task with a lower support or slightly modified data. Being able to use the support counts of frequent item sets from a previous run would save subsequent mining time. It would also be nice to not worry if the same exact data set is being used. That is, translate the

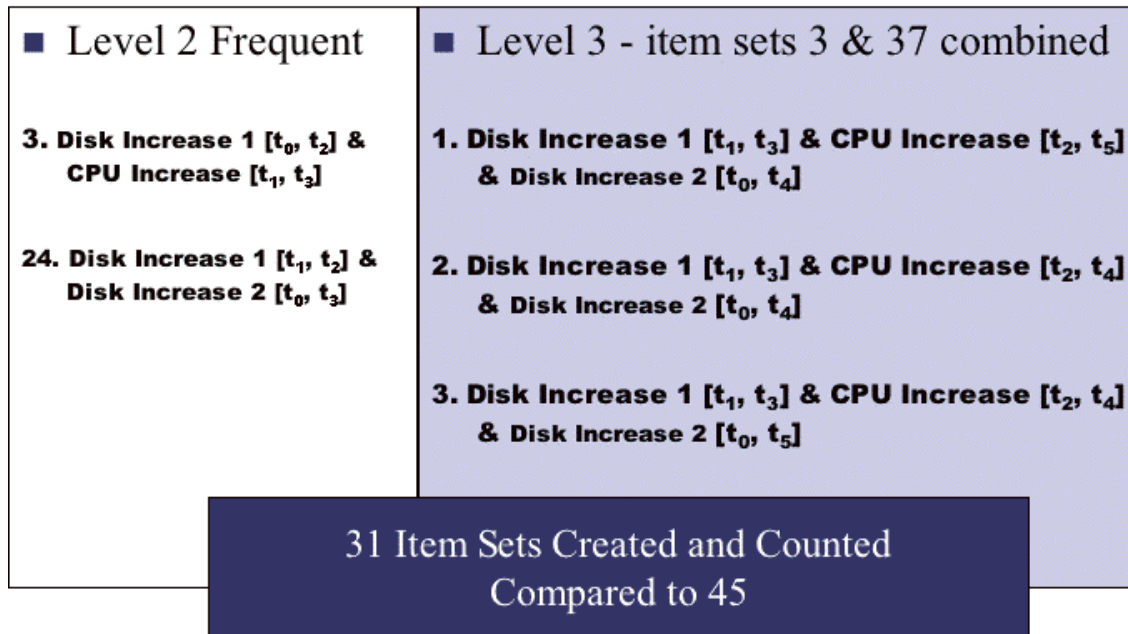


Figure 9.5: Candidates from Combining Item Sets Level 2 Improved

specific item numbers from the cache file to the item numbers used in the current data set. Have to warn user that the instances should be the same and that only the list of attributes in the data set can be different.

The character '=' cannot currently be used in an attribute name. This causes the value of an event item to be interpreted incorrectly. The value being taken to be the string following the '='. When the conversion from attribute value pair to item representation is made a map of the attribute number and attribute name should be made so that ASAS can know where to take the value from.

Bibliography

- [AAP00] Ramesh C. Agarwal, Charu C. Aggarwal, and V.V.V. Prasad. Depth first generation of long patterns. In *Proc. of the Sixth ACM SIGKDD Conference on Knowledge Discover y and Data Mining*, pages 108–118, Boston, MA, Aug. 2000. ACM.
- [AAP01] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001. cite-seer.ist.psu.edu/agarwal99tree.html.
- [AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms (FODO) Conference*, Evanston, Illinois, Oct. 1993.
- [AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993. ACM.
- [All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 1983.
- [Alv03] Sergio A. Alvarez. Chi-squared computation for association rules: Preliminary results. Technical Report BCCS-03-01, Computer Science Department, Boston College, July 2003.
- [AR00] Juan M. Ale and Gustavo H. Rossi. An approach to discovering temporal association rules. In Janice Carroll, Ernesto Damiani, Hisham Haddad, and David Oppenheim, editors, *2000 ACM Symposium on Applied Computing*, volume 1, pages 294–300, Como, Italy, 2000. ACM.

- [AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, Santiago, Chile, 1994.
- [AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *International Conference on Database Engineering*, pages 3–14. IEEE, 1995.
- [AS96] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Trans. On Knowledge And Data Engineering*, 8:962–969, 1996.
- [BAG00] Roberto J. Bayardo, Rakesh Agrawal, and Dimitrios Gunopulos. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2/3):217–240, July 2000.
- [BC95] D.J. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 229–248. AAAI Press/ MIT Press, 1995.
- [BFG⁺03] John Baird, Jay Farmer, Rebecca Gougian, Ken Monterio, and Paul Young. Motif elicitation and multipoint analysis of gene expression. Undergraduate Graduation Project (MQP). Worcester Polytechnic Institute, April 2003.
- [BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [BMS98] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2:39–68, 1998.
- [BSJ96] C. Bettini, X. Sean Wang, and S. Jajodia. Testing complex temporal relationships involving multiple granularities and its application to data mining. In ACM, editor, *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1996, Montréal, Canada, June 3–5, 1996*, volume 15, pages 68–78, New York, NY 10036, USA, 1996. ACM Press.
- [BWJL98] Claudio Bettini, X. Sean Wang, Sushil Jajodia, and Jia-Ling Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):222–237, 1998.

