A method of matching a first query pattern with a plurality of stored data is disclosed. For each stored data pattern, the number of components are counted which are identical to corresponding components in the first query pattern, thereby forming a set of match numbers equals the number of components in any stored pattern, that stored data pattern is displayed as an output pattern set indicating a match. If no match exists then a second query pattern is determined by modifying the first query pattern, component by component, in dependence upon both a first, global influence of all stored patterns on all components of the first query pattern and a second, particular influence of all stored patterns on each respective component of the first query pattern. The first two method steps are then repeated using the second query pattern in place of the first query pattern. If no match a third query pattern similarly is determined by modifying the second query pattern. Finally, the output pattern is displayed, component by component, with those respective components of the third query pattern that have been modified at most once from the first query pattern.
FIG. 4

FIG. 5

FIG. 6
DYNAMIC PATTERN MATCHER USING INCOMPLETE DATA

ORIGIN OF THE INVENTION

The invention described herein was made in the performance of work under a NASA contract and is subject to the provisions of Section 305 of the National Aeronautics and Space Act of 1958, Public Law 85-568 (72 Stat. 435; 42 U.S.C. 2457).

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates generally to pattern matching systems, and more particularly to a method for dynamically adapting the system to enhance the effectiveness of a pattern match.

State of the Art

Apparatus and methods for calculating the similarity between patterns are known. For example, U.S. Pat. No. 3,727,183 to LeMay discloses a pattern recognition device using an image recognition algorithm capable of compensating for registration errors. A scanning waveform is used to scan the input image. The scanning waveform is capable of being modified to minimize the degree of error. U.S. Pat. No. 4,446,531 to Tanaka teaches the use of a computer for calculating the similarity between patterns employing a pattern recognition technique using height or “weight” factors as measures of relative importance.

U.S. Pat. No. Re. 26,104 to Glauberman et al discloses data processing apparatus utilizing a pattern recognition method designed for analyzing character symbols.

U.S. Pat. No. 4,319,221 to Sakoe shows a pattern recognition arrangement wherein a single input pattern feature vector is pattern matched with the reference pattern.

There is considerable interest in the storage and retrieval of data, particularly, when the search is called or initiated by incomplete information. For many search algorithms, a query initiating a data search requires exact information, and the data file is searched for an exact match. Inability to find an exact match thus results in a failure of the system or method.

It is therefore desirable to provide a method of storage and retrieval that shares some of the attributes of an artificial neural network (ANN), such as searching for a match using a query having only incomplete information, while avoiding some of the deficiencies such as long “learning” time and possible “retraining” when additional data is stored. In addition, it is desirable to provide several features not available in ANN systems, such as attaching relative importance as well as time dependence to stored data points. Thus, stored data may change in importance over time, whether the time dependent change is caused by the user, by outside input, or simply by a programmed degradation or appreciation over time. It is desirable that the stored data be allowed to change in this way without affecting the speed of retrieval or requiring additional training.

SUMMARY OF THE INVENTION

It is an object of the present invention to find an exact match between a query pattern and one or more stored patterns, if an exact match exists.

It is another object of the present invention to find the “best” match between a query pattern and a stored pattern; i.e., to find one or more stored patterns which are the closest to a match with the query pattern if no exact match exists.

It is an additional object of the present invention to isolate a subset of the stored patterns for which a partial match is possible and to distinguish those portions of the stored patterns in the subset which match the query from those portions which do not match and are therefore ambiguous.

It is a further object of the present invention to allow the user to efficiently enhance the probability of a match by focusing on only the ambiguous portion of the stored and query patterns.

It is a still further object of the present invention to allow the user to efficiently enhance the probability of a match by restating the query with additional data.

It is a yet further object of the present invention to allow the user to efficiently enhance the probability of a match by storing more data.

It is an even further object of the present invention to allow the user to efficiently enhance the probability of a match by modifying one or more of the stored patterns.

The present invention incorporates procedures that seek a response which will be exact, if the query is an exact match to a positive stored data item of a given relative importance, or a “reasonable guess” in view of both the query and the stored data. The invention will return a stored item if the query is within a predetermined variance of a stored item, or the response may contain certain “ambiguous” components indicating that there is a conflict in the stored data that causes an inexact response to the particular query. The particular portions of the pattern which are ambiguous are indicated to the user. The invention, in this event, asks for instructions as to how the user wishes to proceed in effecting a match, as for example, changing the data in the query or changing one or more of the stored patterns.

Many physical entities (photographs, electrocardiograms, voice patterns, seismic signatures, written documents, star patterns, fingerprints, eye fundus patterns, etc.) are capable of being represented by patterns of other physical entities (elements) in some format suitable for electronic systems such as a sequence of digital electronic signals. As is well known, these patterns are capable of being stored in a computer memory to create a library of stored patterns. The present invention makes use of this capability as well as the ability to incorporate within each pattern a relative time-dependent importance property.

Preferably, the query pattern comprising a set of elements representative of the entity to be matched is also created in the same format as the stored patterns. If not, it should be converted to this format prior to pattern matching.

The aforementioned objects and advantages are achieved in accordance with the present invention, by the following method:

The data to be stored and queried, as well as the query itself, is assumed to be in the form of binary lattice points (1,1,0,0, . . . ,0); that is, as points (also called
"patterns") in a finite dimensional space having only zero or one as components. A positive integer n is used to indicate the length of such a point; i.e., the number of components or elements defining the point. The data and query are then stored in the n-dimensional space. For example, if the data points to be stored, as well as the queries, are of the form (0,0,1,0) or (1,0,1,0), then n=four.

Let K denote the number of data points or patterns stored. For each data point Bm, where m is an integer from 1 to K, we have

\[ B^m = (b_{m1}, \ldots, b_{mn}) \]

which is point associated with the time \( t_m \). For each point \( B^m \) we define an integer \( H(m, t_m) \) indicating the relative importance of the point with respect to the other points at the time \( t_m \). The function \( H(m, t_m) \) (which is the coefficient of a positive integer \( P \), to be more fully explained later), may be allowed to increment or decrement as the time parameter \( t \) varies from some initial time. In addition, for the point \( B^m \) there is associated with the relative importance \( H(m, t_m) \) a direction number \( C(m, t_m) \) indicating whether the point should be sought; i.e., whether it is attractive, (a positive one); avoided or repelled (a negative one); or ignored (a zero).

Generally, the method of the present invention involves a comparison of a query pattern with each of the stored patterns, on an element-by-element basis to determine the total number of elements which match, called the "degree of match", for each stored pattern. A "complete match" is said to exist between the query and one or more of the stored patterns if the degree of match for any stored pattern or patterns is equal to (or within some predetermined variation from) the number of elements in the patterns. For some applications, locating a complete match finishes the exercise. In other applications (for example, when \( H(m, t) \) is a much more significant factor that the degree of match) the pattern matching procedure continues. Of course, if \( C(m, t) \) is zero a complete match is not-significant.

If no complete match is found by this first element-by-element comparison, a new or shifted query, called the "derived query," is formulated from the original query. In formulating the derived query, use is made of the closeness of match, or degree of match, between the query of each of the stored patterns determined in the first comparison. In this way, the probability of a match between one or more of the stored patterns and the derived query is increased.

The derived query is then compared on an element-by-element basis with each of the stored patterns in another attempt to find a match. The closeness of match of the derived query with each of the stored patterns is also determined and may be called the "second degree of match".

A complete match between the derived query and one or more of the stored patterns exists and is indicated if the second degree of match for any stored pattern is equal to the number of elements in the patterns. If a complete match is not found with the derived query pattern, a response pattern is created from the original query pattern and other information determined from the comparisons of the original and derived query patterns with the stored patterns. The response or "answer" pattern has as its elements both "determined" elements—i.e., those which are identical with corresponding elements of one (or more) or the set of closest stored patterns—as well as "ambiguous" elements (those which are not identical).

The representative format of the answer pattern may then be converted to a desired (usually the original) format (photograph, etc.) with the ambiguous elements distinguished from the determined elements in some manner (e.g., lighter, darker, as + or −, etc.).

Various manipulations may then be performed to enhance the probability of match, such as refining, enhancing, supplementing, etc., the ambiguous portions of the query and repeating the above steps.

The preferred embodiments of the present invention will now be described with reference to the accompanying drawings.

**BRIEF DESCRIPTION OF THE DRAWINGS**

FIG. 1 is a representational diagram of a two-dimensional figure (n=2) having vertices at all binary points.

