To appear in Seventh Workshop on Mining Scientific and Engineering Datasets, 24 Apr 2004, Lake Buena Vista, FL

Visual Query Language: Finding patterns in and relationships among time series data

Karen Zita Haigh, Wendy Foslien, Valerie Guralnik
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{karen.haigh,wendy.foslien,valerie.guralnik}@honeywell.com

Abstract

Many scientific datasets archive a large number of variables over time. These timeseries data streams typically track many variables over relatively long periods of time, and therefore are often both wide and deep. In this paper, we describe the Visual Query Language (VQL) [3], a technology for locating time series patterns in historical or real time data. The user interactively specifies a search pattern, VQL finds similar shapes, and returns a ranked list of matches. VQL supports both univariate and multivariate queries, and allows the user to interactively specify the the quality of the match, including temporal warping, amplitude warping, and temporal constraints between features.

1 Introduction

Many scientific datasets archive a large number of variables over time. These timeseries data streams typically track many variables over relatively long periods of time, and therefore are often both wide and deep. As a result, the size of some data sequences is on the order of Gigabytes. Much of this data is irrelevant. Most existing techniques for extracting *patterns of interest* are both time consuming and tedious.

Data sequences have conventionally been analyzed using such techniques as database query languages, or machine learning of pattern labels [6; 8; 15]. Such techniques are frequently limited in the complexity of the pattern that can be described, or fail to incorporate time-based features adequately. Moreover, the lack of an intuitive interface impairs efficiency for many users.

The Visual Query Language (VQL) [3] is a technology for locating time series patterns in historical or real time data. It was initially developed for visually describing signal patterns to find interesting shapes in very large time series data sets. The desired time series “shape” from a set of data is selected interactively by a user, or defined as a template. VQL allows specification of how insensitive the matcher should be to variations in temporal length and variations in feature amplitude. Once this shape is defined, the VQL search engine uses various algorithms to find similar shapes in the historical or real time data stream, and returns a ranked list of matches.

Archived data contains important but often widely scattered information; and archives can be both wide (many-dimensioned) and deep (numerous samples). With VQL, users can define qualitative patterns in the data graphically or select previously defined templates and then expresses the selections as search directives. The VQL search engine uses trend-oriented algorithms to find similar patterns in the data stream and returns a ranked list of matches. Patterns may exist on single variables or multiple variables with temporal constraints on segment positions.

For the purposes of discussion in this paper, our motivating example is a small subset of data collected from the International Space Station’s two Beta Gimbal Assemblies (BGA). The BGAs rotate the two solar panels to generate electricity for the station. Normally, as the angle error between the desired angle and the actual angle increases, the BGA PID controller corrects by increasing motor current (hence increasing the velocity of the angle change). However, sometimes the the velocity does not change, so the current increases until the motor trips. Nominal motor current is 0.2 Amps; ground controllers consider 0.6 Amps to be abnormal; the motor trips at 1.1 Amps.

The data is characterized by 41 BGA 4B parameters, 32 BGA 2B parameters, and 5 parameters external to the two BGAs. We had both nominal behavior (no abnormal events), and data containing abnormal events. The data totalled approximately 18 days, spread over 1.5 years; for a total of 3.2 million lines of data. Most parameters were sampled at 1 Hz, some at 0.1 Hz. There were frequent communication drops between the station and earth, lasting a duration of approx 20 minutes each time.

2 Univariate patterns

VQL is designed to find patterns of interest within timeseries datastreams. Typically, the first question we want to pose to VQL is “where else in the data set did we see a particular behavior in the same variable?” VQL is a template based querying tool, and uses an examplar to locate similar sequences. For a univariate search, we need to define a template feature using historical data, and set search parameters for that feature. Once the template feature is defined, we use one of VQL’s underlying search algorithms to locate a ranked set of matches. In the next sections, we describe the process of creating the template feature.

2.1 Specifying a Univariate Query

The first step the user needs to take is to specify a template for the query. There are three mechanisms to do so. First, the user can click-and-drag over a rectangular region on the datastream; VQL will use the data in that region as a template. Figure 1 shows a feature created in this way. Second, the user can use a *feature editor* to draw a template (or modify an existing one). Third, the user can create a *feature library*, which contains templates of features and can be used across datasets.