- [CDF⁺00] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. In *Proc. of Int'l. Conf. on Data Engineering (ICDE2000)*, pages 489–499, 2000.
- [Cor] Microsoft Corporation. Msdn home page. <http://msdn.microsoft.com/>.
- [CP99] X. Chen and I. Petrounias. Mining temporal features in association rules. In J.M. Zytkow and J. Rauch, editors, *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, volume 1704 of *Lecture Notes in Artificial Intelligence*, pages 295–300, Prague, 1999. Springer.
- [DLM⁺98] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 16–22. ACM, 1998.
- [DP01] W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 67–76. ACM Press, 2001.
- [Dun03] Margaret H. Dunham. *Data Mining, Introduction and Advanced Topics*. Pearson Education, Inc., 2003. From the Southern Methodist University, Chapter 7, Web Mining, pp. 195-219, Chapter 9, Temporal Mining, pp. 245-273.
- [Fin] Yahoo! Finance. Yahoo! finance. <http://finance.yahoo.com/>.
- [FLYH99] Ling Feng, Hongjun Lu, J. X. Yu, and Jiawei Han. Mining inter-transaction associations with templates. In *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, Kansas City, Missouri, USA, November 2-6, 1999*, pages 225–233. ACM, 1999.
- [FM94] M. Ranganathan Faloutsos and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD*, pages 419–429, Minneapolis, MN, May 1994.
- [Fre04] Jonathan Freyberger. Using association rules to guide a search for best fitting transfer models of student learning. Master's thesis, Department of Computer Science, Worcester Polytechnic Institute, April 2004.

- [FW00] E. Frank and I. H. Witten. *Data Mining*. Morgan Kaufmann Publishers, 2000. From the University of Waikato, New Zealand.
- [Hal00] Tara Halwes. Transform-based similarity methods for sequence mining. Undergraduate Graduation Project (MQP). Worcester Polytechnic Institute, Aug. 2000. Winner, Provost's MQP Award for Computer Science.
- [Hid98] C. Hidber. Online association rule mining. Technical Report CSD-98-1004, Dept. of Electrical Engineering and Computer Science, Univ. of California at Berkeley, May 1998.
- [HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.
- [HL03] Sam Holmes and Cindy Leung. Exploring temporal associations in the stock market. Undergraduate Graduation Project (MQP). Worcester Polytechnic Institute, April 2003.
- [HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. Int'l. Conf. of Management of Data (SIGMOD2000)*, pages 1–12, May 2000.
- [Lai93] P. Laird. Identifying and using patterns in sequential data. In Klaus P. Jantke, Shigenobu Kobayashi, Etsuji Tomita, and Takashi Yokomori, editors, *Proceedings of The 4th International Workshop on Algorithmic Learning Theory, ALT '93, Tokyo, Japan, November 8-10, 1993*, Lecture Notes in Computer Science, pages 1–18, Tokyo, Japan, 1993. Springer.
- [Lax04] Parameshvyas Laxminarayan. Exploratory analysis of human sleep data. Master's thesis, Department of Computer Science, Worcester Polytechnic Institute, January 2004.
- [LFH00] Hongjun Lu, Ling Feng, and Jiawei Han. Beyond intratransaction association analysis: mining multidimensional intertransaction association rules. *ACM Transactions on Information Systems (TOIS)*, 18(4):423–454, 2000.
- [Lin98] D-I Lin. *Fast Algorithms for Discovering the Maximum Frequent Set*. PhD thesis, Dept. of Computer Science. New York University, 1998.
- [LR91] Jeffrey Little and Lucien Rhodes. *Understanding Wall Street*. Liberty Hall Press and McGraw-Hill Trade, 3rd edition, 1991.

- [LWJ00] Yingjiu Li, X. Sean Wang, and Sushil Jajodia. Discovering temporal patterns in multiple granularities. In J. F. Roddick and K. Hornsby, editors, *International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining, TSDM2000*, volume 2007 of *Lecture Notes in Artificial Intelligence*, Lyon, France, 2000. Springer.
- [MT96] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proc. of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 146–151, Portland, Oregon, August 1996. AAAI Press.
- [MTV95] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering Frequent Episodes in Sequences. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Canada, 1995. AAAI Press.
- [OSGC95] Tim Oates, Matthew D. Schmill, Dawn E. Gregory, and Paul R. Cohen. Detecting complex dependencies in categorical data. Technical Report UM-CS-1994-081, University of Massachusetts, Amherst, MA, , 1995. cite-seer.nj.nec.com/oates94detecting.html.
- [RMS98] Sridhar Ramaswamy, Sameer Mahajan, and Abraham Silberschatz. On the discovery of interesting patterns in association rules. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 368–379. Morgan Kaufmann, 1998.
- [RR99] C.P. Rainsford and John F. Roddick. Adding temporal semantics to association rules. In J.M. Zytkow and J. Rauch, editors, *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, volume 1704 of *Lecture Notes in Artificial Intelligence*, pages 504–509, Prague, 1999. Springer.
- [RS01] John F. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, 13, 2001.

- [RS02] J.F. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):750–767, 2002.
- [SA96] R. Srikant and R Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Int’l Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.
- [Sho01] C.A. Shoemaker. Mining association rules from set-valued data. Master’s thesis, Department of Computer Science, Worcester Polytechnic Institute, May 2001.
- [SR03] C. Shoemaker and C. Ruiz. Association rule mining algorithms for set-valued data. In J. Liu, Y. Cheung, and H. Yin, editors, *Proc. of the Fourth International Conference on Intelligent Data Engineering and Automated Learning*, pages 669–676. Lecture Notes in Computer Science. Vol. 2690. Springer-Verlag, June 2003.
- [SS02] Zachary Stoecker-Sylvia. Merging the association rule mining modules of the weka and arminer data mining systems. Undergraduate Graduation Project (MQP). Worcester Polytechnic Institute, April 2002.
- [TLHF03] Anthony K.H. Tung, Hongjun Lu, Jiawei Han, and Ling Feng. Efficient mining of intertransaction association rules. *IEEE Transactions On Knowledge And Data Engineering*, 15(1):43–56, January/February 2003.
- [WBY03] Xintao Wu, Daniel Barbará, and Yong Ye. Screening and interpreting multi-item associations based on log-linear modeling. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 276–285. ACM Press, 2003.
- [Web00] Geoffrey I. Webb. Efficient search for association rules. In *Proc. of the Sixth ACM SIGKDD Conference on Knowledge Discover y and Data Mining*, pages 99–107, Boston, MA, Aug. 2000. ACM.
- [XD99] Yongqiao Xiao and Margaret H. Dunham. Considering main memory in mining association rules. In *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery*, pages 209–218, November 1999.