FIG. 2 is a representational diagram of the two-dimensional figure of FIG. 1 in which two stored points of prescribed height are placed perpendicular to the two-dimensional figure.

FIG. 3 is a representational diagram identical to that of FIG. 2 in which a smooth surface S is formed over the set of points in three-dimensional space.

FIG. 4 is a representational diagram similar to that of FIG. 3 showing the slope of the surface at a query point thereon in each of the two directions.

FIG. 5 is a representational diagram showing two stored points placed on a two dimensional figure and illustrating a conflict in the stored data.

FIG. 6 is a representational diagram similar to that of FIGS. 1-4 illustrating that the query point is "moved" to a new position.

FIG. 7 is a process diagram showing the procedure for producing a response to an original query.

**DESCRIPTION OF THE PREFERRED EMBODIMENTS**

**Conceptual Illustration**

As an illustration of the present invention, consider an n-dimensional cube with a direction and relative importance of \( C(m, t)P^R(m, t) \) (see FIG. 1 for n=2). The relative importance of each stored point \( B^m \) may be shown by placing a point in the \( n+1 \) dimension space perpendicular to the n-dimensional cube with direction and relative importance of \( C(m, t)P^R(m, t) \). FIG. 2 illustrates the case where \( n=2 \); two points, \( B^1 \) and \( B^2 \), are stored at \((0,1)\) and \((1,1)\) with relative importance magnitudes of \( H(1, 1) \) and \( H(2, 1) \), respectively. The direction numbers \( C(1, 1) \) and \( C(2, 1) \) are both positive ones since relative importance \( H(1, 1) \) and \( H(2, 1) \) both are shown in the positive \( n+1 \) direction.

As an assist in conceptualizing, imagine a smooth surface S formed over the set of points in \( n+1 \) dimensional space. As shown in FIG. 3, this undulating surface rises at stored points with a positive C value, falls toward a zero height over non-stored points, and is negative at points that are to be avoided; i.e., points having a negative C value. This surface may change in time as the relative importance (magnitude of \( H \)) of stored data items (patterns) increment or decrement. It also, of course, changes as the sign of C changes.

For purposes of this invention, significance lies not so much in the surface itself, but, rather, in the slope of the
surface in each of the n directions at a point of interest such as a query point. (See FIG. 4.)

If a query point (pattern) is present, noting also that it is a binary point in the n-dimensional cube, it is desired to determine toward which binary point in the cube it would need to move to effect a match. Assuming that "high" or relatively important points attract, relatively lower points attract less and negative points repulse, the tendency at the query point would be to move accordingly. Ambiguities may exist, as when the query is equally attracted to two or more binary points (or equally repulsed by two or more points).

For example, in FIG. 5, assume that stored points (0,0) and (1,1) have the same relative importance, signified by the same height perpendicular to the n-dimensional cube. A query at point (1,0) would be equally attracted to both points, as would a query at point (0,1), thus a conflict or ambiguity is indicated. On the other hand, a query at point (1,1) would encounter no conflict and would be most strongly attracted to point (1,1)—i.e., no movement—because the slope of the surface S is zero in every direction. Likewise, a query at point (0,0) would have no tendency to move from that point. Of course, if the relative importance of (1,1) and (0,0) were significantly different, then the response to a query such as (1,0) would not be ambiguous, it would move toward the relatively more important point. The invention facilitates dealing with ambiguities as will be explained in detail hereinbelow.

As previously noted, the imaginary surface S is not significant; only its slope at points of interest (query points) is important in the decision process. Further, the magnitude of the slope is not necessarily significant for our purposes, only its algebraic sign. If the sign of the slope in the i'th direction is positive, then the point should "move" in the i'th direction, while if the sign is negative, the query point should "move" in the negative i'th direction. When the slope in the i'th direction is zero, there is no "movement" in the i'th direction.

The method according to the present invention proceeds as follows: The slopes in each of n directions on the surface are determined for the query point and the values stored. A new or "shifted" query point (not necessarily a binary point) is determined from the original query in a manner to be described below, and it is then applied to the n-dimensional cube. The slopes in each of the n directions on the surface at the new position (the second slopes) are determined and the values noted and stored, completing the numerical computations. (See FIG. 6.) It should be noted that no more than two iterations are required to determine the response. The information gained from the computations can be analyzed for a response to the original query.

The sign of the slope in each direction at the original query point is noted; thereafter, the sign of the slope in each direction at the shifted query point is noted.

The response point or answer pattern is formed from the original query point and the slopes of the original and shifted query points as follows:

If the i'th component or element of the query point is 0 and both the first slope and the second slope in the i'th direction are positive, then the i'th element of the response is 1.

If the i'th element of the query point is 1 and both slopes in the i'th direction are negative, the i'th element of the response is 0.

If the i'th element of the query point is 0 and the first slope in the i'th direction is negative, then the i'th element of the response is 0.

If the i'th element of the query point is 1 and the first slope in the i'th direction is positive, then the i'th element of the response is 1.

If the i'th element of the query point is 0, the first slope in the i'th direction is positive and the second slope in the i'th direction is negative, then the i'th element of the response is ambiguous (denoted herein by the symbol *).

If the i'th element of the query point is 1, the first slope in the i'th direction is negative and the second slope in the i'th direction is positive, then the i'th element of the response is ambiguous; i.e., *.

Thus the response is of the form (0,0,1,1, . . . ,1,0,1) for a definitive response or complete match, or the response is of the form (0,0,1,*,1 . . . ,*,0,1) when some of the elements are ambiguous; i.e., the response is partially ambiguous. A complete match is said to exist if the degree of match is equal to the number of elements in the patterns. In the case of a partially ambiguous response, there are several options or combination of options available to the user. One, obviously, is to rephrase the query or use another query. Another is to store more data in the stored patterns, and yet another is to modify some of the stored data.

It should be noted that the stored points, each of which has a time dependent relative importance property, can decrement over time to a near zero value, or become negative or positive or zero, thus allowing a stored point to more strongly attract (or repulse) or ignore. While the relative importance property of a point can be increased or decreased at any time, or allowed to remain constant, none of these changes require any "retraining time". This particular feature is not found in artificial neural systems.

**FIRST EXAMPLE**

The following example of this method was installed and operated in a Symbolic computer using a LISP program as set forth in the attached Appendix. It should be pointed out that both the "zero" and "one" elements of a point were considered to be of equal significance. Using the previously stated concepts, a pattern match was attempted between a query pattern and a library of stored patterns to determine a response or answer pattern using the following algorithm:

1. Suppose that n is the maximum length of a point to be stored.
2. Each of B1, B2, . . . , Bn is a data point to be stored of relative importance H(1,t1), . . . , H(k,tk) and sign C(l,t1), . . . , C(k,tk) respectively.
3. Let P denote a positive prime number (for mod P calculations) and each of L and L' an integer (L > L' > 1).
4. Let X = (x1, . . . , xn) be a query point.
5. We first compute and store, for each integer i from 1 to n, the value of the D1i (see below), which is proportional to the slope in the i'th direction at the query point X = (x1, . . . , xn),

   \[ D_{1i}(X; B_1, . . . , B_n) = C(l, i) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{1j}(X) - \text{mod}(P, [a(X); X]) S_{1j}(X) \]

   \[ + C(k, i) \text{mod}(P, [a(X); X]) S_{kj}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{kj}(X) \]

   \[ = C(l, i) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{1j}(X) - \text{mod}(P, [a(X); X]) S_{1j}(X) \]

   \[ + C(k, i) \text{mod}(P, [a(X); X]) S_{kj}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{kj}(X) \]

   \[ = C(l, i) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{1j}(X) - \text{mod}(P, [a(X); X]) S_{1j}(X) \]

   \[ + C(k, i) \text{mod}(P, [a(X); X]) S_{kj}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{kj}(X) \]

   \[ = C(l, i) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \text{mod}(P, [a(X); X]) S_{1j}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{1j}(X) - \text{mod}(P, [a(X); X]) S_{1j}(X) \]

   \[ + C(k, i) \text{mod}(P, [a(X); X]) S_{kj}(X) + \ldots + \text{mod}(P, [a(X); X]) S_{kj}(X) \]
where

(1) \( C(m, t) \) is the sign (or 0) of the stored point \( B^m \) at time \( t \) (if 0, \( C \) is then ignored in computations),
(2) \( H(m, t) \) is the height of the stored point \( B^m \) at time \( t \),
(3) \( J(X, j) \) is the number of components of \( B \) that agree with the corresponding component of \( X \), i.e., a measure of closeness and
(4) \( S_j(X,j) \) is \(-1\) if the \( j^{th} \) component of \( B \) agrees with the \( j^{th} \) component of \( X \) and is 0;
1 if the \( j^{th} \) component of \( B \) agrees with the \( j^{th} \) component of \( X \) and is 1;
-P \( j \) if the \( j^{th} \) component of \( X \) is 1 and the \( j^{th} \) component of \( B \) is 0; and
P \( j \) if the \( j^{th} \) component of \( X \) is 0 and the \( j^{th} \) component of \( B \) is 1.