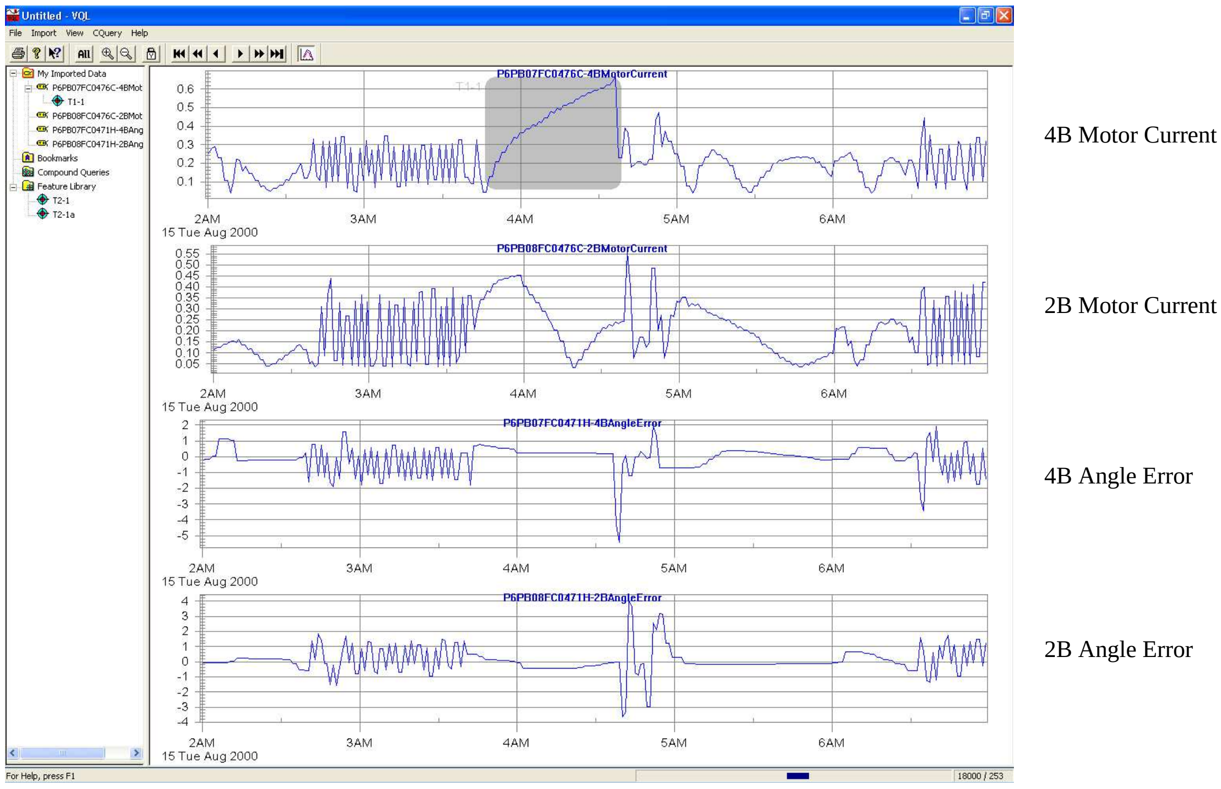


Figure 1: An interesting high-current event in the BGA data.

The user can set a series of parameters on the template, including:

- the *minimum fitness* of returned matches,
- whether to search on *shape* or *energy*,
- the degree to which the *duration* must match (compress and expand),
- the degree to which the *amplitude* must match (grow and shrink),
- a *downsample ratio* (which controls resolution of the search, and trades off accuracy for time),
- expected *periodicity*, and
- the *number of hits* to return

The duration and amplitude constraints are expressed relative to the original feature, e.g. 1.0 would be identical to the feature, while 2.0 is twice as long or twice as intense. VQL will create default values for these parameters; the user can interactively search and then modify parameters as needed.

2.2 Search Algorithm

The application is written in Visual C++ and uses the Microsoft Foundation Classes along with a several Component Object Model (COM) entities. The default search algorithm uses an efficient implementation of a simple moving window correlation calculation. Other search algorithms can be added to VQL by designing additional COM libraries.

The following steps are followed in the default search algorithm. First, a low pass filtered version of the original feature is prepared to eliminate transients that are irrelevant to the feature of interest; the user can set the frequency cutoff (downsample ratio).

Then, a series of time warped versions of the original filter are created according to the input constraints. The shrinking and stretching of the time axis in the template is performed using a variation of uniform scaling [7], rather than dynamic time warping. Prior to the search, a finite set of uniform "warps" of the template feature are generated based on the property settings for the template feature. The user controls the minimum and maximum warp size, as well as the granularity of the warps.

Next, each temporally warped template is swept over the larger target data set. A correlation coefficient is used to compute the degree of match. The coefficient loosely represents the covariance of the two sequences, normalized by their individual standard deviations, effectively implementing the *shape* match. To gain in efficiency, the algorithm uses a circular buffer to incrementally update the mean and variance calculation for the moving window.

The algorithm generates a sequence of correlation coefficients for the target data set. All peaks in this sequence are considered candidate matches. The user specifies limits on the number of peaks to retain; the threshold for the fitness measure (correlation coefficient) for retention of peaks; and thresholds on RMS magnitude of the match. The final list of matches found by the search algorithm is presented to the user sorted by their *fitness*, emphasizing shape (through the correlation coefficient). Note that match duration is included implicitly in this ranking through the shrinking and stretching of the feature, and magnitude limits eliminate matches from the ranked list.

The default search algorithm also includes the capability for periodic searches. In a periodic search, the user can constrain the time of day where searches may be located. This is useful in domains such as commercial building monitoring, where differences in the behavior between in occupied hours and unoccupied hours can be significant. Another capability included in the default algorithm is searching by exception.

There are scenarios where the absence of a match is of interest, rather than the presence. Taking another example from building monitoring, there may be activities that we want to ensure are occurring during unoccupied hours, such as shutting down equipment at specified times. For the periodic searches by exception, VQL tracks the best match inside the specified daily window. At the end of searching, the days that fail to meet the minimal fitness criterion are retained.