- [Zak00] M. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *Proc. Int. Conference on Information and Knowledge Management (CIKM)*, pages 422–429. ACM, 2000.
- [ZKM01] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of the Seventh ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.
- [ZOPL96] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In *CD-ROM Proceedings of Supercomputing'96*, Pittsburgh, PA, November 1996. IEEE.
- [ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, page 283. AAAI Press, 1997.

Appendix A

Appendix Readme File

Topics:

1. Running Weka
 - 1.1 Adding to the Weka GUI
 - 1.2 Using the GUI
 - 1.3 Using the Command Line
2. Data Sets Containing Time Sequence Attributes
 - 2.1 Time Sequence Attributes
 - 2.2 Filtering Events
 - 2.3 Event Weight
 - 2.4 Tips and Tricks
3. Detailed Documentation

1. Running Weka

1.1 Adding to the Weka GUI

To include a new classifier, filter, association algorithm, etc in the Weka GUI follow these steps.

1. If there is not a GenericObjectEditor.props file in the same directory as this Readme.txt file copy the one found in weka/gui/.
2. To this file add lines to the appropriate section. An example of such lines can be found in the AddToGenericObjectEditor.props file also located in the same directory as this Readme.txt file.
3. If you are adding the WPI Weka add on package to Weka please add the lines found in AddToGenericObjectEditor.props to the specified sections in the GenericObjectEditor.props file you copied to this directory.

1.2 Using the GUI

To run Weka use the following command line from the directory containing this Readme.txt file:

```
java -classpath . weka/gui/GUIChooser
```

If you run into "out of memory" problems use the -Xmx flag as follows:

```
java -Xmx1024m -classpath . weka/gui/GUIChooser
```

The previous line allocates 1024 MB of memory to the java virtual machine. Choose the amount that works best for you.

To use the AprioriSetsAndSequences algorithm first select the Weka Explorer interface from the GUI Chooser window.

Under the Preprocess tab in the Explorer interface select "Open File" and browse to the data set you wish to use.

Note: it is possible to use other data sources besides a file but I haven't used them so you'd be on your own.

Switch to the Associate tab and click on the Associator algorithm shown. This will pop a window up showing the current algorithm and user specified parameters.

Choose AprioriSetsAndSequences from the drop down list. By clicking the "More" button or hovering the mouse pointer over each parameter you can get an explanation for each option.

Press "Start" and the system will begin mining for association rules. In the command line window you started Weka in you'll see status and messages about the mining progress.

1.3 Using the Command Line

You can also run AprioriSetsAndSequences directly from the command line. An example is shown below:

```
java -Xmx1024m -classpath . wpi/associations/AprioriSetsAndSequences
-t perfddata-with-increase-decrease-events.arff
```

The full list of command line options is below. You can get a list of these options by simply running AprioriSetsAndSequences with no arguments.

AprioriSetsAndSequences options:

- t <training file>
The name of the training file.
- N <required number of rules output>
The required number of rules. (default = 10)
- C <minimum confidence score of a rule>
The minimum confidence of a rule. (default = 0.9)
- D <delta for minimum support>
The delta by which the minimum support is decreased in each iteration. (default = 0.05)
- U <upper bound for minimum support>
The upper bound for the minimum support. (default = 1.0)
- M <lower bound for minimum support>
The lower bound for the minimum support. (default = 0.1)
- A <range of required antecedent attributes>
The set of attributes required to be in the antecedent.

- Y <range of required consequent attributes>
The set of attributes required to be in the consequent.
- B <maximum antecedent items>
The maximum number of attributes in the antecedent. (0 = no maximum)
- E <minimum antecedent items>
The minimum number of attributes in the antecedent. (0 = no minimum)
- W <maximum events of same type allowed in a rule>
The maximum number of event items of the same type allowed in a rule. (default = 2)
- X <maximum consequent items>
The maximum number of attributes in the consequent. (0 = no maximum)
- Z <minimum consequent items>
The minimum number of attributes in the consequent. (0 = no minimum)
- F <cache file name>
The file from which to load frequent item sets from

2. Data Sets Containing Time Sequence Attributes

This Section includes useful information for using building and using data sets containing time sequence attributes.

Please Note:

AprioriSetsAndSequences will run with normal data sets and data sets containing time sequence events.

It will NOT handle data sets that contain numeric attributes.

It will NOT recognize an event attribute if the attribute name contains a "=".

2.1 Time Sequence Attributes

In the arff file format Weka uses declare a time sequence attribute as follows:

```
@attribute 'my-time-sequence-attribute' string
```

Note: "=" can NOT be used in the attribute name.

Each value in a sequence is separated by colons ":". An example value of a time sequence attribute is shown below:

```
0:1:2:3:4:5:6:7:8:9
```

The values should be numeric in most cases.

While it is possible to have symbolic sequences the current system provides filters that recognize events for numeric sequences only. If you wish to use data consisting of symbolic sequences you must also provide the events or filters that recognize symbolic events.

You must transform time sequences into Events in order to use the `AprioriSetsAndSequences` algorithm. More information on Events can be found in the next section.

2.2 Filtering For Events

Provided in the `wpi` package that currently accompanies the `weka` package are the various utilities and functionality needed to work with time sequences.

