Module 3

Matching techniques:

Matching is the process of comparing two or more structures to discover their likenesses or differences. The structures may represent a wide range of objects including physical entities, words or phrases in some language, complete classes of things, general concepts, relations between complex entities, and the like. The representations will be given in one or more of the formalisms like FOPL, networks, or some other scheme, and matching will involve comparing the component parts of such structures.

Matching is used in a variety of programs for different reasons. It may serve to control the sequence of operations, to identify or classify objects, to determine the best of a number of different alternatives, or to retrieve items from a database. It is an essential operation such diverse programs as speech recognition, natural language understanding, vision, learning, automated reasoning, planning, automatic programming, and expert systems, as well as many others.

In its simplest form, matching is just the process of comparing two structures or patterns for equality. The match fails if the patterns differ in any aspect. For example, a match between the two character strings acdebfba and acdebeba fails on an exact match since the strings differ in the sixth character positions.

In more complex cases the matching process may permit transformations in the patterns in order to achieve an equality match. The transformation may be a simple change of some variables to constants, or ti may amount to ignoring some components during the match operation. For example, a pattern matching variable such as ?x may be used to permit successful matching between the two patterns (a b (c d) e) and (a b ?x e) by binding ?x to (c, d). Such matching are usually restricted in some way, however, as is the case with the unification of two classes where only consistent bindings are permitted. Thus, two patterns such as

(a b (c d) e f) and (a b ?x e ?x)

would not match since ?x could not be bound to two different constants.

In some extreme cases, a complete change of representational form may be required in either one or both structures before a match can be attempted. This will be the case, for

example, when one visual object is represented as a vector of pixel gray levels and objects to be matched are represented as descriptions in predicate logic or some other high level statements. A direct comparison is impossible unless one form has been transformed into the other.

In subsequent chapters we will see examples of many problems where exact matches are inappropriate, and some form of partial matching is more meaningful. Typically in such cases, one is interested in finding a best match between pairs of structures. This will be the case in object classification problems, for example, when object descriptions are subject to corruption by noise or distortion. In such cases, a measure of the degree of match may also be required.

Other types of partial matching may require finding a match between certain key elements while ignoring all other elements in the pattern. For example, a human language input unit should be flexible enough to recognize any of the following three statements as expressing a choice of preference for the low-calorie food item

I prefer the low-calorie choice.

I want the low-calorie item

The low-calorie one please.

Recognition of the intended request can be achieved by matching against key words in a template containing “low-calorie” and ignoring other words except, perhaps, negative modifiers.

Finally, some problems may obviate the need for a form of fuzzy matching where an entity’s degree of membership in one or more classes is appropriate. Some classification problems will apply here if the boundaries between the classes are not distinct, and an object may belong to more than one class.

Fig 8.1 illustrates the general match process where an input description is being compared with other descriptions. As stressed earlier, their term object is used here in a general sense. It does not necessarily imply physical objects. All objects will be represented in some formalism such a s a vector of attribute values, prepositional logic or FOPL statements, rules, frame-like structures, or other scheme. Transformations, if required, may involve simple instantiations or unifications among clauses or more complex operations such

as transforming a two-dimensional scene to a description in some formal language. Once the descriptions have been transformed into the same schema, the matching process is performed element-by-element using a relational or other test (like equality or ranking). The test results may then be combined in some way to provide an overall measure of similarity. The choice of measure will depend on the match criteria and representation scheme employed.

The output of the matcher is a description of the match. It may be a simple yes or no response or a list of variable bindings, or as complicated as a detailed annotation of the similarities and differences between the matched objects.

`To summarize then, matching may be exact, used with or without pattern variables, partial, or fuzzy, and any matching algorithm will be based on such factors as

Choice of representation scheme for the objects being matched,

Criteria for matching (exact, partial, fuzzy, and so on),

Choice of measure required to perform the match in accordance with the

chosen criteria, and

Type of match description required for output.

In the remainder of this chapter we examine various types of matching problems and their related algorithms. We bin with a description of representation structures and measures commonly found in matching problems. We next look at various matching techniques based on exact, partial, and fuzzy approaches. We conclude the chapter with an example of an efficient match algorithm used in some rule-based expert systems.

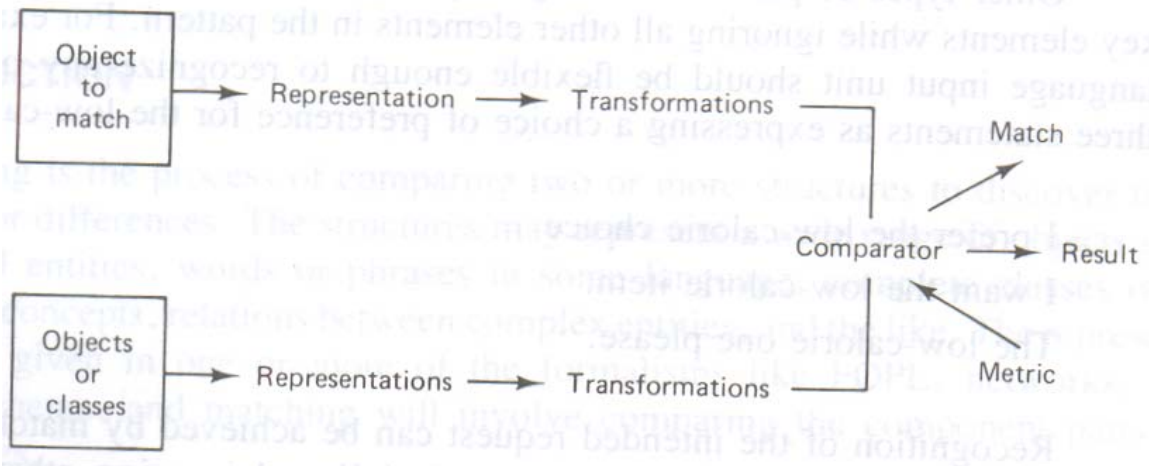


Fig Typical Matching Process

Structures used in Matching

- The types of list structures represent clauses in propositional or predicate logic such as (or ~(MARRIED ?x ?y) ~(DAUGHTER ?z ?y) (MOTHER ?y ?z)) or rules such as (and ((cloudy-sky) (low-bar-pressure) (high-humidity)) (conclude (rain likely)) or fragments of associative networks in below Fig
- The other common structures include strings of characters $a_1\ a_2 \ldots a_k$, where the a_i belong to given alphabet A, vector $X = (x_1\ x_2 \ldots x_n)$, where the x_i represents attribute values, matrices M (rows of vectors), general graphs, trees and sets