In addition, the default search engine can use an *energy* based algorithm, rather than the shape algorithm described above. The energy algorithm uses a fitness measure based on the absolute variance difference between the feature and the search data, using the feature variance as the scaling factor. This contrasts with the windowed correlation approach described above, in that it does not take into account relationships between individual points in the feature and the search data. Some care needs to be exercised with the energy algorithm, because an appropriate value for scaling is very dependent upon variance of the original target relative to the surrounding data. This algorithm is generally used to find periods of flatline data, such as a controller with a saturated output.

2.3 Results

Figure 1 shows the main window in the VQL application. There are two panes in the main window. On the left, a tree view is used to access and manipulate active data, features, and feature matches. The tree is also used to access bookmarks in the data set, multivariate queries (described in Section 3) and feature templates stored in a library.

In Figure 1, we see an example of the BGA data read into VQL. The topmost plot shows a feature of interest that has been selected by the user. In the BGA data, we are generally interested in cases where the motor current rises above about 0.5 Amps. This feature shows a case where the current returned to normal without tripping (at 1.1 Amps).

The user has specified a match threshold, and also defined the tolerance for expansion and shrinkage of the feature in the temporal and amplitude dimensions7. In this example, the repository searched includes approximately 370,000 data points. This search takes about 5 seconds on a 1200 MHz computer using VQL's default search algorithm.

Figure 2 shows the best match to the original template. We can see this match has a similar current rise, as compared

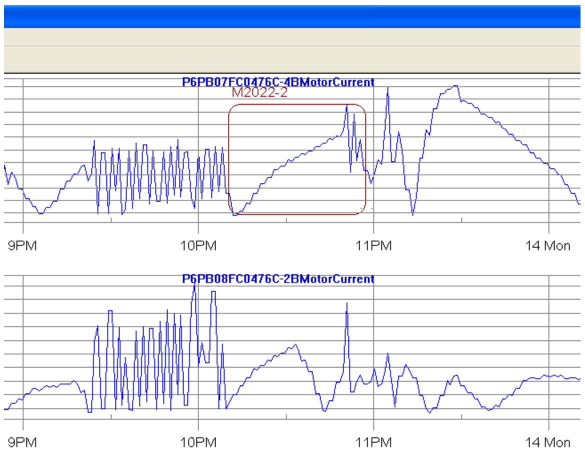


Figure 2: Best match for the original template.

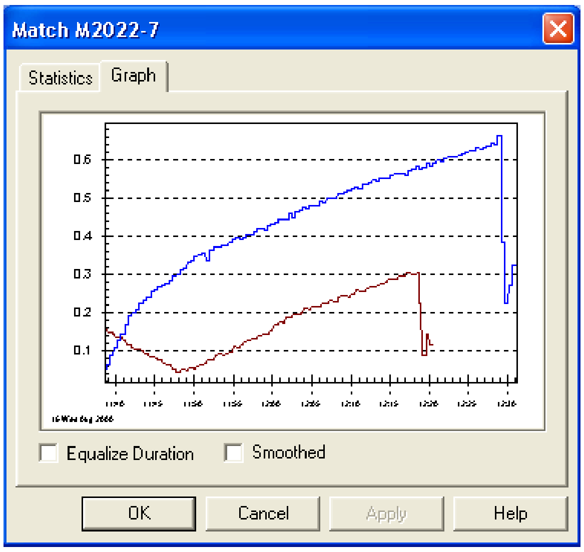


Figure 3: The user can see how well this fair match compares to the original template.

Cylinder	Bell	Funnel
42	29	29

Table 1: Distribution of class instances in the Cyliner-Bell-Funnel Dataset

to the template. This rise is followed by a step decrease in both the template and the match. The durations of the feature and match are also quite similar. The magnitude of the match is slightly smaller, and we can also see some higher frequency oscillation just before the end of the match.

We can also look at a side by side comparison of the feature and match by viewing the properties sheet for the match, as shown in Figure 3 (a fair match). The original feature is shown in the upper curve, the match shown in the lower curve. We see some differences in this feature compared with the original template. Again, we see the characteristic rise and fall of the signal, but the start of the match also includes a linear decrease. The duration of the match is about 80% of the template feature duration, and the overall magnitude is about one half that of the template feature.

Note that we can also view statistics for each match, including the overall fitness measurement, the relative duration and magnitude, as well as the precise time of the match in the historical data.

One of the nice features of VQL is that all the datastreams are tied to the same x axis. That means that when the user zooms in or scrolls on one datastream, all the other datastreams also zoom in or scroll, ensuring that the user sees coordinated events.

Matching Algorithm Effectiveness To test the effectiveness of VQL's default matching algorithm we generated an artificial data set from the *cylinder-bell-funnel task* originally proposed by Saito [12]. The datastream is a randomly-ordered concatenation of these events from the classes cylinder, bell, and funnel, each of which is a univariate time series of fixed length, but with Gaussian noise, and variance on beginning and ends of particular events. The generated data set contained a total of 100 sequences with class distribution shown in Table 1.