This includes a set of filters that can be applied to data sets containing numeric time sequence attributes. These filters detect events (Increase, Decrease, and Sustain) which are simply predetermined sequence patterns.

A new attribute is created for each event type and time sequence attribute.

A time sequence event attribute is declared in `arff` as follows:

```
@attribute 'my-time-sequence-attribute-my-events' string
```

The begin time and end time of each event is noted, separated by a colon ":" and put into a set. Each event in the set is separated by a caret "^". An example follows:

```
'{1:3^4:6^7:8}'
```

The event attribute value above contains three events of the "my-events" type. The events begin and end at time 1 and 3, 4 and 6, and 7 and 8, respectively.

This EventFilter is available from the Weka GUI and can be used like any other Weka filter. Open the data file you wish to filter in the Weka GUI, select the EventFilter from the filter drop down list, specify the options you want to use and Add the filter to list of filters Weka will apply to the dataset. Press "Apply Filters".

To apply a filter to your data set using the command line:

```
java -classpath . wpi/filters/EventFilter -i perpdata.arff
-o perpdata-increase-decrease-events.arff -I -D
```

The above command specifies the input file as perpdata.arff and the output file as perpdata-increase-decrease-events.arff. -I and -D specify the events to search for. A complete list of options are below.

Filter options:

```
-A <value>
    Specify the attributes to find events in.
-I
    Find increase events.
-D
    Find decrease events.
-S
    Find sustain events.
-T <value>
    Specifies the tolerance for including values in events.
-N <value>
    Specifies the minimum required number of values in events.
```

General options:

```
-h
    Get help on available options.
    (use -b -h for help on batch mode.)
-i <file>
    The name of the file containing input instances.
    If not supplied then instances will be read from stdin.
-o <file>
    The name of the file output instances will be written to.
    If not supplied then instances will be written to stdout.
-c <class index>
```

The number of the attribute to use as the class.
"first" and "last" are also valid entries.
If not supplied then no class is assigned.

If your time sequence attributes are not numeric or the existing types of events does not fit your needs you can create your own filters or provide just the event attributes the data set you use with Weka. If you provide your own filter you can add it to the Weka GUI by modifying the GenericObjectEditor.props file contained in the same directory as this Readme.txt file. Also add your entry to the AddToGenericObjectEditor.props file to make upgrading to new Weka versions easier.

2.3 Event Weight

In addition to support and confidence a new rule metric has been added. Event weight is the number of times all the events in a rule appear in the data set. This can be more the number of instances in the data set. If both the antecedent and the consequent of the rule have events then the event weight is used to calculate confidence.

2.4 Tips and Tricks

-Remove sequences from data set before mining for association rules.

Since the events are used for the actual mining process and there is very little chance of entire sequences being used in a rule it saves a lot of mining time to simply remove the sequences from the data set before mining.

-The events you use are very specific to your domain.

If the events you use have no meaning in your data set's domain then the rules will have just as much meaning.

3. Detailed Documentation

Detailed documentation on all the WPI Weka code can be found
in the docs directory.

This Readme.txt file brought to you by Keith A. Pray. Feel free to
send questions to kap@wpi.edu if you encounter any problems running
AprioriSetsAndSequences or find this Readme.txt file lacking.

Good luck,
Keith

Appendix B

Appendix ASAS Log Metrics

AVERAGE LENGTH OF TIMELINE Average length of a time line in the data set

CONFIDENCE Confidence metric.

MAX EVENT LENGTH Maximum length of the events.

MAX PER EVENT LENGTH Maximum length per event.

MIN EVENT LENGTH Minimum length of the events.

MIN PER EVENT LENGTH Minimum length per event.

NUM ATTRIBUTES Number of attributes present in the data set

NUM CANDIDATES GENERATED Number of candidate item sets of size k generated

NUM COMPARISONS SAVED... REMOVING CANDIDATES DURING SUPPORT COUNTING - Number of times a candidate item sets did not have to be compared to instances in the data set because the candidate was removed from the candidate

list when it was clear it could not be considered frequent due to the number of instances the candidate was already compared to

NUM EVENT ATTRIBUTES Number of event attributes present in the data set

NUM EVENT OCCURRENCES Number of times events occur in the data set

NUM EVENTS OF SAME TYPE ALLOWED Number of event items of the same type allowed to appear in a rule

NUM FREQUENT ITEMSETS FOUND Number of frequent item sets of size k found

NUM INSTANCES Number of instances in the data set

NUM OCCURRENCES PER ATTRIBUTE Average number of event occurrences per event attribute

NUM RULES Number of association rules found.

NUM UNIQUE CANDIDATES GENERATED Number of unique item sets candidates of size k generated

NUM UNIQUE FREQUENT ITEMSETS FOUND Number of frequent unique item sets of size k found

RELATION NAME Relation name of the data set.

SECONDS PER FREQUENT ITEMSET Total mining seconds per frequent item set found.

SECONDS TO COUNT SUPPORT Number of seconds to count support of candidate item sets of size k

SECONDS TO FIND CONFIDENCE Number of seconds to calculate find confidence of association rules.

SECONDS TO GENERATE CANDIDATES Number of seconds to generate candidates item sets of size k

SECONDS TO GENERATE RULES Number of seconds to generate association rules from frequent item sets

SUPPORT Support metric.

TIMESTAMP Date and time when the mining started.

TOTAL NUM FREQUENT ITEMSETS Total number of frequent item set found

TOTAL SECONDS TO MINE Number of seconds for the entire frequent item set mining

USE ITEMSET PREFIX TREE Specifies if item set prefix tree used instead of vector.

USE PREVIOUS DUPLICATE HASH Specifies if previous duplicate hash used.