At this point the response to the original query is made in the following manner (see FIG. 7):

(8) The value of the slope in each direction at the original query point \( X \) has been stored and the value of the slope in each direction at the shifted point \( X' \) has been stored.
If the \( j^{th} \) component of the query point is 0 and both the slope of the query point \( X \) and the slope of the shifted point \( X' \) in the \( j^{th} \) direction is positive, then the \( j^{th} \) component of the response is 1.
If the \( j^{th} \) component of the query point is 1 and both the slope of the query point \( X \) and the slope of the shifted point \( X' \) in the \( j^{th} \) direction is negative, then the \( j^{th} \) component of the response is 0.

In the case described in FIGS. 2, 3 and 4, the dimension is two \((n = 2)\) \( P = 2 \), \( L = 5 \) and \( L' = 3 \) and for simplicity we set \( t = t_1 = t_2 = 0 \). The points \( B_1 = (0,1) \) and \( B_2 = (1,1) \) are stored at relative importance \( H(1,0) = 7 \) and \( H(2,0) = 5 \) with \( C(1,0) = 1 \) and \( C(2,0) = 1 \). The query point is \( X = (0,0) \).

For this case:

\[
J(X, 1) = 1; \quad J(X, 2) = 0; \quad S_1(X, 1) = -1; \quad S_2(X, 2) = 0
\]

where

(1) \( C(m, t) \) is the sign (or 0) of the stored point \( B^m \) at time \( t \);
(2) \( H(m, t) \) is the height of the stored point \( B^m \) at time \( t \); and
(3) \( J(X, j) \) is the number of components of \( B \) that agree with the corresponding component of \( X \); and
(4) \( S_j(X,j) \) is \(-1\) if the \( j^{th} \) component of \( B \) agrees with the \( j^{th} \) component of \( X \) and is 0;
1 if the \( j^{th} \) component of \( B \) agrees with the \( j^{th} \) component of \( X \) and is 1;
-P \( j \) if the \( j^{th} \) component of \( X \) is 1 and the \( j^{th} \) component of \( B \) is 0; and
P \( j \) if the \( j^{th} \) component of \( X \) is 0 and the \( j^{th} \) component of \( B \) is 1.
The algorithms indicated for the method according to the preferred embodiment of the present invention are intended to be implemented on a multiprocessor machine as there is a considerable amount of parallelism.

The pattern matching method according to the preferred embodiment of the present invention is summarized in the following Table:

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Step Name</th>
<th>Operation</th>
<th>Subscripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Store Patterns</td>
<td>$B = b_{ij}$</td>
<td>$i = 1, \ldots, n$</td>
</tr>
<tr>
<td>(2)</td>
<td>Define Query</td>
<td>$X = x_i$</td>
<td>$j = 1, \ldots, k$</td>
</tr>
<tr>
<td>(3)</td>
<td>Match Set</td>
<td>$M = m_{ij}$</td>
<td>$j = 1, \ldots, k$</td>
</tr>
<tr>
<td>(4)</td>
<td>Initial Output</td>
<td>$O = o_{ij} = x_{ij}$</td>
<td>$j = 1, \ldots, n$</td>
</tr>
<tr>
<td>(5)</td>
<td>Disturbance</td>
<td>$LM = Lm_i$</td>
<td>$j = 1, \ldots, k$</td>
</tr>
<tr>
<td>(5A)</td>
<td>Disturbance with Height</td>
<td>$LM + H(i) = Lm_i + b_p$</td>
<td>$j = 1, \ldots, k$</td>
</tr>
<tr>
<td>(6)</td>
<td>Sign</td>
<td>$sgn = sgn_{ij} = 2b_{ij} - 1$</td>
<td>$i = 1, \ldots, n$</td>
</tr>
<tr>
<td>(6A)</td>
<td>Sign with Direction</td>
<td>$c(t) = sgn = c_j sgn_{ij} = C_j (2 b_{ij} - 1)$</td>
<td>$j = 1, \ldots, n$</td>
</tr>
<tr>
<td>(7)</td>
<td>Magnitude</td>
<td>$Mag = mag_{ij} = [(\log_{10}(b_{ij} x_i) = 1)]$</td>
<td>$j = 1, \ldots, k$</td>
</tr>
<tr>
<td>(8)</td>
<td>Exponent</td>
<td>$Exp = exp_{ij} = Lm_j + mag_{ij}$</td>
<td>$j = 1, \ldots, k$</td>
</tr>
<tr>
<td>(9)</td>
<td>Positive/ Negative Sums</td>
<td>For each $i = 1, \ldots, n$ form two sums: $Pos = \sum_{j=1}^{k} P(exp_{ij}) + \ldots + P(exp_{ij})$ for each $exp_{ij}$ of $j$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>where $sgn_{ij} &gt; 0$.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neg. $\sum = \sum_{j=1}^{k} P(exp_{ij}) + \ldots + P(exp_{ij})$ for each $exp_{ij}$ of $j$ where $sgn_{ij} &lt; 0$.</td>
<td></td>
</tr>
<tr>
<td>(10)</td>
<td>Second Query</td>
<td>Form intermediate set $D = d_j$</td>
<td>$j = 1, \ldots, n$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If $Pos = 0$, Neg. sum, then $d_j = 1$;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Else $d_j = 0$.</td>
<td></td>
</tr>
<tr>
<td>(11)</td>
<td>Intermediate Response</td>
<td>$R = r_j$</td>
<td>$i = 1, \ldots, n$</td>
</tr>
<tr>
<td>(12)</td>
<td>Initialize</td>
<td>$X = X_i = x_{ij}$</td>
<td>$i = 1, \ldots, n$</td>
</tr>
<tr>
<td>(13)</td>
<td>Recompute</td>
<td>$X = X_i = x_{ij}$</td>
<td>$X = X_i = x_{ij}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$L = L_i$; where $L_i$ is the number of stored patterns. As may be understood from this table, the method comprises the following essential steps:</td>
<td></td>
</tr>
<tr>
<td>(14)</td>
<td>Third Query</td>
<td>For those components that had “set flag”;</td>
<td>$i = 1, \ldots, n$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if $d_i = x_i$, then $x_i^* = x_i$; and</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if $d_i = x_i$, then $x_i^* = x_i^*$.</td>
<td></td>
</tr>
<tr>
<td>(15)</td>
<td>Output</td>
<td>$O = X^* = x_i^*$</td>
<td>$i = 1, \ldots, n$</td>
</tr>
</tbody>
</table>

In the pattern matching method set forth in this table, a first query pattern, taking the form of a set $X = x_i$, is matched with a plurality of stored data patterns, taking the form of a matrix $B = b_{ij}$, where $i = 1, \ldots, n$ is the number of components in each pattern and $j = 1, \ldots, k$ is the number of stored patterns. As may be understood from this table, the method comprises the following essential steps:

(a) For each stored data pattern, the number of components which are identical to corresponding components in said first query pattern are counted, thereby forming a set of match numbers $M = m_{ij}$ (steps (1)–(3) in the Table).

(b) If any match number $m_{ij} = n$, then the respective $j$th stored data pattern is displayed as an output pattern set $O = o_{ij}$ indicating a match (step (4) in the Table).

(c) If no match number $m_{ij} = n$, then a second query pattern $X' = x_i$ is determined by modifying the first query pattern, component by component, in dependence upon both a first, global influence of all stored patterns on all components of the first query pattern and a second, particular influence of all stored patterns on each respective component of said first query pattern (steps (5)–(12) in the Table).

(d) Steps (a) and (b) are then repeated using the second query pattern in place of the first query pattern.

(e) If no match number $m_{ij} = n$, a third query pattern $X'' = x_i''$ is determined by modifying the second query pattern, component by component, in dependence upon both a third, global influence of all stored patterns on all components of the second query pattern and a fourth, particular influence of all stored patterns on each respective component of the second query pattern, with the third and fourth influences being less than the first and second influences, respectively (steps (13) and (14) in the Table).