	Cylinder	Bell	Funnel
No. of Matches	36	29	30
No. of True Positives	33	28	28
No. of False Positives	3	1	2
No. of False Negatives	9	1	1

Table 2: Match Results for Cylinder-Bell-Funnel Data.

We defined a feature representing each class based on the first sequence of that class in the data set. We then searched the entire data set to find matches. The results are presented in Table 2.

The matching algorithm was able to achieve almost 100% accuracy with very small number of false positives and negatives for classes Bell and Funnel. The false positives for all three classes had a poor fit with the original feature. The large number of false negatives in case of the Cylinder class can be explained by large variability in the width of the tails of the shape. We attempted to redefine a VQL feature representing the Cylinder class by omitting most of the right and left tails of the shape and capturing only the part representative of the class. The search algorithm was able to reduce the number of false positives from 9 down to 3. Unfortunately, the number of false positives also increased from 3 to 8. The false positives had a very good match in the shape throughout the first half of the feature. Note that the correlation filter algorithm does not distinguish the location of the 'good' points in the match. Thus, matching only the first half of the feature has an equivalent fitness to matching half of the shape in the overall feature, for the default algorithm.

Time Performance To demonstrate the efficiency of the matching algorithm, we used aircraft engine timeseries data. Each timeseries consisted of around 13.5 million points sampled at 1Hz rate. We defined several VQL features representative of major events in the timeseries. Table 3 summarizes computation times for finding the matches (if such exist) to each VQL query on a 1.7GHz Pentium M. The time to compute matches increases linearly with the width of the query. The underlying calculation in the default algorithm requires a pass through the length of the feature at each step to calculate the sum needed for the correlation coefficient, and thus we would expect this to scale with the length of the feature.

Width of Query		Computation
(time)	(num samples)	Time
4min 47sec	287	16sec
6min 46sec	406	22sec
6min 57sec	417	22sec
8min 30sec	510	26sec
11min 37sec	697	36sec
15min 39 sec	939	46sec
21min 23sec	1283	63sec

Table 3: Computation Time of VQL queries for 13.5 million engine datapoints.

3 Multivariate Queries

Interesting events may be characterized by patterns that occur on multiple variables. For example, the user might be interested in cases when a rapid increase in temperature is accompanied by a rapid decrease in pressure. VQL supports this kind of multivariate query, along with an intuitive mechanism for specifying it.

3.1 Specifying a multivariate Query

The user specifies a multivariate query by first specifying the (univariate) features of interest, then dropping them on the "Compound Query" folder. As described above, the user can set properties on the match quality of each of the features.

The user can also set constraints on the temporal relationship between the features, denoted a *temporal join*. Each feature has an associated *reference point*, denoted Ref_1 and Ref_2, which is stored as a percentage of the feature width, and used by the search algorithm to determine whether and how well a candidate match meets constraints.

Given a match m to feature f, $t(ref_m)$ is the timestamp associated with Ref_f: for example, if a match m for feature 1 starts at 10 seconds and ends at 20 seconds, and $Ref_1 = 0.2$, then $t(ref_m) = 12$. We refer to δ as $t(Ref_2) - t(Ref_1)$.

The *link* between the two reference points has an associated *error bar*, whose limits are denoted *start* and *end*. Given a match m for feature 1, and a match n for feature 2, they will match the compound query if they meet the constraint:

$$t(ref_m) + start \leq t(ref_n) \leq t(ref_m) + end$$

Figure 4 shows a query in which feature 2 (T2-1) must start between 6 and 24 hours after feature 1 (T1-1) starts; i.e. $Ref_1 = 0.0$ and $Ref_2 = 0.0$, $\delta = 0.502$ days, *start* = 0.25 days, and *end* = 1.00 days. Figure 5 shows a query in which feature 2 may only start after feature 1 is complete, up to 12 hours after, i.e. $Ref_1 = 1.0$ and $Ref_2 = 0.0$, $\delta = 0.0$ hours, *start* = 0.0 hours, and *end* = 12.0 hours. Figure 6 shows a query where the two features *coincide*, that is, both reference points fall within the error bars: $Ref_1 = 0.50$ and $Ref_2 = 0.50$, $\delta = -6.0$ hours, *start* = -12.0 hours, and *end* = 6.0 hours. (Note that VQL displays appropriate temporal units based on the features and their distance.)

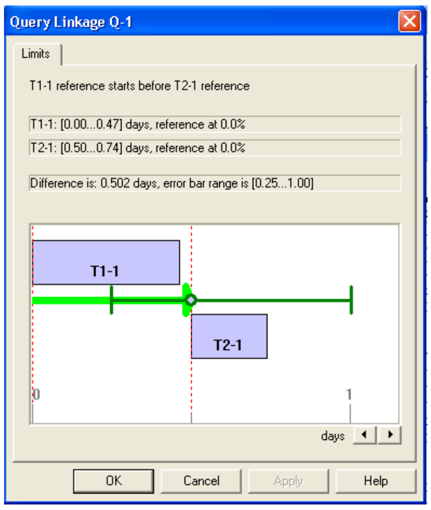
A compound query may be a pair of features, a pair of compound queries, or one of each. Temporal joins over compound queries are specified in the same way as described above. In addition to temporal joins, the user may specify a *logical join* between a pair of compound queries (AND and OR; NOT will be addressed in future work). As an example, let us say that the user has constructed two queries $Q_1 = F_1 \rightarrow F_2$ and $Q_2 = F_2 \rightarrow F_3$. Then, he can construct Q_3 involving Q_1 and Q_2, with either a temporal join or a logical join.