Appendix C

Appendix Computer Performance

Metrics

Memory % Committed Bytes In Use shows the ratio of Memory Committed Bytes to the Memory Commit Limit. Committed memory is physical memory in use for which space has been reserved in the paging file so that it can be written to disk. The commit limit is determined by the size of the paging file. If the paging file is enlarged, the commit limit increases, and the ratio is reduced.

Memory Available Bytes Shows the amount of physical memory, in bytes, available to processes running on the computer. It is calculated by summing the amount of space on the zeroed, free, and standby memory lists. Free memory is ready for use; zeroed memory consists of pages of memory filled with zeros to prevent later processes from seeing data used by a previous process; standby memory is memory that has been removed from a process's working set (its physical memory) en route to disk but is still available to be recalled.

PhysicalDisk Current Disk Queue Length Shows the number of requests that are outstanding on the disk at the time that the performance data is collected. It

includes requests in service at the time of the collection. This is a snapshot, not an average over the time interval. It includes requests in service at the time of the collection. Multispindle disk devices can have multiple requests active at one time, but other concurrent requests are awaiting service. This counter might reflect a transitory high or low queue length, but if there is a sustained load on the disk drive, it is likely that this is consistently high. Requests experience delays proportional to the length of this queue minus the number of spindles on the disks. This difference should average less than two.

This seems to be not the metric I meant to capture. The data collection experiments will have to be re-run to get better disk information.

Process % Processor Time Shows the percentage of elapsed time that all of the threads of this process used the processor to execute instructions. An instruction is the basic unit of execution in a computer; a thread is the object that executes instructions; and a process is the object created when a program is run. Code executed to handle some hardware interrupts and trap conditions are included in this count.

Process Working Set Shows the current number of bytes in the working set of this process. The working set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a certain threshold, pages are left in the working set of a process even if they are not in use. When free memory falls below a certain threshold, pages are trimmed from working sets. If they are needed, they are then soft-faulted back into the working set before they leave main memory.

Processor Total % Processor Time Shows the percentage of time that the processor is executing application or operating system processes other than Idle. This counter is a primary indicator of processor activity. It is calculated by measuring the time

that the processor spends executing the thread of the Idle process in each sample interval, and subtracting that value from 100%. Each processor has an Idle thread which consumes cycles when no other threads are ready to run.

System Processes Total number of processes currently running.

System Threads Total number of threads currently running.

Appendix D

Appendix Computer Performance Data

Descriptions of the attributes of the data set used for ASAS evaluation over computer system performance data are given here. Some attributes describe the processes running on the machine the data was collected from. For each description of these attributes there are actually an attribute of that type for each process. These attribute types are noted with a '*' where the process name would appear. The list of processes is as follows: agntrsvc, Avsynmgr, bash, bash#1, bash#2, CMD, CMD#1, CSRSS, dbsnmp, explorer, explorer#1, Idle, java, LSASS, make, mgabg, MsPMSPSv, mstask, net, net1, ntpd, nutsv4, pdesk, realsched, regsvc, rnathchk, scrnsave.scr, SERVICES, sleep, smlogsvc, SMSS, spoolsv, svchost, svchost#1, svchost#2, svchost#3, System, taskmgr, _Total, wcescomm, WINLOGON, WinMgmt.

For descriptions of the Windows Performance Monitor Metrics please see Appendix C.

option-M Data type: Nominal. Possible values: { (-inf-3.00009, 3.00009-6.00008, 6.00008-inf } Discretized value of the M option for neural networks back propagation.

option-I Data type: Nominal. Possible values: { All } Discretized value of the I option for neural networks back propagation.

option-H Data type: Nominal. Possible values: { All } Discretized value of the H option for neural networks back propagation.

option-E Data type: Nominal. Possible values: { All } Discretized value of the E option for neural networks back propagation.

option-U Data type: Nominal. Possible values: { -U, none } The U option flag for decision trees.

option-R Data type: Nominal. Possible values: { -R } The R option flag for decision trees.

option-S Data type: Nominal. Possible values: { -S } The S option flag for decision trees.

training-data Data type: Nominal. Possible values: { census-income.data, census-income-normalized.data, census-income-discretized.data, census-income-normalized-discretized.data, nomissing-census-income.data, dcensus-income-normalized-5000.data } The data set file used as input for the machine learning task.

learning-algorithm Data type: Nominal. Possible values: { weka.classifiers.j48.J48, IBk, weka.classifiers.NaiveBayes, NaiveBayesSimple, BackPropagation } The machine learning algorithm used for the machine learning task.

Process(*)% Processor Time-increase-events Data type: Event Set. Specifies where in the Process % Processor Time sequence increase events have been detected.

Process(*)% Processor Time-decrease-events Data type: Event Set. Specifies where in the Process % Processor Time sequence decrease events have been detected.

Process(*)% Processor Time-sustain-events Data type: Event Set. Specifies where in the Process % Processor Time sequence sustain events have been detected.

Process(*)Working Set-increase-events Data type: Event Set. Specifies where in the Process Working Set sequence increase events have been detected.

Process(*)Working Set-decrease-events Data type: Event Set. Specifies where in the Process Working Set sequence decrease events have been detected.

Process(*)Working Set-sustain-events Data type: Event Set. Specifies where in the Process Working Set sequence sustain events have been detected.

Memory% Committed Bytes In Use-increase-events Data type: Event Set. Specifies where in the Memory % Committed Bytes In Use sequence increase events have been detected.

Memory% Committed Bytes In Use-decrease-events Data type: Event Set. Specifies where in the Memory % Committed Bytes In Use sequence decrease events have been detected.

Memory% Committed Bytes In Use-sustain-events Data type: Event Set. Specifies where in the Memory % Committed Bytes In Use sequence sustain events have been detected.

MemoryAvailable Bytes-increase-events Data type: Event Set. Specifies where in the Memory Available Bytes sequence increase events have been detected.