(f) The output pattern $O = o_{ij}$ is then displayed, component by component, with those respective components of the third query pattern that have been modified at most once from the first query pattern (step (15) in the Table).
11

If desired, those respective components that have been modified twice from the first query pattern may be displayed in the output pattern O=x; in such a manner as to indicate conflict between the first query pattern and the set of all stored data patterns. For example, these components that have been modified twice may be displayed as an asterisk (*).

The step (c) indicated above preferably comprises the steps of:

1. multiplying each match number m of a match set M by a first disturbance factor L to produce a set LM;
2. determining a sign matrix \( Sgn \) for all components \( sgn_{ij} \) by setting each component equal to \(-1\) if the corresponding stored pattern component \( b_{ij} \) is 0, and to \(+1\) if the stored pattern component is 1;
3. determining a magnitude matrix \( Mag \) for all components according to the formula \( mag_{ij} = [\log(xor (b_{ij}, x_i)) = 1] \times N \times i \), where \( N \) is a magnification factor;
4. determining an exponent matrix \( Exp = LM + Mag \) for all components according to the formula \( ex_{ij} = \log(N) + mag_{ij} \);
5. forming the positive and negative sums for all components as follows:
   - \( Pos.\sum_i = P(ex_{i,j}) + \ldots + P(ex_{i,k}) \) for each \( ex_{i,j} \) of \( j \) where \( sgn_{ij} > 0 \),
   - \( Neg.\sum_i = P(ex_{i,j}) + \ldots + P(ex_{i,k}) \) for each \( ex_{i,j} \) of \( j \) where \( sgn_{ij} < 0 \)
6. determining a second query pattern set \( X' = x' / \) for all \( i \) as follows:
   - if \( Pos.\sum_i = Neg.\sum_i, \) then \( x'_i = 1 \),
   - else \( x'_i = 0 \).

Thereafter, a response set \( R = r_i \) is determined for all \( x'_i \) as:
   - \( r_i = x_i \) if \( x'_i = x_i \), and
   - \( r_i = \text{set flag, if } x'_i = x_i \).

Similarly, step (e) indicated above preferably includes the steps (1) through (6) using a second disturbance factor \( L' \) which is less than the first disturbance factor \( L \) for those components \( i \) that have a set flag.

According to a preferred embodiment of the present invention, the height factor set \( H = h_i \) is associated with the components of each stored pattern \( b_{ij} \). Each height factor component \( h_{ij} \) is indicative of the relative importance of each stored pattern with respect to the other stored patterns. This height factor set \( H \) is added to the set \( LM \) for purposes of determining the matrix \( Exp \).

The height factor components \( h_{ij} \) may be a function of time and are also preferably an integer, such as \(-1, 0, 1, 2, \) etc.

According to a further preferred embodiment of the present invention a sign factor set \( C = c_j \) is associated with the components of each stored pattern \( b_{ij} \). Each sign factor component \( c_j \) is indicative of whether the pattern component is to be sought, avoided or ignored. The sign factor set \( C \) is multiplied by the sign matrix \( Sgn \) for the purposes of determining the positive and negative sums.

Some or all of the sign factor components \( c_j \) may be a function of time. These sign factor components \( c_j \) are preferably the integer values \(+1, 0\) and \(-1\).

The maximum range for the first disturbance factor \( L \) is approximately 2 to 20; the best range for this factor \( L \) is 3 to 7. For example, the first disturbance factor \( L \) may be chosen to be 3 while the second disturbance factor \( L' \) is chosen to be 2.

The maximum range for the magnification factor \( N \) is 2 to 20 with the best range of values 3 to 7. For example, the magnification factor may be chosen as 3.

In conclusion, the method of matching a first query pattern, represented by the set \( X \), with a plurality of stored data patterns, represented by the matrix \( B \), is accomplished by approximating the surface in vector space defined by the stored patterns by a Bernstein Polynomial. This approximation makes it possible to compute the derivative of the surface in all directions at the query point. It is thus possible to determine a new query point by proceeding in the direction of positive slope (derivative) and in the opposite direction of a negative slope (derivative).

A preferred embodiment and best mode of a LIST computer program which implements the present invention is set forth in the attached Appendix.

There has thus been shown and described a novel dynamic pattern matcher which fulfills all the objects and advantages sought therefor. Many changes, modifications, variations and other uses and applications of the subject invention will, however, become apparent to those skilled in the art after considering this specification and the accompanying drawings which disclose the preferred embodiments thereof. All such changes, modifications, variations and other uses and applications which do not depart from the spirit and scope of the invention are deemed to be covered by the invention, which is to be limited only by the claims which follow.

APPENDIX
(defmethod (box-mouse-sensitive-items-mixin :after :init) (ignore)
  (setf item-blinker
    (tv:make-blinker self 'tv:rollover-rectangular-blinker :visibility nil)))

(defmethod (box-mouse-sensitive-items-mixin :find-item) (rc-list)
  (loop for item in item-list
    when (and (member item-name item) rc-list :test #'equal)
    do (list item)))

(defmethod (box-mouse-sensitive-items-mixin :make-item) (name left top right bottom)
  (let ((item (make-item :left left :top top :right right :bottom bottom :name name)))
    (push item item-list)))

(defun mouse-sensitive-items-print-content)
  (loop for item in item-list
    when (and (member item-name item) rc-list :test #'equal)
    do (list item)))

(defun mouse-sensitive-items-remove-item (item)
  (setf item-list (delete item item-list))
  (tv:mouse-wakeup))

(defun mouse-sensitive-items-remove-all ()
  (setf item-list nil)
  (tv:mouse-wakeup))

(defun mouse-sensitive-items-reset-state ()
  (loop for item in item-list
    do (setf (item-state item) 0)))

; Note this function uses the current item-list order and to generate a linear bit vector
; item-list order cannot be altered!!
(defun mouse-sensitive-items-make-state-array ()
  (let* ((length item-list)
    (state-array (make-array '(* 1 (list item-name) 1)
      :element-type 'tv:element))
    (loop for item in item-list
      for index from 0
tenail
      do (setf (state-array index) (item-state item)))))

(defun foo (row-index col-index lat dir)
  (list row-index col-index lat dir)
  (loop for c from 0 to max-row-index
    collect
      (loop for r from 0 to max-col-index
        collect
          (case dir
            (t (row nth (+ (* c no-rows) r) lat)
          (t (col nth (+ (* c no-cols) r) lat))))))

(defun test (no-rows no-cols lat dir)
  (loop for c from 0 below no-rows
    collect
      (loop for r from 0 below no-cols
        collect
          (case dir
            (t (row nth (+ (* c no-rows) r) lat)
          (t (col nth (+ (* c no-cols) r) lat))))))

(defun bar (lat)
  (let (i) (length (car lat)))
  (loop for i in lat)
(defun nconc (loop for k from 0 below 11 by 2
collect (let ((counter 0))
(if (odotimes c 3 (if (=& counter 2) t nil))
  (and (= (nth (+ c k) 1) 1) (incf counter))
  1)
  0))))

(defvar xx (item-list no-rows no-cols optional (type :row))
(let (nil (reverse item-list))
  (case type
    (row (setq outer-loop-count no-rows)
      (setq inner-loop-count no-cols)
      (loop for r from 0 below outer-loop-count
collect (loop for i from 0 below inner-loop-count
collect (nth (case type
                  (col (= (+ r outer-loop-count) c))
                  (row (= (+ c outer-loop-count) r)))) nil))
    (col (item-state (nth (+ c outer-loop-count) r)) nil))))

(defvar-method get-list box-mouse-sensitive-items-mixin (no-rows no-cols optional (type :row))
(let (nil (reverse item-list))
  (case type
    (row (setq outer-loop-count no-rows)
      (setq inner-loop-count no-cols)
      (loop for c from 0 below outer-loop-count
collect (loop for r from 0 below inner-loop-count
collect (item-state (nth (case type
                                (col (= (+ r outer-loop-count) c))
                                (row (= (+ c outer-loop-count) r)))) nil))))
    (col (item-state (nth (+ c outer-loop-count) r)) nil))))

(defvar-method compress-rule box-mouse-sensitive-items-mixin (list)
(let (nil (length (car list)))
  (loop for i in list
collect (loop for k from 0 below (1- 11) by 2
collect (let ((counter 0))
                  (if (odotimes c 3 (if (=& counter 2) t nil))
                    (and (= (nth (+ c k) 1) 1) (incf counter))
                    1)
                    0))))

(defvar make-col-list (list no-rows no-cols)
  (loop for c from 0 below no-rows
collect (loop for r from 0 below no-cols
collect (nth (+ c (* r no-rows)) list)))))