3.2 Search Algorithm

Our search algorithm finds matches in a bottom-up fashion. The algorithm takes the specification of the compound query and the relevant data as inputs. The algorithm finds the univariate matches for each of the features, according to the properties of the feature. Then, the algorithm merges the matches of the individual features according to the join's temporal or logical constraints, pruning out the ones which do not fit join specifations.

To be a match, all of the query's hard logical and temporal constraints must be met. When the results are presented to the user, the set of matches is sorted according to fitness, where a match's fitness is defined by three factors:

- the fitness of match to the first feature, independently: Fit(m)


Query Linkage Q-1

Limits

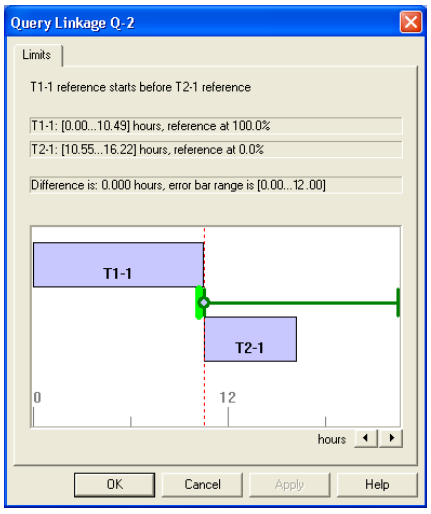
T1-1 reference starts before T2-1 reference

T1-1: [0.00...0.47] days, reference at 0.0%

T2-1: [0.50...0.74] days, reference at 0.0%

Difference is: 0.502 days, error bar range is [0.25...1.00]

Figure 4: Feature 2 starts between 6 and 24 hours after feature 1 starts.


Query Linkage Q-2

Limits

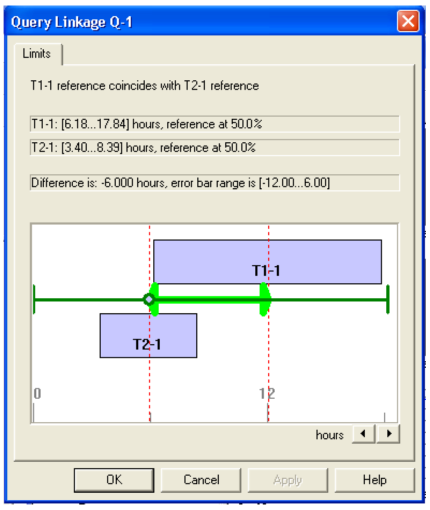
T1-1 reference starts before T2-1 reference

T1-1: [0.00...10.49] hours, reference at 100.0%

T2-1: [10.55...16.22] hours, reference at 0.0%

Difference is: 0.000 hours, error bar range is [0.00...12.00]

Figure 5: Feature 2 must occur only after feature 1 is complete.


Query Linkage Q-1

Limits

T1-1 reference coincides with T2-1 reference

T1-1: [6.18...17.84] hours, reference at 50.0%

T2-1: [3.40...8.39] hours, reference at 50.0%

Difference is: -6.000 hours, error bar range is [-12.00...6.00]

Figure 6: Features 1 and 2 may coincide; either may occur first.

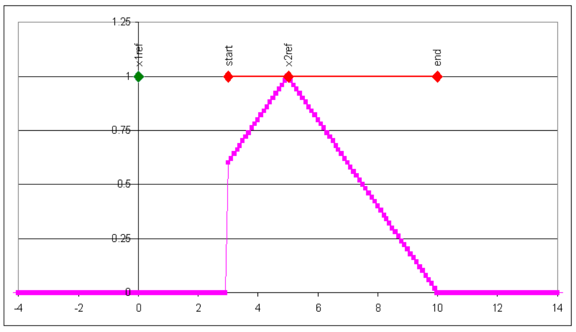


Figure 7: The fitness of a temporal join is linear, based on the widest error bar.

- the fitness of match to the second feature, independently: Fit(n)
- the fitness of the temporal join: Fit(*join*)

The fitness of the compound query is:

$$\text{Fit(CQ)} = \text{Fit}(\textit{join}) \times \sqrt{\text{Fit}(m) \times \text{Fit}(n)}$$

Intuitively, if both m and n are 50% fit and there is no error on the join, then Fit(CQ) is 0.5.

We define fitness of the join to be:

$$\text{Fit}(\textit{join}) = \left[1 - \frac{\text{abs}(t(\textit{ref}_n) - t(\textit{ref}_m) - \delta)}{\max(\textit{end} - \delta, \delta - \textit{start})} \right]$$

where $\delta = t(\textit{Ref}_2) - t(\textit{Ref}_1)$. Figure 7 shows one example (normalized to t($\textit{Ref}_1$) $= 0$), where $t(\textit{Ref}_2) = \delta = 5$, *start* = 3 and *end* = 10. A match whose $t(\textit{ref}_n) - t(\textit{ref}_m)$ is 8 has a fitness of 0.4.