MemoryAvailable Bytes-decrease-events Data type: Event Set. Specifies where in the Memory Available Bytes sequence decrease events have been detected.

MemoryAvailable Bytes-sustain-events Data type: Event Set. Specifies where in the Memory Available Bytes sequence sustain events have been detected.

PhysicalDisk(0 C)Current Disk Queue Length-increase-events Data type: Event Set. Specifies where in the Physical Disk(0 C) Current Disk Queue Length sequence increase events have been detected.

PhysicalDisk(0 C)Current Disk Queue Length-decrease-events Data type: Event Set. Specifies where in the Physical Disk(0 C) Current Disk Queue Length sequence decrease events have been detected.

PhysicalDisk(0 C)Current Disk Queue Length-sustain-events Data type: Event Set. Specifies where in the Physical Disk(0 C) Current Disk Queue Length sequence sustain events have been detected.

PhysicalDisk(_Total)Current Disk Queue Length-increase-events Data type: Event Set. Specifies where in the Physical Disk(_Total) Current Disk Queue Length sequence increase events have been detected.

PhysicalDisk(_Total)Current Disk Queue Length-decrease-events Data type: Event Set. Specifies where in the Physical Disk(_Total) Current Disk Queue Length sequence decrease events have been detected.

PhysicalDisk(_Total)Current Disk Queue Length-sustain-events Data type: Event Set. Specifies where in the Physical Disk(_Total) Current Disk Queue Length sequence sustain events have been detected.

Processor(_Total)% Processor Time-increase-events Data type: Event Set. Specifies where in the Processor (_Total) % Processor Time sequence increase events have been detected.

Processor(_Total)% Processor Time-decrease-events Data type: Event Set. Specifies where in the Processor (_Total) % Processor Time sequence decrease events have been detected.

Processor(_Total)% Processor Time-sustain-events Data type: Event Set. Specifies where in the Processor (_Total) % Processor Time sequence sustain events have been detected.

SystemProcesses-increase-events Data type: Event Set. Specifies where in the System Processes sequence increase events have been detected.

SystemProcesses-decrease-events Data type: Event Set. Specifies where in the System Processes sequence decrease events have been detected.

SystemProcesses-sustain-events Data type: Event Set. Specifies where in the System Processes sequence sustain events have been detected.

SystemThreads-increase-events Data type: Event Set. Specifies where in the System Threads sequence increase events have been detected.

SystemThreads-decrease-events Data type: Event Set. Specifies where in the System Threads sequence decrease events have been detected.

SystemThreads-sustain-events Data type: Event Set. Specifies where in the System Threads sequence sustain events have been detected.

Appendix E

Appendix Stock Market Data

Descriptions of the attributes of the data set used for ASAS evaluation over company financial and stock market data are given here. These attributes describe companies stock market values over time as well as other interesting events. For each description of these attributes there are actually an attribute of that type for each company in the data set. These attributes are noted with a '*' where the company name would appear. The list of companies is as follows: AMD, Cisco (CSCO), Intel (INTL), Microsoft (MSFT), Nextel (NXTL), Oracle (ORCL), SUN Microsystems (SUNW).

- * **product release** Data type: Event Set. Specifies when company announces a new product.
- * **award** Data type: Event Set. Specifies when company announces it has won an award.
- * **bad news** Data type: Event Set. Specifies when the company has bad news.
- * **expand merge** Data type: Event Set. Specifies when the company undergoes an expansion or merger.
- * **rounded top.tpl** Data type: Event Set. Specifies when a rounded top event, shown in Figure E.1, was detected in the company's stock value.

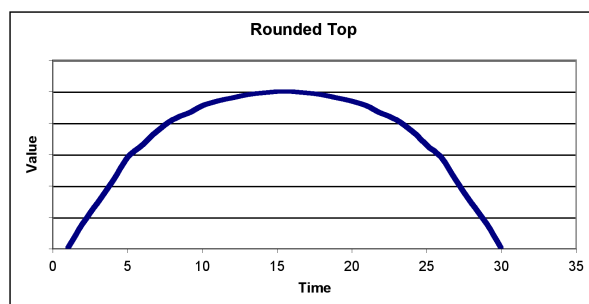


Figure E.1: Rounded Top

- * **selling climax.tpl** Data type: Event Set. Specifies when a selling climax event, shown in Figure E.2, was detected in the company's stock value.

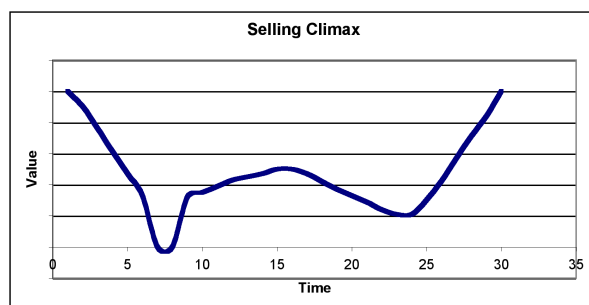


Figure E.2: Selling Climax

- * **asc triangle.tpl** Data type: Event Set. Specifies when a ascending triangle event, shown in Figure E.3, was detected in the company's stock value.
- * **broadening top.tpl** Data type: Event Set. Specifies when a broadening top event, shown in Figure E.4, was detected in the company's stock value.
- * **desc triangle.tpl** Data type: Event Set. Specifies when a descending triangle event, shown in Figure E.5, was detected in the company's stock value.
- * **double bottom.tpl** Data type: Event Set. Specifies when a double bottom event, shown in Figure E.6, was detected in the company's stock value.