(defvar make-rec-list (list no-rows no-cols key (type :col))
(let (outer-loop-count inner-loop-count)
  (case type
    (row (setq outer-loop-count no-rows)
      (setq inner-loop-count no-cols)
      (loop for r from 0 below outer-loop-count
collect (loop for i from 0 below inner-loop-count
collect (nth (case type
                      (col (= (+ r outer-loop-count) c))
                      (row (= (+ c outer-loop-count) r)))) list)))))

(defvar-method (box-mouse-sensitive-items-mixin :compress) (type)
(let* ((size (item-size (car item-list)))
   (no-rows (= size car size))
   (no-cols (= size cdr size))
   (compress-rows (= (floor (/ no-rows 3)))
   (compress-cols (= (floor (/ no-cols 3))))
   (compress-rec-vector (make-array (* compress-rows compress-cols)
      \(element-type \'(unsigned-byte 8))
   (compress-list outer-loop-count inner-loop-count)))
  (declare (special compress-rec-vector))
  (print (make-rec-list
   (compress-rule (get-list no-rows no-cols :col)
   compress-rows no-cols type :row))
   (compress-list outer-loop-count inner-loop-count)))
(setq compress-list nil)
(if (eq type 'col)
  (print (compress-rule (make-re-list
    (compress-rule (get-list no-rows no-cols :rew))
    no-rows compress-cols (type 'col))
  ;; the following code is wrong
  (print (compress-rule (make-re-list
    (compress-rule (get-list no-rows no-cols :rew))
    compress-rews no-cols (type 'rew))))
  (case type
    (row (setq outer-loop-count compress-rews)
      (setq inner-loop-count compress-cols)
      (loop for c from 0 below outer-loop-count
do (loop for r from 0 below inner-loop-count
do (send (bit compress-cc-vector (case type
          (:row (secq outer-loop-count compress-rews)
            (:eo1 outer-loop-count compress-rews)
            (:rev (* r (* c outer-loop-count) 1))
        (loop for c from 0 below outer-loop-count
do (loop for r from 0 below inner-loop-count
do (ret2 (bit (send (case type
          (:row (secq outer-loop-count compress-rews)
            (:eo1 outer-loop-count compress-rews)
            (:rev (* r (* c outer-loop-count) 1))
        (loop for c from 0 below outer-loop-count
do (loop for r from 0 below inner-loop-count
do (send :remove-all)
      (loop for c from 0 below col
do (send self :make-item (cons r c) left-top (+ left pad-width) (+ top pad-height)))
  (defmethod (sensitive-pad-mixin :draw-grid) (optional (make-sensitive-areas t))
(erase-content t)  
(aux pad-width pad-height grid-width grid-height)  
(when erase-content  
(send self expose)  
(send self (clear-window))  
(multiple-value-bind (width height) (send self (inside-size))  
(setq pad-width (floor (/ width col)))  
(setq pad-height (floor (/ height row)))  
(setq grid-width (+ col pad-width))  
(setq grid-height (* row pad-height)))  
(loop for c from 0 to row  
for y from 0 by pad-height  
do  
(if (<= y height)  
(decf y l))  
(setq pad-height (+ pad-height y tv:alu-sets))  
(loop for r from 0 to col  
for x from 0 by pad-width  
do  
(if (<= x width)  
(setq grid-width (+ grid-width x tv:alu-sets))  
(if make-sensitive-areas  
(send self (make-sensitive-area pad-width pad-height))))  
(defun draw-pattern (row col pattern-array window .optional (size nil) (from-x 0) (from-y 0)  
aux box-width box-height alu (draw-half nil) (dim -1)  
(p pattern-array))  
(declare (ys:array-register p))  
(multiple-value-bind (width height) (if (null size) (send window (inside-size))  
(appy 'values size))  
(setq box-width (floor (/ width col)))  
(setq box-height (floor (/ height row)))  
(loop for c from 1 to row  
for top from from-y by box-height  
do  
(loop for r from 1 to col  
for left from from-x by box-height  
do  
(case (bit p (incf dim))  
((1 t) (setq alu (tv:alu-sets))  
(10 nil) (setq alu (tv:alu-andcons))  
(* (setq draw-half t))  
(if (not draw-half)  
(send window :draw-rectangle box-width box-height (1+ left) (1+ top) alu)  
(send window :draw-rectangle  
(floor (/ box-width 2))  
(floor (/ box-height 2))  
(left (floor (/ box-width 4)))  
(top (floor (/ box-height 4)))  
(tv:alu-sets)  
(setq draw-half nil))))))  
(setq top (tv:window 'tv:window :edges from mouse :blinker-p nil))  
(defflavor stored-patterns-pane (input-pattern-stream state-list qarys-state-list height-list)  
(box mouse-sensitive-items min sensitive-pad-min  
tv:graphics-mixin t'vpane-mixin tv:window)  
:stateable :instance :variables)  
(defmethod (stored-patterns-pane :who-line-documentation-string) ()  
(nil string "?L:Delete, Ctl-L:Modify"))  
(defmethod (stored-patterns-pane :divide-storage-boxes) (row col)  
(send self (set-row row)  
(send self (set-col col))  
(send self (set-draw-grid t)))  
(defmethod (stored-patterns-pane :update-state-and-height-list) ()  
(setq state-list nil)  
(setq qarys-state-list nil)  
(setq height-list nil)  
(loop for l in item-list  
for state = (item-state l)  
for h = (item-name l)  
do  
when (typep state 'array)  
(setq state-list (cons state state-list))  
(setq height-list (cons h height-list)))  
(setq qarys-state-list (mapcar 'convert-to-one state-list)))  
(defun-method edit-pattern stored-patterns-pane (item state window)  
(let (item-array item-state)  
(item-items (reverse (send window (item-list)))))  
(declare (ys:array-register item-array))  
(send window (reset-state)  
(send window (clear-window))  
(send window (draw-grid nil))  
(loop for i in item-items  
for index from 0  
do  
(if (= bit item-array index)  
(send window (update-item i :draw))))
(defmethod (score-change-patterns :after refresh) (ip)
  (send Bdf :draw-grid nil)
)

(defun revisions-flavor (store-patterns-array-and-height-list)
  (setf store-patterns-array (loop for p-pair in store-patterns-array-and-height-list
                                 collect (cons p-pair (draw-pattern p-pair)))))

(defun input-pattern-pane (collect-update-boxes (col))
  (let ((current-row-col-index (collect-update-boxes current-row-col-index)))
    (update-one-item-item-type))
)

(defun input-pattern-pane (store-pattern)()
  (let ((sp (send self :store-patterns-stream))
        (p-col (send self :col))
        (pattern-array (send self :make-item-state-array))
        (stored-pad-item left-top)
        (setf stored-pad-item (loop for i in (reverse (send sp :item-list))
                                   thereis (and (numberp (item-state i)))))
        (setf left (item-left stored-pad-item))
        (setf top (item-top stored-pad-item))
        (setf item-name stored-pad-item-pattern-array)
        (setf (item-name stored-pad-item-pattern-array) 3)
        (default-height :draw-pattern p-col p-col-pattern-array sp
                        (list (- (item-right stored-pad-item) left)
                              (- (item-bottom stored-pad-item) top))
                              left top)
        (send sp :draw-grid nil nil))
)

(defun input-pattern-pane (after store-pattern)()
  (let ((sp (send self :store-patterns-stream))
        (send sp :update-state-and-height-list)))
)