Temporal Join Optimization. To limit the search space we need to employ optimization strategies. The basic strategy is to use a technique from the database community: determining in which order the logical and temporal join operations in compound query should be processed. After the order is determined, each of the joins operations can be optimized independently.

To optimize the search in case of temporal join, we employ a sliding window technique commonly used in other temporal data mining algorithms [11]. First, the matches of each of the two sub-queries are determined and then sorted in increasing order of their reference point.

To find all the matches that meet the constraints of the join for the match of the first sub-query, the algorithm starts by iterating over matches of the second sub-query until it finds a match such that the second reference point is past the *start* error bar position: $t(\textit{ref}_m) + \textit{start} \geq t(\textit{ref}_n)$. The algorithm adds additional candidate matches until the second reference point is past the *end* error bar position: $t(\textit{ref}_m) + \textit{end} < t(\textit{ref}_n)$.

When next match of the left sub-query is considered, the algorithm does not start iteration from the very first match of the second sub-query. Instead, the iteration starts from the first match that had a reference point within the *start* error bar of the previous search.

The algorithm is outlined in Table 4.

```
Let $\mathcal{M}_1$ be the matches for feature 1, sorted by reference time
Let $\mathcal{M}_2$ be the matches for feature 2, sorted by reference time
Let $\mathcal{M}_{CQ}$ be the (NULL) set of compound matches
Let $i = 1$; denote $n_i$ to be the $i^{th}$ element of $\mathcal{M}_2$
Foreach $m \in \mathcal{M}_1$
    While $t(\textit{ref}_m) + \textit{start} < t(\textit{ref}_{ni})$
        $i = i + 1$
    $j = i$; denote $n_j$ to be the $j^{th}$ element of $\mathcal{M}_2$
    While $t(\textit{ref}_m) + \textit{end} \leq t(\textit{ref}_{nj})$
        Add $(m \rightarrow n_j)$ to $\mathcal{M}_{CQ}$
        $j = j + 1$
Return $\mathcal{M}_{CQ}$
```

Table 4: Multivariate search algorithm.

Reusability of (intermediate) results. Since compound queries are constructed from other queries and features, it is reasonable to assume that some of those queries were executed and results of those queries are available. The cached results of those queries can be re-used as needed as long as their search parameters have not been modified. In cases when constraints were tightened, the cached results can also

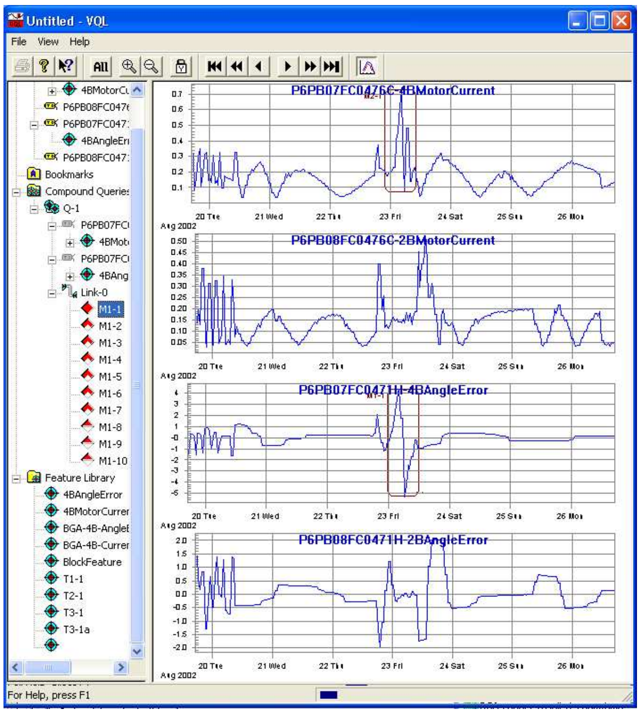


Figure 8: Best match for a multivariate query.

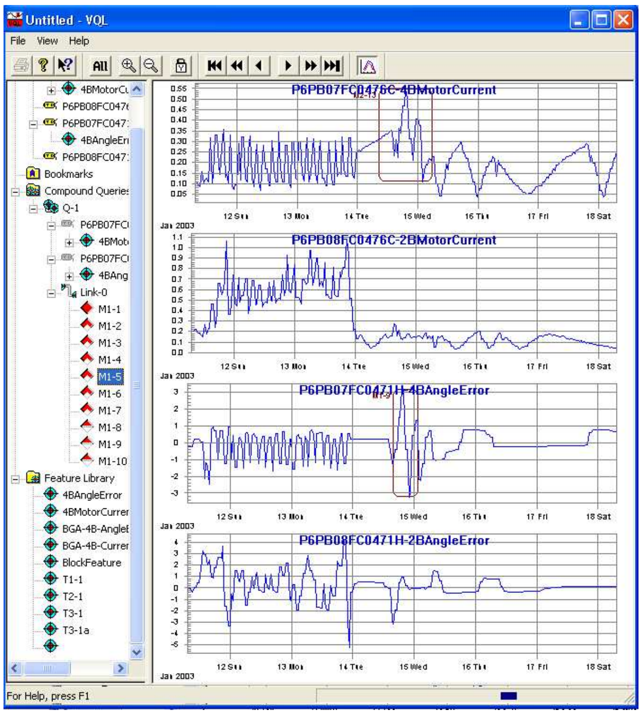


Figure 9: Fair match (#5) for the query.

be reused by pruning the matches that do not fit tighter constraints. The pruning operation is more efficient, because it operates on relatively small number of matches and does not require searching through the whole data set in case of univariate features or performing expensive join operation in case of multivariate (sub-)query.