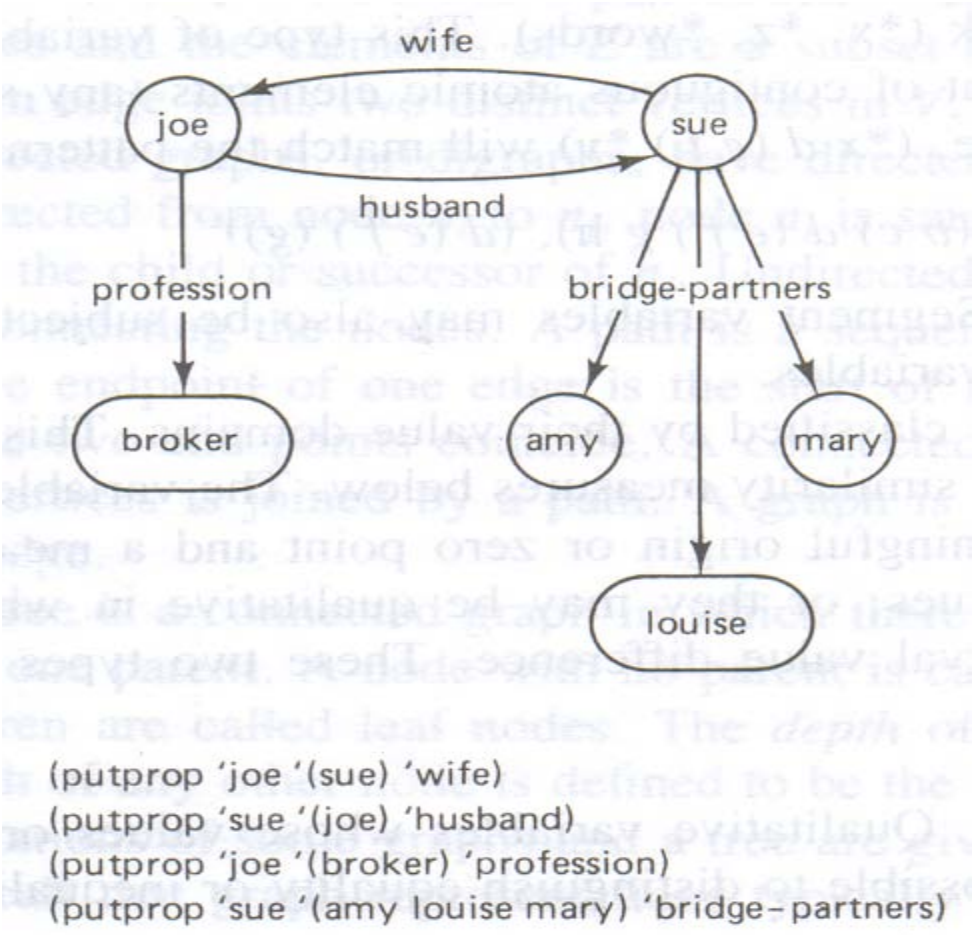


Fig Fragment of associative network and corresponding LISP Code

- Variables
  - The structures are constructed from basic atomic elements, numbers and characters

- Character string elements may represent either constants or variables
- If variables, they may be classified by either the type of match permitted or their value domains
- An open variable can be replaced by a single item
- Segment variable can be replaced by zero or more items
- Open variable are replaced with a preceding question mark (?x. ?y, ?class)
- They may match or assume the value of any single string element or word, but they are subject to consistency constraints
- For example, to be consistent, the variable ?x can be bound only to the same top level element in any single structure
- Thus (a ?x d ?x e) may match (a b d b e), but not (a b d a e)
- Segment variable types will be preceded with an asterisk (*x, *z, *words)
- This type of variable can match an arbitrary number or segment of contiguous atomic elements
- For example, (* x d (e g) *y) will match the patterns

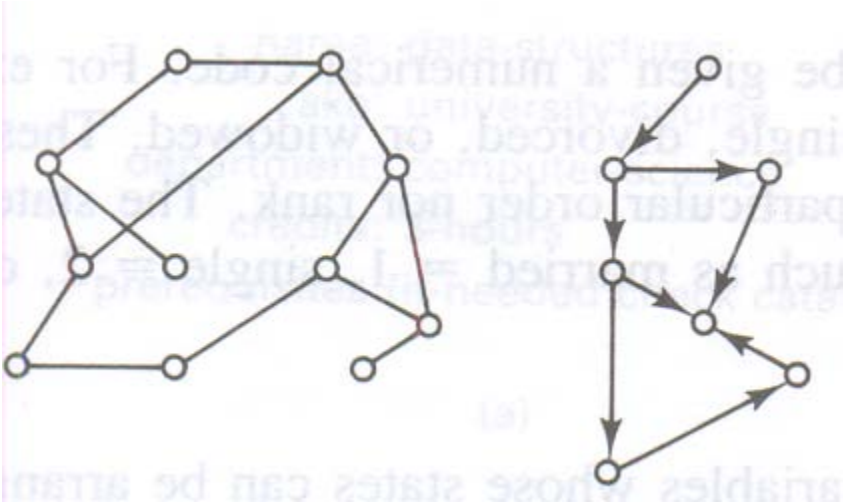
  (a (b c) d (e f) g h), (d (e f) (g))

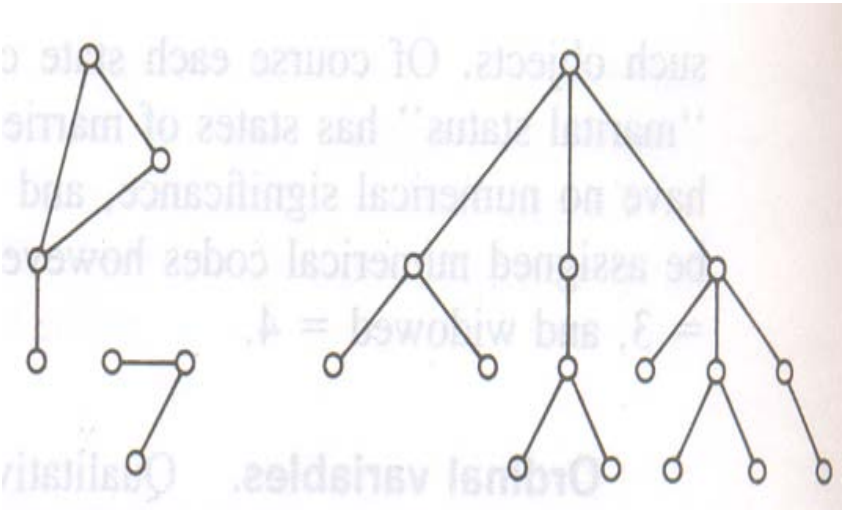
- Subject variable may also be subject to consistency constraints similar to open variables

- Nominal variables
  - Qualitative variables whose values or states have no order nor rank
  - It is possible to distinguish between equality or inequality between two objects
  - Each state can be given a numerical code
  - For example, “marital status” has states of married, single, divorced or widowed. These states could be assigned numerical codes, such as married = 1, single = 2, divorced = 3 and widowed = 4
- Ordinal variables
  - Qualitative variables whose states can be arranged in a rank order
  - It may be assigned numerical values
  - Foe example, the states very tall, tall, medium, short and very short can be arranged in order from tallest to shortest and can be assigned an arbitrary scale of 5 to 1
- Binary variable

- Qualitative discrete variables which may assume only one of two values, such as 0 or 1, good or bad, yes or no, high or low

- Interval variables or Metric variables
  - Qualitative variables which take on numeric values and for which equal differences between values have the same significance
  - For example, real numbers corresponding to temperature or integers corresponding to an amount of money are considered as interval variables
- Graphs and Trees
  - A graph is a collection of points called vertices, some of which are connected by line segments called edges
  - Graphs are used to model a wide variety of real-life applications, including transportation and communication networks, project scheduling, and games
  - A graph G = (V, E) is an ordered pair of sets V and E. the elements V are nodes or vertices and the elements of E are a subset of V X V called edges
  - An edge joints two distinct vertices in V
  - Directed graphs or digraphs, have directed edges or arcs with arrows
  - If an arc is directed from node n_i to n_j , node n_i is said to be a parent or successor of n_j and n_j is the child or successor of n_i
  - Undirected graphs have simple edges without arrows connecting the nodes
  - A path is a sequence of edges connecting two modes where the endpoint of one edge is the start of its successor
  - A cycle is a path in which the two end points coincide
  - A Connected graph is a graph for which every pair of vertices is joined by a path
  - A graph is complete if every element of V X V is an edge
  - A tree is a connected graph in which there are no cycles and each node has at most one parent below
  - A node with no parent is called root node
  - A node with no children is called leaf node
  - The depth of the root node is defined as zero
  - The depth of any other node is defined to be the depth of its parent plus 1





- Sets and Bags
  - A set is represented as an unordered list of unique elements such as the set (a d f c) or (red blue green yellow)
  - A bag is a set which may contain more than one copy of the same member a, b, d and e
  - Sets and bags are structures used in matching operations

Measure for Matching

- The problem of comparing structures without the use of pattern matching variables. This requires consideration of measures used to determine the likeness or similarity between two or more structures

- The similarity between two structures is a measure of the degree of association or likeness between the object's attributes and other characteristics parts.
- If the describing variables are quantitative, a distance metric is used to measure the proximity
- Distance Metrics
  - For all elements x, y, z of the set E, the function d is metric if and only if

    $d(x, x) = 0$

    $d(x,y) \geq 0$

    $d(x,y) = d(y,x)$

    $d(x,y) \leq d(x,z) + d(z,y)$

  - The Minkowski metric is a general distance measure satisfying the above assumptions
  - It is given by

$$d_p = \left[\sum_{i=1}^{n} |x_i - y_i|^p \right]^{1/p}$$

  - For the case p = 2, this metric is the familiar Euclidian distance. Where p = 1, d_p is the so-called absolute or city block distance
- Probabilistic measures
  - The representation variables should be treated as random variables
  - Then one requires a measure of the distance between the variates, their distributions, or between a variable and distribution
  - One such measure is the Mahalanobis distance which gives a measure of the separation between two distributions
  - Given the random vectors X and Y let C be their covariance matrix
  - Then the Mahalanobis distance is given by