(defun input-pattern-pane (mouse-click) (button x y))
(let ((item (send self :mouse-sensitive-item x y)))
  (update-one-item item :steppe) t)
  ((eq button #(mouse-L-2)
    (when (and row col) 
    (send self :draw-grid nil) 
    (send self :reset-state) t))
  ((eq button #(mouse-L-1)
    (send self :store-pattern) t))
  ((eq button #(mouse-L-1)
    ;:: (process-run-function "Run Berstein"
    ;:: (lambda ()
      (let ((sp (send self :store-patterns-stream))
        (ep (send (send self :superior) :get-pane 'output-pattern)) 
        (lisp (send (send self :superior) :get-pane 'lisp))
        (send lisp (clear-window))
        (send ep :clear-window)
        (draw-pattern row col)
        (top-level (send self :make-item-state-array)
          (send sp :state-list) (send sp :height-list) lisp) ep)))
  ((eq button #(super-mouse-L-1)
    ;:: (process-run-function "Run Berstein"
    ;:: (lambda ()
      (let ((sp (send self :store-patterns-stream))
        (ep (send (send self :superior) :get-pane 'output-pattern))
        (lisp (send (send self :superior) :get-pane 'lisp))
        (send lisp (clear-window))
        (send ep :clear-window)
        (draw-pattern row col)
        (top-level (send self :make-item-state-array)
          (send sp :lisp-state-list) lisp) ep)))
  (t t))))
(defwhepper (input-pattern-pane :mouse-moves) (x y)
  (let ((item (send self :mouse-sensitive-item x y))
    (button (tv:mouse-buttons)))
    (cond ((send item :mouse-sensitive-item) :update-one-item :steppe) t)
    (t nil))
    (continue-whepper x y))
(defllaver output-pattern-pane ()
  (tv:pane-mixin tv:window))
(defllaver lisp-pane ()
  (tv:lisp-listener-pane tv:window))
 defeval interace ()
  :repltable-instance-variables
  (default-init-pplist
  :panes
    ;((stored-patterns-stream)
      :lable , (tolist-string "stored-patterns")
      ;(lisp) :save-bits nil
      (input-pattern-stream)
      :lable , (tolist-string "input-pattern")
      ;(lisp) :save-bits nil
      (output-pattern-stream)
      :lable , (tolist-string "output-pattern")
      ;(lisp) :save-bits nil)
    (lisp lisp-pane)
  :configurations
    ;"(main (layout
      (main row input-pattern-middle lisp)
      (middle row input-pattern output-pattern))
      (sizes
        (main (stored-patterns 0.50)
        (middle (input-pattern 0.50)
        (lisp (even))
        (middle (input-pattern 0.50)
        (output-pattern (even)))))
    :configurations "main")
  (defmethod (interface-rafter :init) ()
    (reset ignore)
    (let ((sp (send self :get-pane 'stored-patterns))
      (lisp (send self :get-pane 'input-pattern))
      (send sp :set-input-pattern-stream sp)
      (send lisp :set-store-patterns-stream sp)))
```
(defun add-bits (on-bits-list on-bit)
  (loop for wm-list - (mein-list on-bit on-bits-list)
       until (and (null mein-list) (push on-bit on-bits-list)
                (return on-bits-list))
    do
    (if (= (logxor q r) 1) (+ ,sum-value 3) ,sum-value)))
```

```lisp
(defun closure-count (pattern-length q store-vectors-list)
  "Loop for store-vector in ,store-vectors-list
   collect (let (a store-vector)
               (declare (type array-register s))
               (loop for i from 0 below ,pattern-length
                    count (eq (bit ,q ,i) (bit s ,i))))")
```

```lisp
(defun make-q-array (length initial-contents)
  (make-array length :element-type (unsigned-byte 1) :initial-element initial-contents :fill-pointer t))
```

```lisp
(defun retrieve (file window-frame)
  (let ((sp (send window-frame :get-pane 'input-pattern))
        (list (- (item-right pa) left)
              (- (item-bottom pa) top))
        do
        (setf (item-state pa) p)
        (setf (item-name pa) h)
        (draw-pattern (car "pattern-row-col") (cdr "pattern-row-col") p sp
                      (list (- (item-right pa) left)
                            (- (item-bottom pa) top))
                      left top))
    (send sp :draw-grid nil nil)
    (send sp :set-state-list 'store-array-list)
    (send sp :set-pairs-state-list
      (loop for s in (send sp :state-list)
            collect (convert-to-one s))
      (send sp :set-height-list 'store-height-list))")
```

```lisp
(defun close? (x)
  (cond ((= ,x 0) 1)
        (t 0)))
```

```lisp
(defun add-bits (on-bits-list on-bit)
  (loop for wm-list = (member on-bit on-bits-list)
        until (and (null wm-list) (push on-bit on-bits-list) (return on-bits-list))
    do
    (if (= (logxor q r) 1) (+ ,sum-value 3) ,sum-value)))
```

```lisp
(defun make-q-array (length initial-contents)
  (make-array length :element-type (unsigned-byte 1) :initial-element initial-contents :fill-pointer t))
```

```lisp
(defun closure-count (pattern-length q store-vectors-list)
  "Loop for store-vector in ,store-vectors-list
   collect (let (a store-vector)
               (declare (type array-register s))
               (loop for i from 0 below ,pattern-length
                    count (eq (bit ,q ,i) (bit s ,i))))")
```

```lisp
(defun add-bits (on-bits-list on-bit)
  (loop for wm-list = (member on-bit on-bits-list)
        until (and (null wm-list) (push on-bit on-bits-list) (return on-bits-list))
    do
    (if (= (logxor q r) 1) (+ ,sum-value 3) ,sum-value)))
```

```lisp
(defun close? (x)
  (cond ((= ,x 0) 1)
        (t 0)))
```
(setq on-bits-list (remove on-bit on-bits-list))
(setq on-bit (= on-bit 1))

(defun d-index-new (q-value stored-vectors-list sum-value-list dim-index)
  ;;optional (window pp))
(let ((pos-sum nil)
       (neg-sum nil))
  exact-match)
(setq exact-match
  (loop for b in stored-vectors-list
        for b-value = (bit b dim-index)
        for s-value in sum-value-list
        do
          (if (null s-value)
              (return b))
        (multiple-value-bind (bucket val123)
            (sum-third-term-aux s-value q-value b-value)
              (case bucket
                (1 (setq pos-sum (add-bits pos-sum val123)))
                (0 (setq neg-sum (add-bits neg-sum val123)))))
  ;;(print exact-match 1))
  ;;if exact-match
  ;;exact-match
  (loop for max-pos = ((if (null pos-sum) -1 (apply #'+ max-pos sum)))
        for max-neg = ((if (null neg-sum) -1 (apply #'+ max-neg sum))))
        until (or (and (-1 max-pos max-neg) (return nil))
                 (and (= max-pos max-neg) (return 1) (return 0)))).
  (setq pos-sum (delete max-pos pos-sum))
  (setq neg-sum (delete max-neg neg-sum))
)
)

(defun d-dimensions (pattern-length query-array-vector stored-vector-list
  disturb-factor stored-height-list result-array
  ;;optional (process-index-list nil))
(let ((result-array)
  (q query-array-vector)
  a-1-2)
  (declare (sys:array-register r q))
  (setq a-1-2 (sum-1-2 (sum-1-2 dim-1-2-term pattern-length
        (closeness-count pattern-length q stored-vector-list)
        disturb-factor stored-height-list)))
  ;;(print a-1-2 1))
  (if process-index-list
      (loop for i in process-index-list
        for qbit-value = (bit q i)
        for d-value = (d-index-new qbit-value stored-vector-list a-1-2)
        until (and (typep d-value 'array) (return d-value))
        do
          ;;(break "in process-index-list")
          ;;if format 1 "-4 d-value = -4 d-value"
          ;;(if = (logxor pre-bit-value d-value)
          ;;  1)
          ;;(setf (bit r i) '*))
        (setq process-index-list
          (loop for i from 0 below pattern-length
            for pre-bit-value = (bit r i)
            for qbit-value = (bit q i)
            for d-value = (d-index-new qbit-value stored-vector-list a-1-2)
            do
              (format t "-4 d-value = -4 d-value"
                (logxor pre-bit-value d-value)
                1)
            (setf (bit r i) (flip pre-bit-value)
                (list i)))
          (values r process-index-list))
    (return nil))
)

(defun top-level (query-array-vector stored-vector-list stored-height-list
  ;;optional (window ti))
(let ((pattern-length (array-total-size query-array-vector))
  result-array
  process-index-list)
  (setq result-array (make-array pattern-length))
  (copy-array-contents query-array-vector result-array)
  (format window "-4 First iteration!!")
  (multiple-value-setq (result-array process-index-list)
    (d-dimensions pattern-length query-array-vector stored-vector-list 5
      stored-height-list result-array))
  (when (and process-index-list
                (listp process-index-list))
    (format window "-4 Second iteration!!")
    (format window "-4 Process-index-list = -5 process-index-list"
      (d-dimensions pattern-length result-array stored-vector-list 3
        stored-height-list result-array process-index-list))
  result-array))
<table>
<thead>
<tr>
<th>YEARLY CONSTANT $ DESCRIPTION</th>
<th>COST</th>
<th>QTY</th>
<th>ENERGY TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEANING</td>
<td>1310</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEATING</td>
<td>13026</td>
<td>304</td>
<td>NAT GAS</td>
</tr>
<tr>
<td>A/C</td>
<td>14</td>
<td>114</td>
<td>ELECTRIC</td>
</tr>
<tr>
<td>FNS/LGHTS</td>
<td>16</td>
<td>130</td>
<td>ELECTRIC</td>
</tr>
</tbody>
</table>

#### DELETE ESTIMATE DATA

Initial menu item "8" is selected to delete estimate data. CCMAS tells a user how data may be deleted, then asks for the CCMAS-ID.