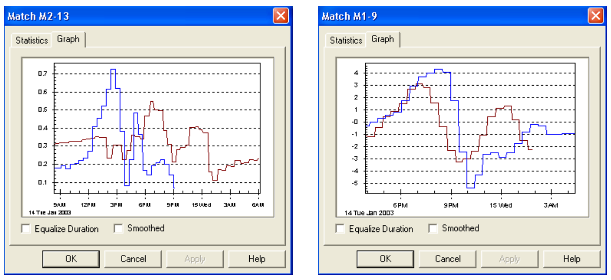
3.3 Results

In Figure 8, we see an example of a multivariate query. We are interested in cases when both the angle error and the motor current oscillate significantly. Searching for this kind of feature would be extremely challenging without the interactive, visual interface. By changing search parameters and altering search constraints, we can quickly find interesting sections of the data.

We set the following margins on the match constaints: matches for individual features had to be at least 30% “good”, with an allowable temporal warp range of 0.8-2.0, and an amplitude range of 0.5-20.0. The temporal join properties were structured to expect the features to coincide: $Ref_1 = 0.5$ and $Ref_2 = 0.5$, $\delta = 0.0$ hours, *start* = -3.0 hours, and *end* = 3.0 hours (each feature lasted approx 12 hours).

Figure 8 shows the best match, and Figure 9 shows a fair match. We can clearly see that the intent of the query has been maintained, even though the exact shape of the features does not match the exact shape of the templates, as shown in Figure 10. In Figure 10a, the match is twice as long as the template, and approximately 80% of the template’s amplitude. In Figure 10b, the match is about 90% of the duration, and approximately 60% of the template’s amplitude.

Note that it would be very difficult to describe these two features in a non-visual manner, especially given considerations like amplitude and duration. In particular, a non-visual technique to describe the templates would not return this match to feature 1, since it has an “extra” local minimum.



(a) Feature 1 (b) Feature 2

Figure 10: Detailed comparison for the two features in compound match #5.

4 Related Work

Timesearcher [9] is similar to VQL in that it is a visually interactive method to specify searches on timeseries data. Unlike VQL, however, timesearcher is interested in *filtering* through many similar time series streams; for example, given a set of 1500 stocks, show the stocks whose value was low on a certain date, and then high on a later date. By drawing a rectangle on the timeseries, the user specifies the search constraints.

There are several systems whose similarily to VQL lies in searching for similar events or patterns in the data. However, rather than using a visual method for describing features and time warping to find matches, they usually use an alphabet or labeling system [4; 6; 8; 10; 15]. The claim is made that dynamic time warping is too expensive; VQL uses the simpler approach of uniform scaling. The warping via uniform scaling also gives the following benefits:

- it is easier for a user to describe complex shapes
- the search can be constrained by amplitude and time warping considerations
- the search is less susceptible to noisy data

Keogh et al [10] use a probabilistic approach to find the best possible match to a univariate query. In their work, a time series is segmented with piece-wise straight lines. Features such as peaks and plateaus are defined using this representation. The query is defined as a sequence of features with permissible time distance between features. The model allows specification of degree of deformation as well as specification of degree of elasticity in time and amplitude. The best possible match to a query is found by scanning the timeseries for matches of individual features and then combining them according to the specified sequence for the best possible match.

Tarzan's [8] uniqueness is in its ability to find *surprising* patterns in timeseries data. The algorithm discritizes the data according to an alphabet, builds suffix trees to model the strings seen in the training data, and then finds the probability of occurrence of a string seen in the test data. A pattern is surprising if the frequency of its occurrence differs substantially from that expected by chance.

Timeweaver [15] predicts events in multivariate *categorical* time series data. A data value on a timestream is an event (i.e. not continuous); a pattern is a sequence of events with ordering constraints. A genetic algorithm searches over the space of patterns, and is shown to be effective if the event of interest is within the prediction window. For example `351:<|TMSP|?|MJ|>*<|?|?|MJ|>*<|?|?|MN|>` means that a major severity alarm occurs on a TMSP device is followed by a major then a minor alarm, all within 351 seconds.

Höppner [5; 6] uses "english" labels (e.g. decreasing, increasing), extracts intervals in time that conform to these labels, and then induces sequences of labeled intervals. Höppner detects multivariate relationships using association rule mining [1] with a sliding window to detect patterns. The relationships between intervals are described through Allen's temporal logic [2], which could also be expressed through multivariate VQL query specification capability. VQL's query specification capability allows a user to tighten or loosen constraints to customize a query to more match user's needs. In Höppner's work the emphasis is on finding all frequent temporal sequences of simple univariate patterns. In our work the emphasis is on finding the best matches of queries cosisting of complex shapes.