    $D = X' C^{-1} Y$

  - Where the prime (') denotes transpose (row vector) and C^{-1} is the inverse of C
  - The X and Y vectors may be adjusted for zero means bt first substracting the vector means u_x and u_y

- Another popular probability measure is the product moment correlation r, given by

$$r = \frac{Cov(X, Y)}{[Var(X) * Var(Y)]^{1/2}}$$

- Where Cov and Var denote covariance and variance respectively
- The correlation r, which ranges between -1 and +1, is a measure of similarity frequently used in vision applications
- Other probabilistic measures used in AI applications are based on the scatter of attribute values
- These measures are related to the degree of clustering among the objects
- Conditional probabilities are sometimes used
- For example, they may be used to measure the likelihood that a given X is a member of class C_i , $P(C_i| X)$, the conditional probability of C_i given an observed X
- These measures can establish the proximity of two or more objects

- Qualitative measures
  - Measures between binary variables are best described using contingency tables in the below
  - Table

	Variable X	1	0	Totals
Variable Y	1	a	b	a + b
	0	c	d	c + d
	Totals	a + c	b + d	n

  - The table entries there give the number of objects having attribute X or Y with corresponding value of 1 or 0
  - For example, if the objects are animals might be horned and Y might be long tailed. In this case, the entry a is the number of animals having both horns and long tails

- Note that n = a + b + c + d, the total number of objects
- Various measures of association for such binary variables have been defined
- For example

$$\frac{a}{a+b+c+d} = \frac{a}{n}, \quad \frac{a+d}{n}$$

$$\frac{a}{a+b+c}, \quad \frac{a}{b+c}$$

- Contingency tables are useful for describing other qualitative variables, both ordinal and nominal. Since the methods are similar to those for binary variables
- Whatever the variable types used in a measure, they should all be properly scaled to prevent variables having large values from negating the effects of smaller valued variables
- This could happen when one variable is scaled in millimeters and another variable in meters

- Similarity measures
  - Measures of dissimilarity like distance, should decrease as objects become more alike
  - The similarities are not in general symmetric
  - Any similarity measure between a subject description A and its referent B, denoted by s(A,B), is not necessarily equal
  - In general, $s(A,B) \neq s(B,A)$ or "A is like B" may not be the same as "B is like A"
  - Tests on subjects have shown that in similarity comparisons, the focus of attention is on the subject and, therefore, subject features are given higher weights than the referent

- For example, in tests comparing countries, statements like “North Korea is similar to Red China” and “Red China is similar to North Korea” were not rated as symmetrical or equal
- Similarities may depend strongly on the context in which the comparisons are made
- An interesting family of similarity measures which takes into account such factors as asymmetry and has some intuitive appeal has recently been proposed
- Let $O = \{o_1, o_2, \ldots\}$ be the universe of objects of interest
- Let A_i be the set of attributes used to represent o_i
- A similarity measure s which is a function of three disjoint sets of attributes common to any two objects A_i and A_j is given as

$$s(A_i, A_j) = F(A_i \& A_j, A_i - A_j, A_j - A_i)$$

- Where $A_i \& A_j$ is the set of features common to both o_i and o_j
- Where $A_i - A_j$ is the set of features belonging to o_i and not o_j
- Where $A_j - A_i$ is the set of features belonging to o_j and not o_i
- The function F is a real valued nonnegative function

$$s(A_i, A_j) = af(A_i \& A_j) - bf(A_i - A_j) - cf(A_j - A_i) \text{ for some } a,b,c \geq 0$$

- Where f is an additive interval metric function
- The function f(A) may be chosen as any nonnegative function of the set A, like the number of attributes in A or the average distance between points in A

$$S(A_i, A2_j) = \frac{f(A_i \& A_j)}{f(A_i \& A_j) + af(A_i - A_j) + bf(A_j - A_i)}$$

for some $a,b \geq 0$

- When the representations are graph structures, a similarity measure based on the cost of transforming one graph into the other may be used
- For example, a procedure to find a measure of similarity between two labeled graphs decomposes the graphs into basic subgraphs and computes the minimum cost to transform either graph into the other one, subpart-by-subpart

Matching like Patterns

- We consider procedures which amount to performing a complete match between two structures
- The match will be accomplished by comparing the two structures and testing for equality among the corresponding parts
- Pattern variables will be used for instantiations of some parts subject to restrictions
- Matching Substrings
  - A basic function required in many match algorithms is to determine if a substring S_2 consisting of m characters occurs somewhere in a string S_1 of n characters, $m \leq n$
  - A direct approach to this problem is to compare the two strings character-by-character, starting with the first characters of both S_1 and S_2
  - If any two characters disagree, the process is repeated, starting with the second character of S_1 and matching again against S_2 character-by-character until a match is found or disagreements occurs again
  - This process continues until a match occurs or S_1 has no more characters
  - Let i and j be position indices for string S1 and a k position index for S_2
  - We can perform the substring match with the following algorithm

```
i := 0

While i ≤ (n – m + 1) do

begin

        i := i + 1;

        j := i;

        k :=1;

while S1(j) = S2(k) do

begin

        if k = m

                writeln("success")

        else do
```

```
                    begin
                        j := j +1;
                        k := k + 1;
                    end
                end
            end
            writeln("fail")
            end
```

- This algorithm requires m(n - m) comparisons in the worst case
- A more efficient algorithm will not repeat the same comparisons over and over again
- One such algorithm uses two indices I and j, where I indexes the character positions in S_1 and j is set to a "match state" value ranging from 0 to m
- The state 0 corresponds to no matched characters between the strings, while state 1 corresponds to the first letter in S_2 matching character i in S_2
- State 2 corresponds to the first two consecutive letters in S_2 matching letters i and i+1 in S_1 respectively, and so on, with state m corresponding to a successful match
- Whenever consecutive letters fail to match, the state index is reduced accordingly

- Matching Graphs
  - Two graphs G_1 and G_2 match if they have the same labeled nodes and same labeled arcs and all node-to-node arcs are the same
  - If G_2 with m nodes is a subgraph of G_1 with n nodes, where $n \geq m$
  - In a worst cas match, this will require n!/(n - m)! node comparison and $O(m^2)$ arc comparisons
  - Finding subgraph isomorphisms is also an important matching problem
  - An isomorphism between the graphs G_1 and G_2 with vertices V_1, V_2 and edges E1, E2, that is, (V1, E1) and (V2, E2) respectively, is a one-to-one mapping to f between V1 and V2, such that for all v1 € V1, f(v1) = v2, and for each arc e1 € E1 connecting v1 and $v1^{'}$, there is a corresponding arc e2 € E2 connecting f(v1) and $f(v1^{'})$

- Matching Sets and Bags
  - An exact match of two sets having the same number of elements requires that their intersection also have the number of elements
  - Partial matches of two sets can also be determined by taking their intersections
  - If the two sets have the same number of elements and all elements are of equal importance, the degree of match can be the proportion of the total members which match
  - If the number of elements differ between the sets, the proportion of matched elements to the minimum of the total number of members can be used as a measure of likeness
  - When the elements are not of equal importance, weighting factors can be used to score the matched elements
  - For example, a measure such as

    $s(S1,S2) = (\sum w_i N(a_i))/m$

    could be used, where $w_i = 1$ and $N(a_i) = 1$ if a_i is in the intersection; otherwise it is 0

  - An efficient way to find the intersection of two sets of symbolic elements in LISP is to work through one set marking each element on the elements property list and then saving all elements from the other list that have been marked
  - The resultant list of saved elements is the required intersection
  - Matching two bags is similar to matching two sets except that counts of the number of occurrences of each element must also be made
  - For this, a count of the number of occurrences can be used as the property mark for elements found in one set. This count can then be used to compare against a count of elements found in the second set
- Matching to Unify Literals
  - One of the best examples of nontrivial pattern matching is in the unification of two FOPL literals
  - For example, to unify P(f(a,x), y ,y) and P(x, b, z) we first rename variables so that the two predicates have no variables in common
  - This can be done by replacing the x in the second predicate with u to give P(u, b, z)