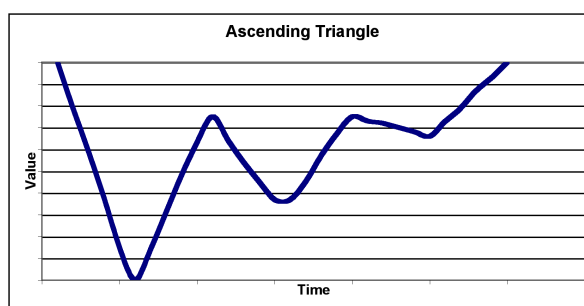


Figure E.3: Ascending Triangle

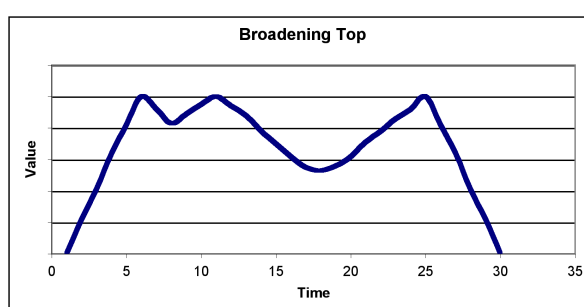


Figure E.4: Broadening Top

- * **double top.tpl** Data type: Event Set. Specifies when a double top event, shown in Figure E.7, was detected in the company's stock value.
- * **hs.tpl** Data type: Event Set. Specifies when a head and shoulders event, shown in Figure E.8, was detected in the company's stock value.
- * **inverse hs.tpl** Data type: Event Set. Specifies when a inverse head and shoulders event, shown in Figure E.9, was detected in the company's stock value.
- * **panic rev.tpl** Data type: Event Set. Specifies when a panic reversal event, shown in Figure E.10, was detected in the company's stock value.
- * **rounded bottom.tpl** Data type: Event Set. Specifies when a rounded bottom event, shown in Figure E.11, was detected in the company's stock value.

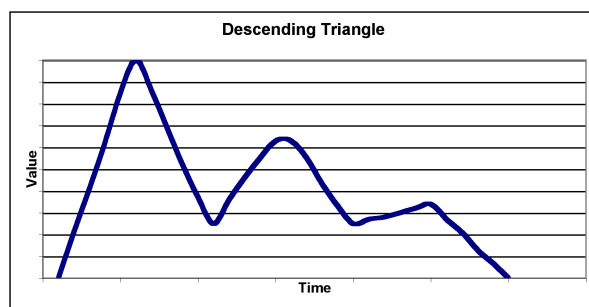


Figure E.5: Descending Triangle

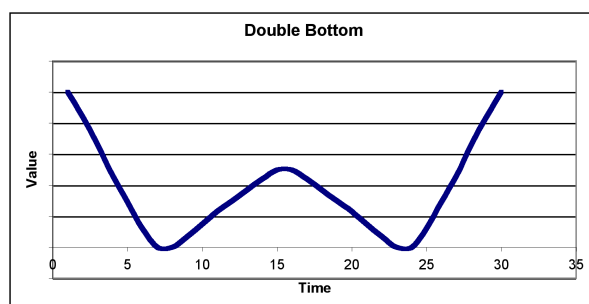


Figure E.6: Double Bottom

- * **triple bottom.tpl** Data type: Event Set. Specifies when a triple bottom event, shown in Figure E.12, was detected in the company's stock value.
- * **triple top.tpl** Data type: Event Set. Specifies when a triple top event, shown in Figure E.13, was detected in the company's stock value.
- * **sustain.tpl** Data type: Event Set. Specifies when a sustain event, shown in Figure E.14, was detected in the company's stock value.
- * **increase.tpl** Data type: Event Set. Specifies when an increase event, shown in Figure E.15, was detected in the company's stock value.
- * **decrease.tpl** Data type: Event Set. Specifies when a decrease event, shown in Figure E.16, was detected in the company's stock value.