Weber et al [14] developed an approach to visualize periodic patterns in time-series data based on spirals. In this approach time-series are mapped to spiral graphs, where the cycle length indicates the periodicity of the data. Color, texture, thickness and/or icons is used to indicate value of time-series data. By changing a cycle length a user is able to visually detect periodic patterns as the apperance will change from unstructured to structured.

Wijk et al [13] presented a visualiation approach that is based on clustering similary daily time-series and present a user with cluster representatives as graphs and the corresponding days on a calendar. Color is used to indicate clusters and corresponding days on a calendar.

5 Conclusion

Using VQL, we were able to identify numerous properties in the BGA data that were previously unknown. For example, our NASA expert had informed us of specific dates and times where known events had occurred. We found numerous additional events, including several in the "nominal" dataset – collected before any events had been noticed by the ground controllers. Using VQL, we gained an understanding of the relationships among parameters, including a good understanding of the internal workings of the PID controller. We were able to specifically find events when the PID controller did not achieve its desired results.

VQL's search algorithm is an powerful way to search through large amounts of data for specific events, especially in a noisy datastream. By adjusting the search parameters, the user can control the flexibility of the search algorithm, and find patterns that match his general intent.

Using a visually-oriented apporach to analyzing data sequences allows the user to specify events of interest intuitively and without needing to know traditional query languages. In addition, the user can confidently avoid reviewing large amounts of data not relevant to the current query, instead viewing regions of interest quickly and easily. VQL is particularly useful in data sequences in which (relatively rare) events of interest are represented by relatively short data sequences contained in large amounts of data.

Please contact the authors if you are interested in obtaining a copy of VQL.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pages 3–14, Taipei, Taiwan, 1995.

[2] J. Allen. Maintaining knowledge about temporal intervals. *ACM Communications*, 26(11):832–843, 1983.

[3] W. Foslien, S. A. Harp, K. Lakshminarayan, and D. Mylaraswamy. Content based retrieval of time series data, 2 July 2002. United States patent pending 09/346,245.

[4] P. Geurts and L. Wehenkel. Early prediction of electric power system blackouts by temporal machine learning. In A. Danyluk, T. Fawcett, and F. Provost, editors, *Proceedings of the ICML98-AAAI98 Workshop on "Predicting the Future: AI Approaches to Time-Series Problems"*, pages 21–28, Madison, WI, July 1998. AAAI Press (Menlo Park, CA). Technical Report WS-98-07.

[5] F. Höppner. Discovery of temporal patterns: Learning rules about the qualitative behaviour of time series. In L. D. Raedt and A. Siebes, editors, *Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases, Lecture Notes in Computer Science 2168*, pages 192–203. Springer-Verlag (Heidelberg, Germany), September 2001.

[6] F. Höppner. Learning dependencies in multivariate time series. In C. Dousson, F. Höppner, and R. Quiniou, editors, *Proceedings of the ECAI'02 Workshop on Knowledge Discovery from Temporal and Spatial Data*, pages 25–31, Lyon, France, July 2002.

[7] E. Keogh. Efficiently finding arbitrarily scaled patterns in massive time series databases. In *Knowledge Discovery in Databases: Pkdd 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 253–265, Cavtat-Dubronik, Croatia, 2003.

[8] E. Keogh, S. Lonardi, and W. Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 550–556, Edmonton, Alberta, Canada, July 2002.

[9] E. J. Keogh, H. Hochheiser, and B. Shneiderman. An augmented visual query mechanism for finding patterns in time series data. In T. A. *et al*, editor, *In the 5th International Conference on Flexible Query Answering Systems, Lecture Notes in Computer Science 2522*, pages 240–250, Copenhagen, Denmark, October 2002. Springer-Verlag (Heidelberg, Germany).

[10] E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In D. Heckerman, H. Mannila, and D. Pregibon, editors, *Proceedings of the Third Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 20–24, Newport Beach, CA, August 1997. AAAI Press (Menlo Park, CA).

[11] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 210 – 215, Montreal, Canada, August 1995.

[12] N. Saito. *Local feature extraction and its application using a library of bases*. PhD thesis, Yale University, 1994.

[13] J. J. van Wijk and E. R. van Selow. Cluster and calendarbased visualization of time series data. In *Proceedings of IEEE Symposium on Information Visualization*, pages 4–9, October 1999.

[14] M. Weber, M. Alexa, and W. Mller. Visualizing time-series on spirals. In *Proceedings of IEEE Symposium on Information Visualization*, pages 7–14, October 2001.

[15] G. M. Weiss and H. Hirsh. Learning to predict rare events in categorical time-series data. In A. Danyluk, T. Fawcett, and F. Provost, editors, *Proceedings of the ICML98-AAAI98 Workshop on "Predicting the Future: AI Approaches to Time-Series Problems"*, pages 83–90, Madison, WI, July 1998. AAAI Press (Menlo Park, CA). Technical Report WS-98-07.