- Compare the two symbol-by-symbol from left to right until a disagreement is found
- Disagreements can be between two different variables, a nonvariable term and a variable, or two nonvariable terms. If no disagreements is found, the two are identical and we have succeeded
- If disagreements is found and both are nonvariable terms, unification is impossible; so we have failed
- If both are variables, one is replaced throughout by the other. Finally, if disagreement is a variable and a nonvariable term, the variable is replaced by the entire term
- In this last step, replacement is possible only if the term does not contain the variable that is being replaced. This matching process is repeated until the two are unified or until a failure occurs
- For the two predicates P, above, a disagreement is first found between the term f(a,x) and variable u. Since f(a,x) does not contain the variable u, we replace u with f(a,x) everywhere it occurs in the literal
- This gives a substitution set of {f(a,x)/u} and the partially matched predicates P(f(a,x),y ,y) and P(f(a,x), b, z)
- Proceeding with the match, we find the next disagreements pair y and b, a variable and term, respectively
- Again we replace the variable y with the terms b and update the substitution list to get {f(a,x)/u, b/y}
- The final disagreements pair is two variables. Replacing the variable in the second literal with the first we get the substitution set {f(a,x)/u, b/y, y/z}or {f(a,x), b/y, b/z}
- For example, a LISP program which uses both the open and the segment pattern matching variables to find a match between a pattern and a clause

```
(defun match (pattern clause)

  (cond ((equal pattern clause) t )              ; return t if

    ((or (null pattern) (null clause)) nil)      ; equal, nil

                                                 ; if not

    ((or (equal (car pattern) (car clause))      ;not, ?x
```

```
                                                    ; binds
        (equal (car pattern) '?x))                  ; to single
    (match (cdr pattern) (cdr clause)))             ; term, *y
                                                    ; binds
((equal (car pattern) '*y)                          ; to several
(or (match (cdr pattern) (cdr clause))              ;contiguous
(match pattern (cdr clause))))))                   ; terms
```

- When a segment variable is encountered (the *y), match is recursively executed on the cdrs of both pattern and clause or on the cdr of clause and pattern as *y matches one or more than one item respectively

Partial Matching

- For many AI applications complete matching between two or more structures is inappropriate
- For example, input representations of speech waveforms or visual scenes may have been corrupted by noise or other unwanted distortions.
- In such cases, we do not want to reject the input out of hand. Our systems should be more tolerant of such problems
- We want our system to be able to find an acceptable or best match between the input and some reference description
- Compensating for Distortions
  - Finding an object in a photograph given only a general description of the object is a common problems in vision applications
  - For example, the task may be to locate a human face or human body in photographs without the necessity of storing hundreds of specific face templates
  - A better approach in this case would be to store a single reference description of the object

- Matching between photographs regions and corresponding descriptions then could be approached using either a measure of correlation, by altering the image to obtain a closer fit
- If nothing is known about the noise and distortion characteristics, correlation methods can be ineffective or even misleading. In such cases, methods based on mechanical distortion may be appropriate
- For example, our reference image is on a transparent rubber sheet. This sheet is moved over the input image and at each location is stretched to get the best match alignment between the two images
- The match between the two can then be evaluated by how well they correspond and how much push-and-pull distortion is needed to obtain the best correspondence
- Use the number of rigid pieces connected with springs. This pieces can correspond to low level areas such as pixels or even larger area segments

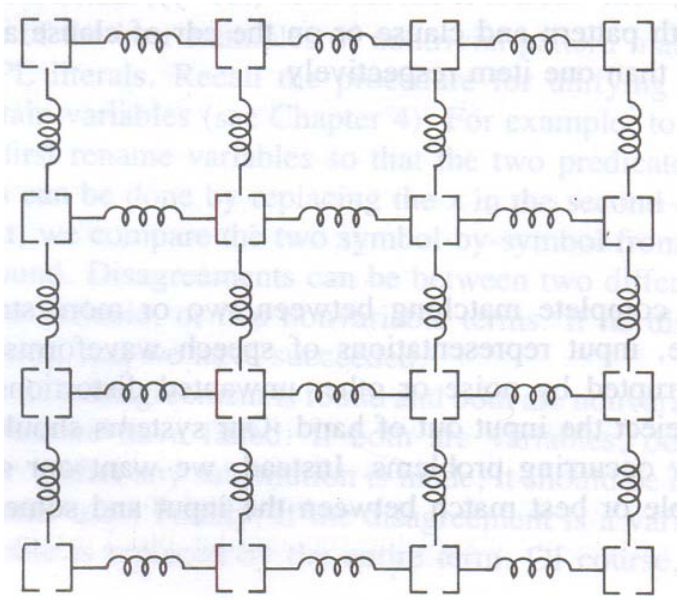


Fig Discrete version of stretchable overlay image

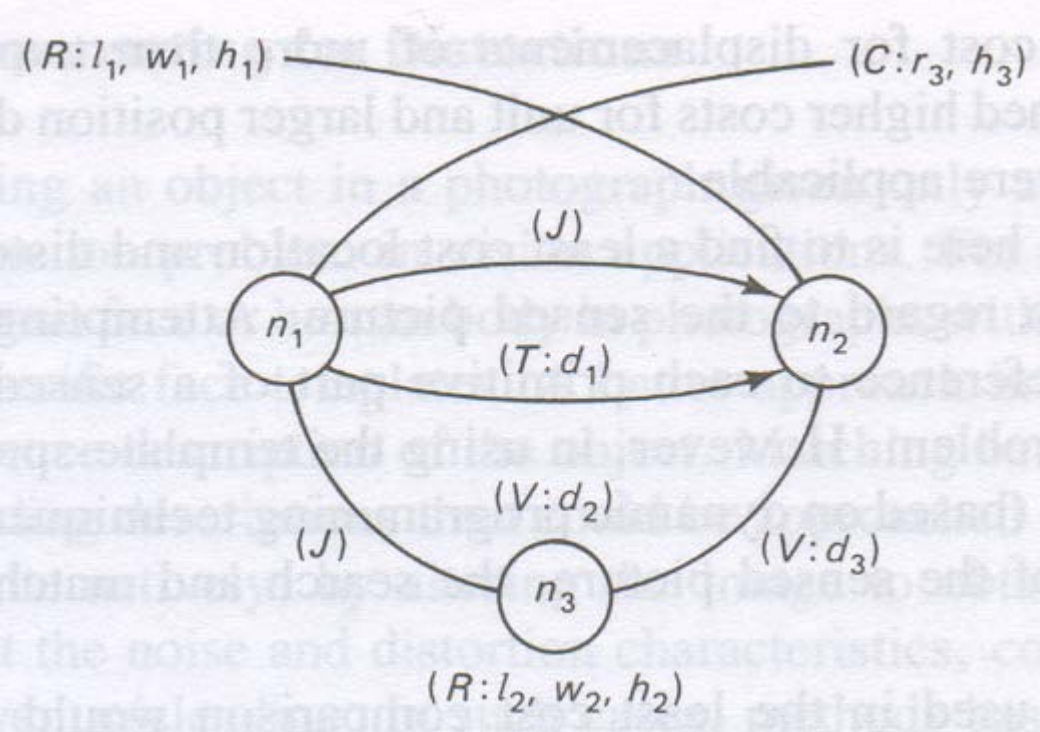
- To model any restrictions such as the relative positions of body parts, nonlinear cost functions of piece displacements can be used
- The costs can correspond to different spring tensions which reflect the constraints
- For example, the cost of displacing some pieces might be zero for no displacement, one unit for single increment displacements in any one of the

permissible directions, two units for two position displacements and infinite cost for displacements of more than two increments. Other pieces would be assigned higher costs for unit and larger position displacements when stronger constraints were applicable