**DATA CAN BE DELETED BY ENTERING:**

- CCMAS-ID; CCMAS-ID AND RUN #; OR CCMAS-ID, RUN# AND SET#;
- ENTER PROGRAM, PROJECT, FACILITY-ID (CCMAS-ID);
- ENTER RUN NUMBER;

If the user inputs a <CR> instead of a run number CCMAS responds:

**DO YOU WANT TO DELETE ALL ESTIMATE DATA FOR:**

CCMAS-ID: XXXXXXXXXX

If the user inputs a run number, CCMAS responds:

**ENTER SET NUMBER:**

If the user inputs a <CR> instead of a set number CCMAS responds:

**DO YOU WANT TO DELETE ALL ESTIMATE DATA FOR:**

CCMAS-ID: XXXXXXXXXX

RUN NUMBER: X

If the user enter a set number CCMAS responds:

**DO YOU WANT TO DELETE ALL ESTIMATE DATA FOR:**

CCMAS-ID: XXXXXXXXXX

RUN NUMBER: X

SET NUMBER: X

A "No" response to the DO YOU WANT TO DELETE ALL ESTIMATE DATA FOR: returns the user to the previous data input. To a "YES" reply, CCMAS responds:

**DATA WILL BE DELETED**

DELETING ESTIMATE DATA FOR:

CCMAS-ID: XXXXXXXXXX

RUN NUMBER: X (If Specified)

SET NUMBER: X (If Specified)

**DATA DELETED**

The user is then returned to the initial menu.
What is claimed is:
1. A method of comparing, with the aid of a computing system, a query pattern with a set of stored patterns, said method comprising the steps of:
   a) creating a library of stored patterns in a desired format wherein each pattern comprises a set of elements, each pattern having a relative time-dependent importance/avoidance property;
   b) presenting a query pattern comprising a set of elements representative of the entity to be matched in the same size and format as the stored patterns;
   c) comparing the query pattern with each of the stored patterns, on an element-by-element basis and determining a first degree of match for each stored pattern;
   d) indicating a complete match if one exists;
   e) creating, if no complete match exists, a derived query pattern having the same number of elements as the original query pattern and the stored patterns, said derived query pattern being created as follows:
   1) determining a set of second change numbers D according to the following formula:

\[
D_1(i); X; B^1, \ldots, B^n = C(1,i) p_{(i)}^{R(i,0)} [p_{(i)}^{L(i,0)}] S_1(X,i) + \\
C(2,i) p_{(i)}^{R(i,0)} [p_{(i)}^{L(i,0)}] S_1(X,i) + \cdots + C(k,i) p_{(i)}^{R(i,0)} [p_{(i)}^{L(i,0)}] S_k(X,i)
\]

where:
- \(D_1(i)\) is the first change number for the \(i^{th}\) element of the query pattern,
- \(t\) indicates a function of time,
- \(C(i,i)\) is the algebraic sign of the relative importance property at time \(t\) of stored pattern \(B^i\),
- \(p_{(i)}^{R(i,0)}\) is the magnitude of the relative importance property of stored pattern \(B^i\) at time \(t\),
- \(J(X,i)\) is the degree of match with stored pattern \(B^i\),
- \((S_1(X,i))\) is:
  - \(-1\) if the \(i^{th}\) element of \(B^i\) and the \(i^{th}\) element of \(X\) are both 0;
  - 1 if the \(i^{th}\) element of \(B^i\) and the \(i^{th}\) element of \(X\) are both 1;
  - \(-p_3\) if the \(i^{th}\) element of \(B^i\) is 0 and the \(i^{th}\) element of \(X\) is 1;
  - \(p_3\) if the \(i^{th}\) element of \(B^i\) is 1 and the \(i^{th}\) element of \(X\) is 0;
- 2) setting the value of the \(i^{th}\) element of the derived query pattern at 1 if the value of the \(i^{th}\) element of the query pattern is 1 and the first change number for the \(i^{th}\) element is positive;
- 3) setting the value of the \(i^{th}\) element of the derived query pattern at 1 if the value of the \(i^{th}\) element of the query pattern is 1 and the first change number for the \(i^{th}\) element is zero;
- 4) setting the value of the \(i^{th}\) element of the derived query pattern at 0 if the value of the \(i^{th}\) element of the query pattern is 1 and the first change number for the \(i^{th}\) element is negative;
- 5) setting the value of the \(i^{th}\) element of the derived query pattern at 1 if the value of the \(i^{th}\) element of the query pattern is 0 and the first change number for the \(i^{th}\) element is positive;
- 6) setting the value of the \(i^{th}\) element of the derived query pattern at 0 if the value of the \(i^{th}\) element of the query pattern is 0 and the first change number for the \(i^{th}\) element is zero;
- 7) setting the value of the \(i^{th}\) element of the derived query pattern at 0 if the value of the \(i^{th}\) element of the query pattern is 0 and the first change number for the \(i^{th}\) element is negative;
- 8) comparing the derived query pattern with each of the stored patterns, on an element by element basis and determining a second degree of match;
- 9) indicating an answer pattern, said answer pattern being the derived query pattern if a complete match exists between the derived query pattern and a stored pattern,
- 10) creating, if no complete match exists, an answer pattern having the same number of elements as the derived query pattern and the stored patterns, said answer pattern being created as follows:
   1) determining a set of second change numbers \(D_2\) according to the following formula:

\[
D_2(i); X'; B^1, \ldots, B^n = C(1,0) p_{(i',0)}^{R(0,0)} [p_{(i',0)}^{L(0,0)}] S_1(X',i) + \\
C(2,0) p_{(i',0)}^{R(0,0)} [p_{(i',0)}^{L(0,0)}] S_1(X',i) + \cdots + C(k,0) p_{(i',0)}^{R(0,0)} [p_{(i',0)}^{L(0,0)}] S_k(X',i)
\]

where:
- \(D_2(i)\) is the second change number for the \(i^{th}\) element of the query pattern,
- \(t\) indicates a function of time,
- \(C(i,i)\) is the algebraic sign of the relative importance property (or zero) of the stored pattern \(B^i\) at time \(t\),
- \(p_{(i)}^{R(0,0)}\) is the magnitude of the relative importance property of stored pattern \(B^i\),
- \(J(X',i)\) is the second degree of match \(B^i\) and \(X'\),
- \((S_1(X',i))\) is:
  - \(-1\) if the \(i^{th}\) element of \(B^i\) and the \(i^{th}\) element of \(X'\) are both 0;
  - 1 if the \(i^{th}\) element of \(B^i\) and the \(i^{th}\) element of \(X'\) are both 1;
  - \(-p_3\) if the \(i^{th}\) element of \(B^i\) is 0 and the \(i^{th}\) element of \(X'\) is 1;
  - \(p_3\) if the \(i^{th}\) element of \(B^i\) is 1 and the \(i^{th}\) element of \(X'\) is 0;
- 2) setting the value of the \(i^{th}\) element of the answer pattern at 1 if the value of the \(i^{th}\) element of the query pattern is 0 and both the first change number and the second change number for the \(i^{th}\) element are positive;
- 3) setting the value of the \(i^{th}\) element of the answer pattern at 0 if the value of the \(i^{th}\) element of the query pattern is 0 and both the first change number and the second change number for the \(i^{th}\) element are negative;
- 4) setting the value of the \(i^{th}\) element of the answer pattern at 0 if the value of the \(i^{th}\) element of the query pattern is 0 and the first change number for the \(i^{th}\) element is negative;

5) setting the value of the \( i \)th element of the answer pattern at 1 if the value of the \( i \)th element of the query is 1 and the first change number is positive;
6) setting the value of the \( i \)th element of the answer pattern at \( * \), where \( * \) indicates ambiguity, if the value of the \( i \)th element of the query pattern is 0, the first change number for the \( i \)th element is positive and the second change number is negative;

and setting the value of the \( i \)th element of the answer pattern at \( * \) if the value of the \( i \)th element of the query pattern is 1, the first change number is negative and the second change number for the \( i \)th element is positive; and

i) converting answer pattern from representational format to desired format with ambiguous elements distinguished from determined elements in some manner.