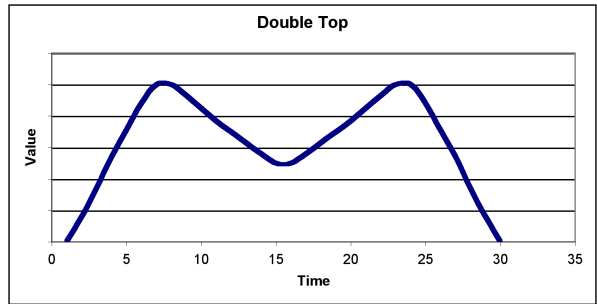


Figure E.7: Double Top

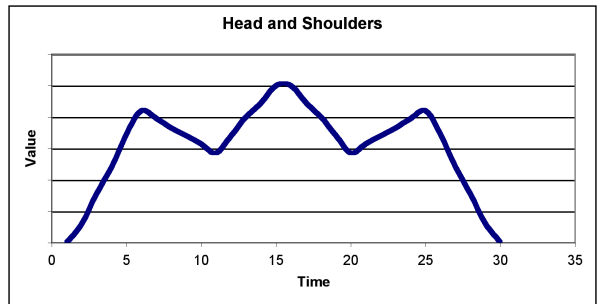


Figure E.8: Head and Shoulders

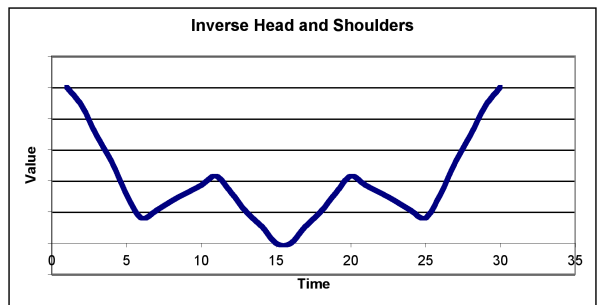


Figure E.9: Inverse Head and Shoulders

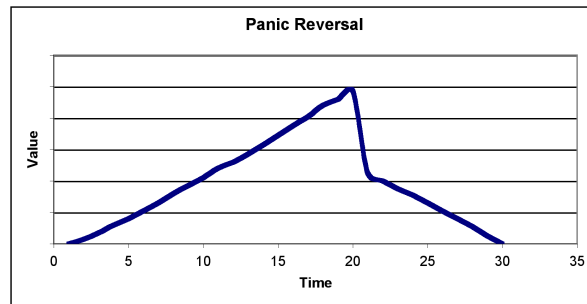


Figure E.10: Panic Reversal

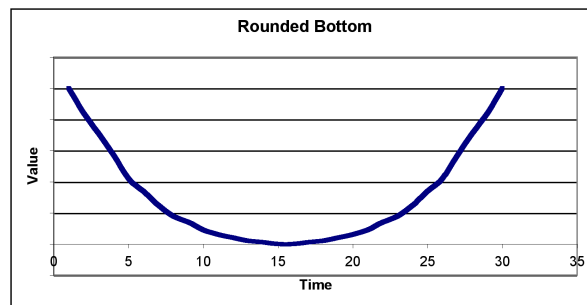


Figure E.11: Rounded Bottom

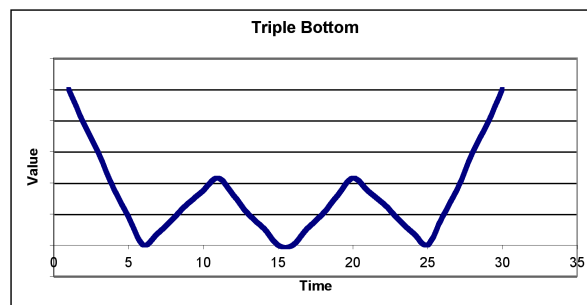


Figure E.12: Triple Bottom

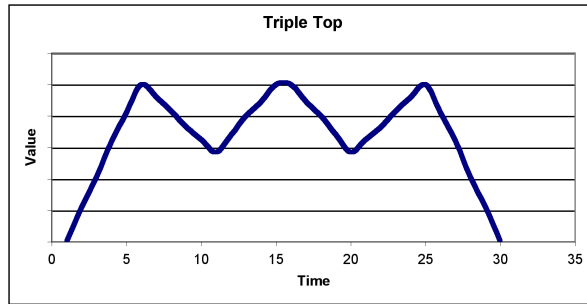


Figure E.13: Triple Top

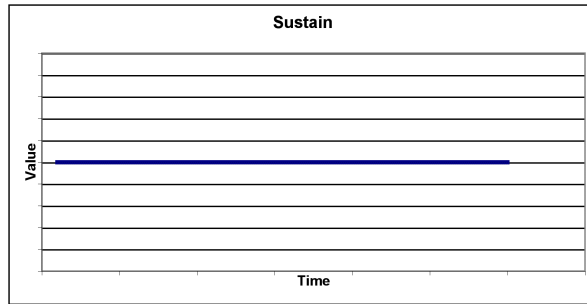


Figure E.14: Sustain

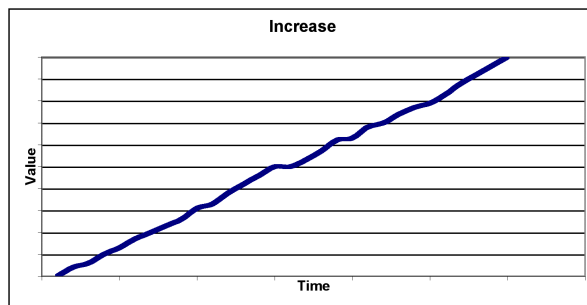


Figure E.15: Increase

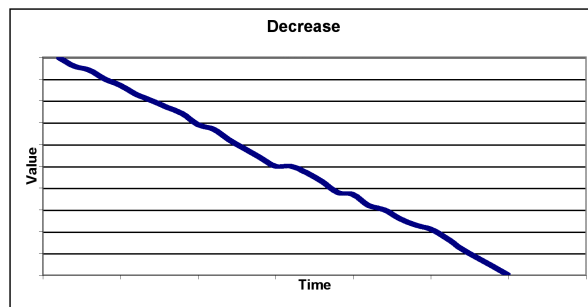


Figure E.16: Decrease

Appendix F

Appendix Sleep Data

Descriptions of the attributes of the data set used for ASAS evaluation over clinical sleep study data are given here.

sleepstageN Data type: Event Set. Specifies when Sleep Stage N occurs during the patient's night's sleep.

sleepstageW Data type: Event Set. Specifies when the patient awakes during the night's sleep.

sleepstage1 Data type: Event Set. Specifies when Sleep Stage 1 occurs during the patient's night's sleep.

sleepstage2 Data type: Event Set. Specifies when Sleep Stage 2 occurs during the patient's night's sleep.

sleepstage3 Data type: Event Set. Specifies when Sleep Stage 3 occurs during the patient's night's sleep.

sleepstage4 Data type: Event Set. Specifies when Sleep Stage 4 occurs during the patient's night's sleep.

sleepstageR Data type: Event Set. Specifies when Sleep Stage REM occurs during the patient's night's sleep.

low-oxy-pot Data type: Event Set. Specifies when patient experiences a low oxygen potential during the night's sleep.

very-low-heart-rate Data type: Event Set. Specifies when patient experiences a low heart rate during the night's sleep.

mild-reduce-heart-rate Data type: Event Set. Specifies when patient experiences a mildly reduced heart rate during the night's sleep.

OSA Data type: Nominal. Possible values: { no, mild, moderately-severe, severe, CPAP-removes-apnea, insignificant, mildtomoderate } Specifies the patient's severity of Obstructive Sleep Apnea.