- The matching problem is to find a least cost location and distortion pattern for the reference sheet with regard to the sensed picture
- Attempting to compare each component of some reference to each primitive part of a sensed picture is a combinatorially explosive problem
- In using the template-spring reference image and heuristic methods to compare against different segments of the sensed picture, the search and match process can be made tractable
- Any matching metric used in the least cost comparison would need to take into account the sum of the distortion costs C_d, the sum of the costs for reference and sensed component dissimilarities Cc , and the sum of penalty costs for missing components C_m. Thus, the total cost is given by

$$C_t = C_d + C_c + C_m$$

- Finding Match Differences
  - Distortions occurring in representations are not the only reason for partial matches
  - For example, in problem solving or analogical inference, differences are expected. In such cases the two structures are matched to isolate the differences in order that they may be reduced or transformed. Once again, partial matching techniques are appropriate
  - In visual application, an industrial part may be described using a graph structure where the set of nodes correspond to rectangular or cylindrical block subparts
  - The arc in the graph correspond to positional relations between the subparts
  - Labels for rectangular block nodes contain length, width and height, while labels for cylindrical block nodes, where location can be above, to the right of, behind, inside and so on
  - In Fig 8.5 illustrates a segment of such a graph



- In the fig the following abbreviations are used
- Interpreting the graph, we see it is a unit consisting of subparts, mode up of rectangular and cylindrical blocks with dimensions specified by attribute values
- The cylindrical block n_1 is to the right of n_2 by d_1 units and the two are connected by a joint
- The blocks n_1 and n_2 are above the rectangular block n_3 by d_2 and d_3 units respectively, and so on
- Graphs such as this are called attributed relational graphs (ATRs). Such a graph G is defined as a sextuple G = (N, B, A, G_n, G_b)
- Where N = { n_1, n_2, . . ., n_k}is a set of nodes, A = {a n_1, an_2, . . . , an_k} is an alphabet of node attributes, B = { b_1, b_2, . . . , b_m}is a set of directed branches (b = (n_i, n_j)), and G_n and G_b are functions for generating node and branch attributes respectively
- When the representations are graph structures like ARGs, a similarity measure may be computed as the total cost of transforming one graph into the other
- For example, the similarity of two ARGs may be determined with the following steps:
    - Decompose the ARGs into basic subgraphs, each having a depth of one

- Compute the minimum cost to transform either basic ARG into the other one subgraph-by-subgraph
- Compute the total transformation cost from the sum of the subgraph costs

➢ An ARG may be transformed by the three basic operations of node or branch deletions, insertions, or substitutions, where each operation is given a cost based on computation time or other factors

The RETE matching algorithm

- One potential problem with expert systems is the number of comparisons that need to be made between rules and facts in the database.
- In some cases, where there are hundreds or even thousands of rules, running comparisons against each rule can be impractical.
- The Rete Algorithm is an efficient method for solving this problem and is used by a number of expert system tools, including OPS5 and Eclipse.
- The Rete is a directed, acyclic, rooted graph.
- Each path from the root node to a leaf in the tree represents the left-hand side of a rule.
- Each node stores details of which facts have been matched by the rules at that point in the path. As facts are changed, the new facts are propagated through the Rete from the root node to the leaves, changing the information stored at nodes appropriately.
- This could mean adding a new fact, or changing information about an old fact, or deleting an old fact. In this way, the system only needs to test each new fact against the rules, and only against those rules to which the new fact is relevant, instead of checking each fact against each rule.
- The Rete algorithm depends on the principle that in general, when using forward chaining in expert systems, the values of objects change relatively infrequently, meaning that relatively few changes need to be made to the Rete.
- In such cases, the Rete algorithm can provide a significant improvement in performance over other methods, although it is less efficient in cases where objects are continually changing.
- The basic inference cycle of a production system is match, select and execute as indicated in Fig 8.6. These operations are performed as follows

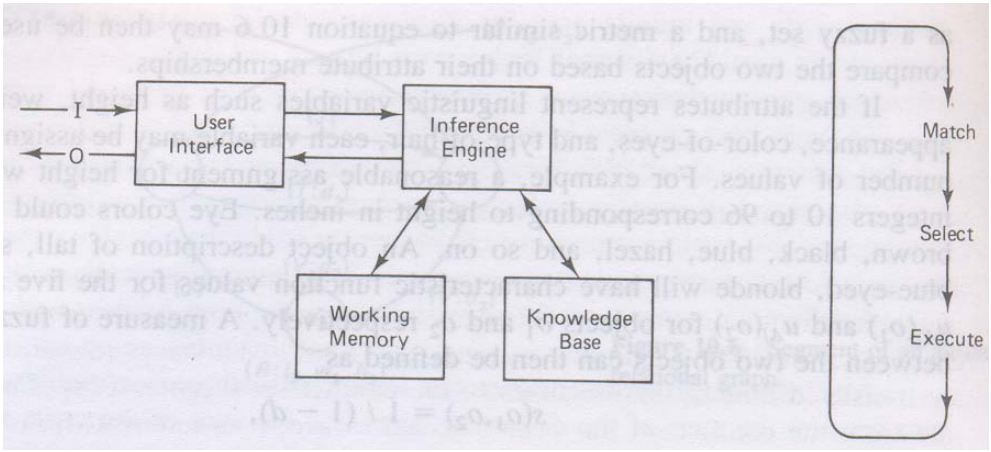


Fig Production system components and basic cycle

- Match
  - During the match portion of the cycle, the conditions in the LHS of the rules in the knowledge base are matched against the contents of working memory to determine which rules have their LHS conditions satisfied with consistent bindings to working memory terms.
  - Rules which are found to be applicable are put in a conflict set
- Select
  - From the conflict set, one of the rules is selected to execute. The selection strategy may depend on recency of useage, specificity of the rule or other criteria
- Execute
  - The rule selected from the conflict set is executed by carrying the action or conclusion part of the rule, the RHS of the rule. This may involve an I/O operation, adding, removing or changing clauses in working memory or simply causing a halt
  - The above cycle is repeated until no rules are put in the conflict set or until a stopping condition is reached
  - The main time saving features of RETE are as follows
    1. in most expert systems, the contents of working memory change very little from cycle to cycle. There is persistence in the data known as temporal redundancy. This makes exhaustive matching on every cycle unnecessary. Instead, by saving match information, it is only necessary to compare working memory changes on each cycle. In RETE,

addition to, removal from, and changes to working memory are translated directly into changes to the conflict set in Fig . Then when a rule from the conflict set has been selected to fire, it is removed from the set and the remaining entries are saved for the next cycle. Consequently, repetitive matching of all rules against working memory is avoided. Furthermore, by indexing rules with the condition terms appearing in their LHS, only those rules which could match. Working memory changes need to be examined. This greatly reduces the number of comparisons required on each cycle

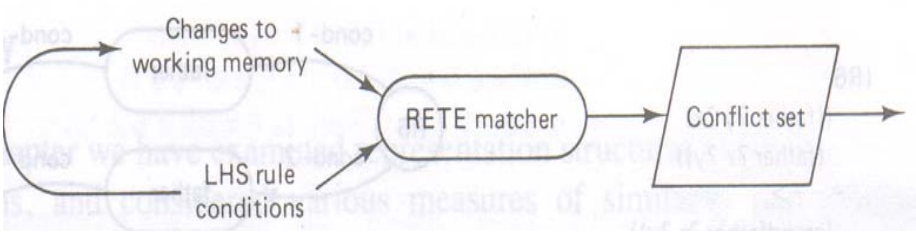


Fig Changes to working memory are mapped to the conflict set

2. Many rules in a knowledge base will have the same conditions occurring in their LHS. This is just another way in which unnecessary matching can arise. Repeating testing of the same conditions in those rules could be avoided by grouping rules which share the same conditions and linking them to their common terms. It would then be possible to perform a single set of tests for all the applicable rules shown in Fig below