2. A method, using a computing system, for calculating the similarity between a first pattern, called a "query pattern", and at least one of a library of \( k \) second patterns, called "stored patterns," the query pattern being representative of a physical entity and being represented by a first sequence of successive feature elements \( B = \{B_1, B_2, \ldots, B_n\} \) where \( n \) equals the number of feature elements, each stored pattern being representative of a physical entity and being represented by a second sequence of successive feature elements \( X = \{X_1, X_2, \ldots, X_n\} \) where \( n \) equals the number of feature elements of each pattern.

The method comprising the steps of:

(a) comparing the query pattern with the \( m \)th stored pattern on an element by element basis;
(b) counting as a "first degree of match" for the \( m \)th stored pattern, the number of elements of the query pattern which are equal to the corresponding element of the \( m \)th stored pattern;
(c) outputting a "matched pattern" indication if the first degree of match for the \( m \)th stored pattern is equal to \( n \);
(d) storing the count for the first degree of match for the \( m \)th stored pattern if less than \( n \);
(e) repeating steps (a) through (d) for each remaining stored pattern;

(f) computing, using the query pattern and the first change numbers for the set of stored patterns using the query pattern and the second degree of match;

(h) comparing the derived query pattern to the \( m \)th stored pattern, the number of elements of the derived query pattern which are equal to the corresponding element of the \( m \)th stored pattern;

(i) outputting a "matched pattern" indication if the second degree of match for the \( m \)th stored pattern is equal to \( n \);

(k) repeating the count for the second degree of match for the \( m \)th stored pattern if less than \( n \);

(l) repeating steps (h) through (k) for each remaining stored pattern;

(m) computing a second set of \( n \) change numbers for the set of stored patterns using the query pattern and the second degree of match;
(n) computing an "answer pattern" using the query pattern, the first set of change numbers and the second set of change numbers.

3. The method of claim 2, further comprising the further step of computing the second set of \( n \) change numbers according to the following equation:

\[
D_2[i,j]: X_iB_1, B_2, \ldots B_n = C(i,0) p_1^{R(i,0)} [p_2^{L(i,0)}] S_1(X,j) + \ldots + C(i,0) p_1^{R(i,0)} [p_2^{L(i,0)}] S_n(X,j) + \ldots +
\]

where:

(1) \( D_2[i] \) is the second change number for the \( i \)th element of the query pattern;
(2) \( t \) indicates a function of time;
(3) \( C(i,t) \) is the algebraic sign of the relative importance property (or zero);
(4) \( p_1^{R(i,t)} \) is the magnitude of the relative importance property;
(5) \( J(X,i) \) is the second degree of match (the number of elements of \( B \) that agree with the corresponding element of \( X \));
(6) \( S(X,i) \) is:
- 1 if the element of \( B \) and the \( i \)th element of \( X \) are both 1;
- 0 if the \( i \)th element of \( B \) is 0 and the \( i \)th element of \( X \) is 1; and
- \( p_3 \) if the \( i \)th element of \( B \) is 1 and the \( i \)th element of \( X \) is 0;

4. A method of matching a first query pattern, taking the form of a set \( X = x_1 \) with a plurality of stored data patterns, taking the form of a matrix \( B = b_{ij} \), where \( i = 1, \ldots, n \) is the number of components in each pattern and \( j = 1, \ldots, k \) is the number of stored data patterns, said method comprising the steps of:

(a) for each stored data pattern, counting the number of components which are identical to corresponding components in said first query pattern, thereby...
forming a set of match numbers \( M = m_{ij} \) where 
\[ j = 1, \ldots, k; \]
(b) if any match number \( m_j = n \), then displaying the 
respective \( j \)th stored data pattern an an output pat-
tern set \( Q_j = q_j \) indicating a match;
(c) if no match number \( m_j = n \), then determining a 
second query pattern \( X'' = x'' \) by modifying said 
first query pattern, component by component, in 
dependence upon both a first, global influence of 
all stored patterns on all components of said first 
query pattern and a second, particular influence of 
all stored patterns on each respective component of 
said first query pattern, wherein such step further 
comprises the steps:
(1) multiplying each match number \( m_j \) of a match 
set \( M \) by a first disturbance factor \( L \) to produce 
a set \( LM \);
(2) determining a sign matrix \( Sgn \) for all compo-
ments \( sgn_i \) by setting each component equal to 
\(-1 \) if the corresponding stored pattern compo-
ponent \( b_j \) is \( 0 \), and to \(+1 \) if the stored pattern 
component is \( 1 \);
(3) determining a magnitude matrix \( Mag \) for all 
components according to the formula \( mag_i = [-\) 
\( \log_2(b_j, x_j) = 1 \) \( ] \), where \( N \) is a magnification 
factor;
(4) determining an exponent matrix \( Exp = LM + \) 
\( Mag \) for all components according to the for-
\( mula \exp_i = exp_i = \exp_i, k \) for each 
\( \exp_i, k \) of \( j \) where \( sgn_i, j > 0 \),
\( Neg.sum_i = P(exp_i, l) + \ldots + P(exp_i, k) \) for each 
\( \exp_i, l \) of \( j \) where \( sgn_i, j < 0 \);
(5) forming the positive and negative sums for all 
components as follows:
\[ \text{Pos.sum} = P(exp_i, l) + \ldots + P(exp_i, k) \]
\( \text{Neg.sum} = P(exp_i, l) + \ldots + P(exp_i, k) \)
(6) determining a second query pattern set \( X'' = x'' \)
for all \( i \) as follows:
if \( Pos.sum_i \geq Neg.sum_i \), then \( x'' = 1 \),
else \( x'' = 0 \);
(d) repeating steps (a) and (b) using said second query 
pattern in place of said first query pattern;
(3) if no match number \( m_j = n \), then determining a 
third query pattern \( X''' = x''' \) by modifying said 
second query pattern, component by component, in 
dependence upon both a third, global influence of 
all stored patterns on all components of said 
second query pattern and a fourth, particular influence 
of all stored patterns on each respective component of 
said second query pattern, said third and 
fourth influences being less than said first and sec-
ond influences, respectively; and
(f) displaying as said output pattern \( O-32 o_j \), com-
ponent by component, those respective components 
of said third query pattern that have been modified 
at most once from said first query pattern.
5. The method defined in claim 4, further comprising 
the step of determining a response set \( R = r_j \) for all \( i \) as 
follows:
\[ r_j = x_j \] if \( x_j = x_j \) and
\[ r_j = \text{set flag} \] if \( x_j = x_j \).
6. The method defined in claim 5, wherein step (e) 
includes the steps (1) through (6), using a second dis-
urbance factor \( L' \) which is less than said first disturbance 
factor \( L \), for those components \( i \) that have a set flag.
7. The method defined in claim 4, wherein a height 
factor set \( H = h_j \) is associated with the components of 
each stored pattern \( b_j \), each height factor component \( h_j \)
being indicative of the relative importance of each 
stored pattern with respect to the other stored patterns, 
and wherein said height factor set \( H \) is added to the set 
\( LM \) for purposes of determining the matrix \( Exp \).
8. The method defined in claim 7, wherein at least 
some of said height factor components \( h_j \) are a function 
of time.
9. The method defined in claim 7, wherein each of 
said height factor components \( h_j \) is an integer.
10. The method defined in claim 4, wherein a sign 
factor set \( C = c_j \) is associated with the components of 
each stored pattern \( b_j \), each sign factor component \( c_j \)
being indicative of whether the pattern component is to 
be sought, avoided or ignored, and wherein said sign 
factor set \( C \) is multiplied by the sign matrix \( Sgn \) for the 
purposes of determining the positive and negative sums.
11. The method defined in claim 10, wherein at least 
some of said sign factor components \( c_j \) are a function 
of time.
12. The method defined in claim 10, wherein each of 
said sign factor components \( c_j \) is an integer.
13. The method defined in claim 10, wherein each of 
said sign factor components \( c_j \) assumes one of the values 
\(+1, 0 \) and \(-1 \).
14. The method defined in claim 4, wherein said first 
disturbance factor \( L \) is in the range of 2 to 20.
15. The method defined in claim 4, wherein said first 
disturbance factor \( L \) is in the range of 3 to 7.
16. The method defined in claim 4, wherein said first 
disturbance factor \( L \) is 3.
17. The method defined in claim 16, wherein said 
second disturbance factor \( L' \) is 2.
18. The method defined in claim 4, wherein said mag-
ification factor \( N \) is in the range of 2 to 20.
19. The method defined in claim 4, wherein said mag-
ification factor \( N \) is in the range of 3 to 7.
20. The method defined in claim 4, wherein said mag-
ification factor \( N \) is 3.