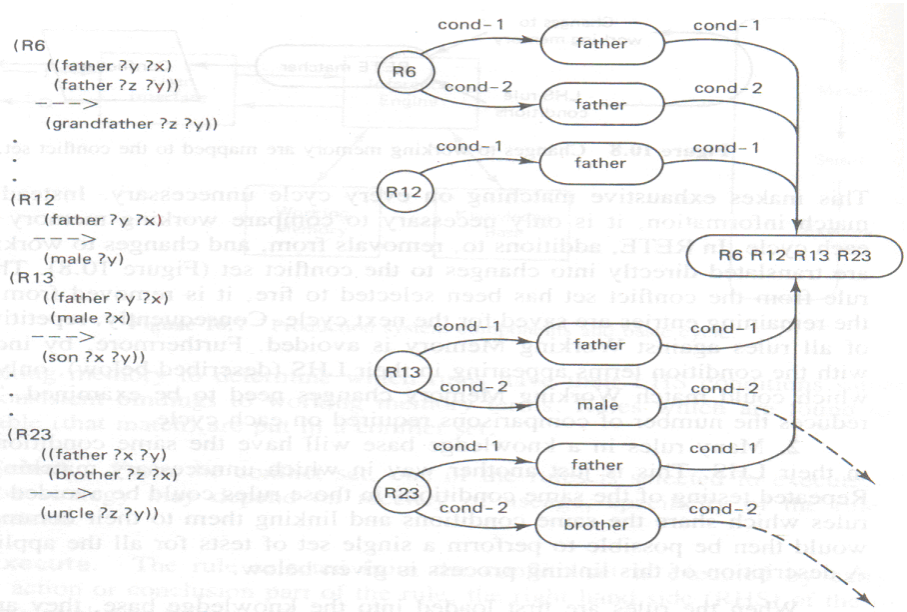


Fig Typical rules and a portion of a compiled network

Knowledge Organization and Management

The advantage of using structured knowledge representation schemes (frames, associative networks, or object-oriented structures) over unstructured ones (rules or FOPL clauses) should be understood and appreciated at this point. Structured schemes group or link small related chunks of knowledge together as a unit. This simplifies the processing operations, since knowledge required for a given task is usually contained within a limited semantic region, which can be accessed as a unit or traced through a few linkages.

But, as suggested earlier, representation is not the only factor, which affects efficient manipulation. A program must first locate and retrieve the appropriate knowledge in an efficient manner whenever it is needed. One of the most direct methods for finding the appropriate knowledge is exhaustive search or the enumerations of all items in memory. This is also one of the least efficient access methods. More efficient retrieval is accomplished through some form of indexing or grouping. We consider some of these processes in the next section where we review traditional access and retrieval methods used in memory organizations. This is followed by a description of less commonly used forms of indexing.

A “smart” expert system can be expected to have thousands or even tens of thousands of rules (or their equivalent) in its KB. A good example is XCON (or RI), an expert system which was developed for the Digital Equipment Corporation to configure their customer’s

computer systems. XCON has a rapidly growing KB, which, at the present time, consists of more than 12,000 production rules. Large numbers of rules are needed in systems like this, which deal with complex reasoning tasks. System configuration becomes very complex when the number of components and corresponding parameters is large (several hundred). If each rule contained above four or five conditions in its antecedent or If part and an exhaustive search was used, as many as 40,000-50,000 tests could be required on each recognition cycle. Clearly, the time required to perform this number of tests is intolerable. Instead, some form of memory management is needed. We saw one way this problem was solved using a form of indexing with the RETE algorithm described in the preceding chapter, More direct memory organization approaches to this problem are considered in this chapter.

We humans live in a dynamic, continually changing environment. To cope with this change, our memories exhibit some rather remarkable properties. We are able to adapt to varied changes in the environment and still improve our performance. This is because our memory system is continuously adapting through a reorganization process. New knowledge is continually being added to our memories, existing knowledge is continually being revised, and less important knowledge is gradually being forgotten. Our memories are continually being reorganized to expand our recall and reasoning abilities. This process leads to improved memory performance throughout most of our lives.

When developing computer memories for intelligent systems, we may gain some useful insight by learning what we can from human memory systems. We would expect computer memory systems to possess some of the same features. For example, human memories tend to be limitless in capacity, and they provide a uniform grade of recall service, independent of the amount of information store. For later use, we have summarized these and other desirable characteristics that we feel an effective computer memory organization system should possess.

1. It should be possible to add and integrate new knowledge in memory as needed without concern for limitations in size.
2. Any organizational scheme chosen should facilitate the remembering process. Thus, it should be possible to locate any stored item of knowledge efficiently from its content alone.

3. The addition of more knowledge to memory should have no adverse effects on the accessibility of items already stored there. Thus, the search time should not increase appreciably with the amount of information stored.
4. The organization scheme should facilitate the recognition of similar items of knowledge. This is essential for reasoning and learning functions. It suggests that existing knowledge be used to determine the location and manner in which new knowledge is integrated into memory.
5. The organization should facilitate the process of consolidating recurrent incidents or episodes and "forgetting" knowledge when it is no longer valid or no longer needed.

These characteristics suggest that memory be organized around conceptual clusters of knowledge. Related clusters should be grouped and stored in close proximity to each other and be linked to similar concepts through associative relations. Access to any given cluster should be possible through either direct or indirect links such as concept pointers indexed by meaning. Index keys with synonomous meanings should provide links to the same knowledge clusters. These notions are illustrated graphically in Fig 9.1 where the clusters represent arbitrary groups closely related knowledge such as objects and their properties or basic conceptual categories. The links connecting the clusters are two-way pointers which provide relational associations between the clusters they connect.

Indexing and retrieval techniques

- **The Frame Problem**

  One tricky aspect of systems that must function in dynamic environments is due to the so called frame problem. This is the problem of knowing what changes have and have not taken place following some action. Some changes will be the direct result of the action. Other changes will be the result of secondary or side effects rather than the result of the action. Foe example, if a robot is cleaning the floors in a house, the location of the floor sweeper changes with the robot even though this is not explicitly stated. Other objects not attached to the robot remain in their original places. The actual changes must somehow be reflected in memory, a fear that requires some ability to

infer. Effective memory organization and management methods must take into account effects caused by the frame problem

The three basic problems related to knowledge organization:

1. classifying and computing indices for input information presented to system
2. access and retrieval of knowledge from mrmory through the use of the computed indices
3. the reorganization of memory structures when necessary to accommodate additions, revisions and forgetting. These functions are depicted in Fig 9.1

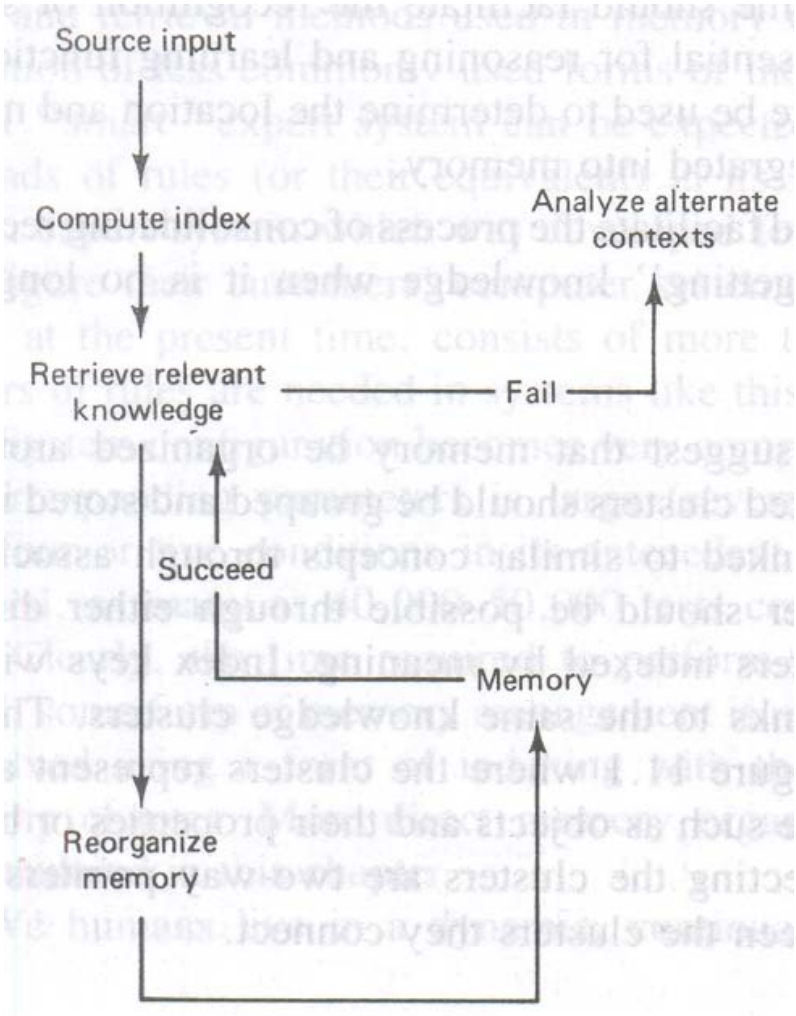


Fig Memory Organization Function

- When a knowledge base is too large to be held in main memory, it must be stored as a file in secondary storage (disk, drum or tape).
- Storage and retrieval of information in secondary memory is then performed through the transfer of equal-size physical blocks consisting of between 256 and 4096 bytes.
- When an item of information is retrieved or stored, at least one complete block must be transferred between main and secondary memory.
- The time required to transfer a block typically ranges between 10ms and 100ms, about the same amount of time required to sequentially searching the whole block for an item.
- Grouping related knowledge together as a unit can help to reduce the number of block transfers, hence the total access time
- An example of effective grouping can be found in some expert system KB organizations
- Grouping together rules which share some of the same conditions and conclusions can reduce block transfer times since such rules are likely to be needed during the same problem solving session
- Collecting rules together by similar conditions or content can help to reduce the number of block transfers required
- **Indexed Organization**
    - While organization by content can help to reduce block transfers, an indexed organization scheme can greatly reduce the time to determine the storage location of an item
    - Indexing is accomplished by organizing the information in some way for easy access
    - One way to index is by segregating knowledge into two or more groups and storing the locations of the knowledge for each group in a smaller index file
    - To build an indexed file, knowledge stored as units is first arranged sequentially by some key value
    - The key can be any chosen fields that uniquely identify the record
    - A second file containing indices for the record locations is created while the sequential knowledge file is being loaded
    - Each physical block in this main file results in one entry in the index file
    - The index file entries are pairs of record key values and block addresses

- The key value is the key of the first record stored in the corresponding block
- To retrieve an item of knowledge from the main file, the index file is searched to find the desired record key and obtain the corresponding block address
- The block is then accessed using this address. Items within the block are then searched sequentially for the desired record
- An indexed file contains a list of the entry pairs (k,b) where the values k are the keys of the first record in each block whose starting address is b
- Fig 9.2 illustrates the process used to locate a record using the key value of 378

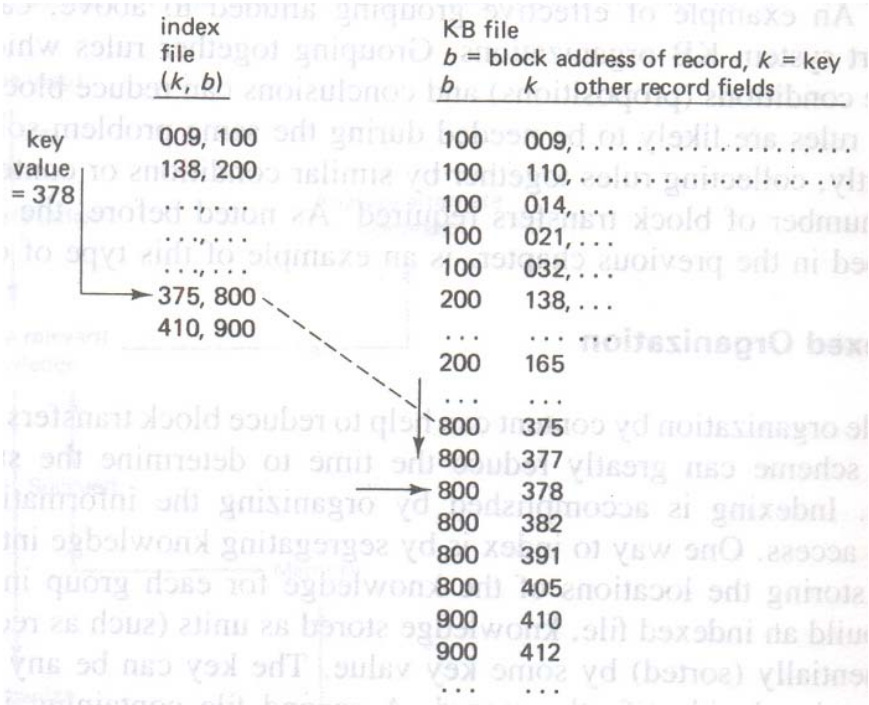


Fig Indexed File Organization

- The largest key value less than 378 (375) gives the block address (800) where the item will be found
- Once the 800 block has been retrieved, it can be searched linearly to locate the record with key value 378. this key could be any alphanumeric string that uniquely identifies a block, since such strings usually have a collation order defined by their code set
- If the index file is large, a binary search can be used to speed up the index file search
- A binary search will significantly reduce the search time over linear search when the number of items is not too small

- When a file contains n records, the average time for a linear search is proportional to n/2 compared to a binary search time on the order of $ln_2(n)$
- Further reductions in search time can be realized using secondary or higher order arranged index files
- In this case the secondary index file would contain key and block address pairs for the primary index file
- Similar indexing would apply for higher order hierarchies where a separate file is used for each level
- Both binary search and hierarchical index file organization may be needed when the KB is a very large file
- Indexing in LISP can be implemented with property lists, A-lists, and/or hash tables. For example, a KB can be partitioned into segments by storing each segment as a list under the property value for that segment
- Each list indexed in this way can be found with the get property function and then searched sequentially or sorted and searched with binary search methods
- A hash-table is a special data structure in LISP which provides a means of rapid access through key hashing

- **Hashed Files**
  - Indexed organizations that permit efficient access are based on the use of a hash function
  - A hash function, h, transforms key values k into integer storage location indices through a simple computation
  - When a maximum number of items C are to stored, the hashed values h(k) will range from 0 to C – 1. Therefore, given any key value k, h(k) should map into one of 0…C – 1
  - An effective hash function can be computed by choosing the largest prime number p less than or equal to C, converting the key value k into an integer k' if necessary, and then using the value k' mod p as the index value h
  - For example, if C is 1000, the largest prime less than C is p = 997. thus, if the record key value is 123456789, the hashed value is h = (k mod 997) = 273
  - When using hashed access, the value of C should be chosen large enough to accommodate the maximum number of categories needed

- The use of the prime number p in the algorithm helps to insure that the resultant indices are somewhat uniformly distributed or hashed throughout the range 0 . . . C – 1
- This type of organization is well suited for groups of items corresponding to C different categories
- When two or more items belong to the same category, they will have the same hashed values. These values are called synonyms
- One way to accommodate collisions is with data structures known as buckets
- A bucket is a linked list of one or more items, where each item is record, block, list or other data strucyure
- The first item in each bucket has an address corresponding to the hashed address
- Fig 9.3 illustrates a form of hashed memory organization which uses buckets to hold all items with the same hashed key value

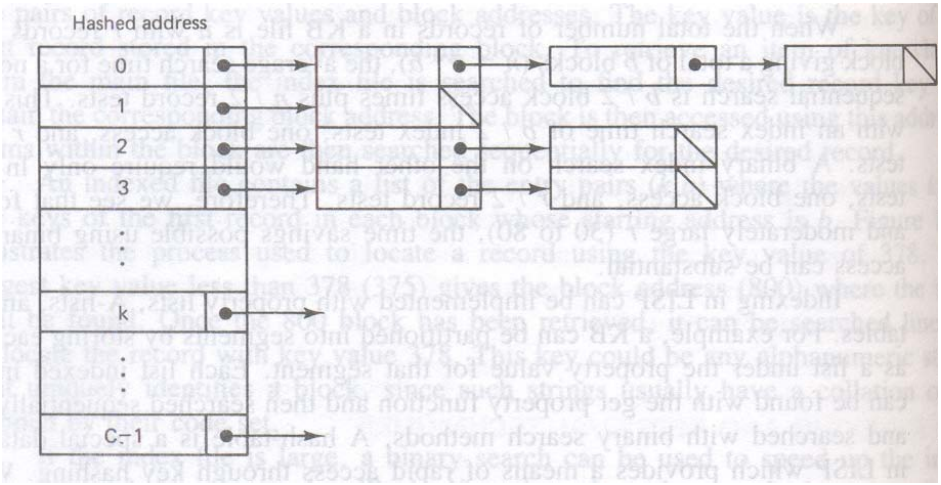


Fig Hashed Memory File organization

- The address of each bucket in this case is the indexed location in an array

- **Conceptual Indexing**
  - A better approach to indexed retrieval is one which makes use of the content or meaning associated with the stored entities rather than some nonmeaningful key value
  - This suggests the use of indices which name and define the entity being retrieved. Thus, if the entity is an object, its name and characteristic attributes would make meaningful indices

- If the entity is an abstract object such as a concept, the name and other defining traits would ne meaningful as indices
- Nodes within the network correspond to different knowledge entities, whereas the links are indices or pointers to the entities
- Links connecting two entities name the association or relationship between them
- The relationship between entities may be defined as a hierarchical one or just through associative links
- As an example of an indexed network, the concept of computer science CS should be accessible directly through the CS name or indirectly through associative links like a university major, a career field, or a type of classroom course
- These notions are illustrated in Fig 9.4

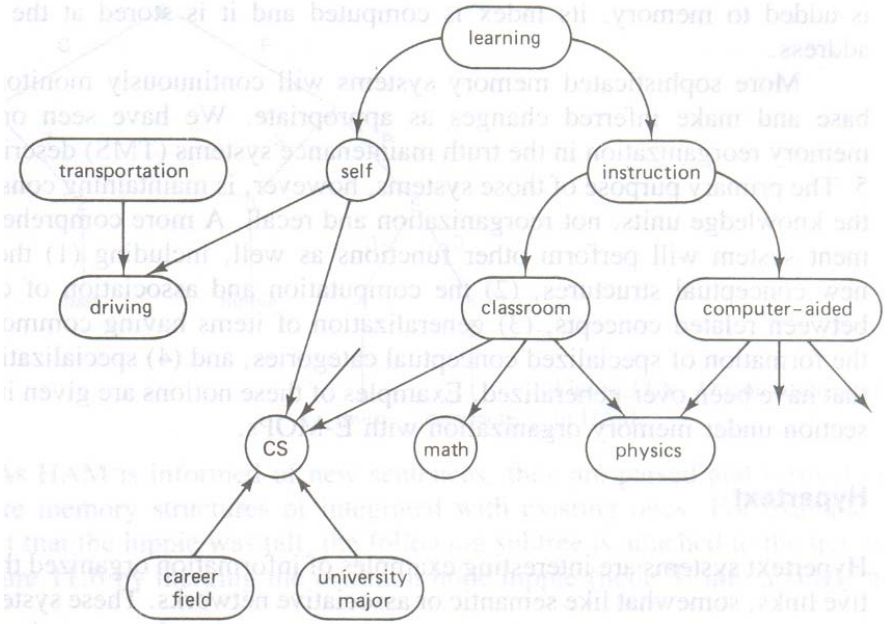


Fig Associative Network Indexing and Organization

- Object attributes can also serve as indices to locate items based on the attribute values
- In this case, the best attribute keys are those which provide the greatest discrimination among objects within the same category

- For example, suppose we wish to organize knowledge by object types. In this case, the choice of attributes should depend on the use intended for the knowledge. Since objects may be classified with an unlimited number of attributes , those attributes which are most discriminable with respect to the concept meaning should be chosen

Integrating knowledge and memory

- Integrating new knowledge in traditional data bases is accomplished by simply adding an item to its key location, deleting an item from a key directed location, or modifying fields of an existing item with specific input information.
- When an item in inventory is replaced with a new one, its description is changed accordingly. When an item is added to memory, its index is computed and it is stored at the corresponding address
- More sophisticated memory systems will continuously monitor a knowledge base and make inferred changes as appropriate
- A more comprehensive management system will perform other functions as well, including the formation of new conceptual structures, the computation and association of casual linkages between related concepts, generalization of items having common features and the formation of specialized conceptual categories and specialization of concepts that have been over generalized
- **Hypertext**
  - Hypertext systems are examples of information organized through associative links, like associative networks
  - These systems are interactive window systems connected to a database through associative links
  - Unlike normal text which is read in linear fashion, hypertext can be browsed in a nonlinear way by moving through a network of information nodes which are linked bidirectionally through associative
  - Users of hypertext systems can wander through the database scanning text and graphics, creating new information nodes and linkages or modify existing ones
  - This approach to documentation use is said to more closely match the cognitive process

- It provides a new approach to information access and organization for authors, researchers and other users of large bodies of information

Memory organization system

- **HAM, a model of memory**
  - One of the earliest computer models of memory was the Human Associative memory (HAM) system developed by John Anderson and Gordon Bower
  - This memory is organized as a network of prepositional binary trees
  - An example of a simple tree which represents the statement "In a park s hippie touched a debutante" is illustrated in Fig 9.5
  - When an informant asserts this statement to HAM, the system parses the sentence and builds a binary tree representation
  - Node in the tree are assigned unique numbers, while links are labeled with the following functions:

C: context for tree fact	P: predicate
e: set membership	R: relation
F: a fact	S: subject
L: a location	T: time
O: object	

  - As HAM is informed of new sentences, they are parsed and formed into new tree-like memory structures or integrated with existing ones
  - For example, to add the fact that the hippie was tall, the following subtree is attached to the tree structure of Fig below by merging the common node hippie (node 3) into a single node

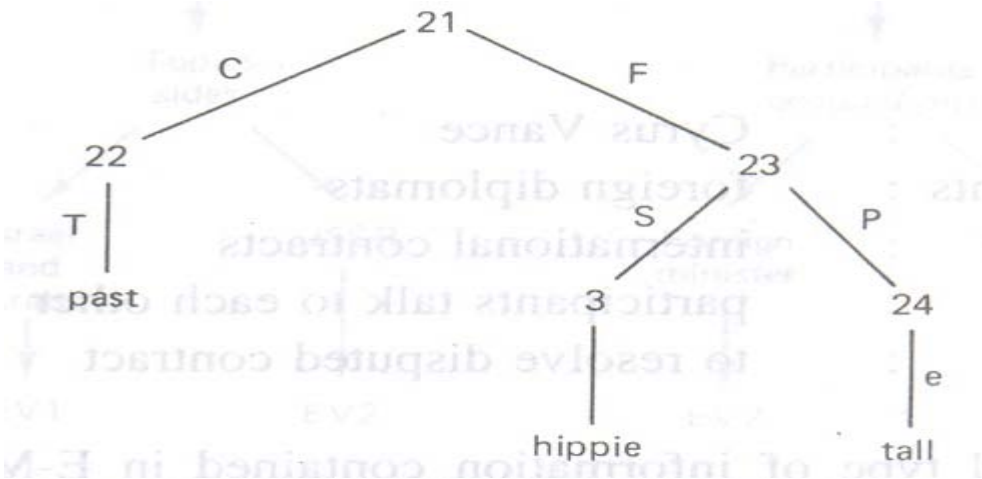


Fig Organization of knowledge in HAM

- When HAM is posed with a query, it is formed into a tree structure called a probe. This structure is then matched against existing memory structures for the best match
- The structure with the closest match is used to formulate an answer to the query
- Matching is accomplished by first locating the leaf nodes in memory that match leaf nodes in the probe
- The corresponding links are then checked to see if they have the same labels and in the same order
- The search process is constrained by searching only node groups that have the same relation links, based on recency of usage
- The search is not exhaustive and nodes accessed infrequently may be forgotten
- Access to nodes in HAM is accomplished through word indexing in LISP

- **Memory Organization with E-MOPs**
  - One system was developed by Janet Kolodner to study problems associated with the retrieval and organization of reconstructive memory, called CYRUS (Computerized Yale Retrieval and Updating System) stores episodes from the lives of former secretaries of state Cyrus Vance and Edmund Muskie
  - The episodes are indexed and stored in long term memory for subsequent use in answering queries posed in English

- The basic memory model in CYRUS is a network consisting of Episodic Memory Organization Packets (E-MOPs)
- Each such E-MOP is a frame-like node structure which contains conceptual information related to different categories of episodic events
- E-MOP are indexed in memory by one or more distinguishing features. For example, there are basic E-MOPs for diplomatic meetings with foreign dignitaries, specialized political conferences, traveling, state dinners as well as other basic events related to diplomatic state functions
- This diplomatic meeting E-MOP called $MEET, contains information which is common to all diplomatic meeting events
- The common information which characterizes such as E-MOP is called its content
- For example, $MEET might contain the following information:
- A second type of information contained in E-MOPs are the indices which index either individual episodes or other E-MOPs which have become specializations of their parent E-MOPs
- A typical $MEET E-MOP which has indices to two particular event meetings EV1 and EV2, is illustrated in Fig 9.6

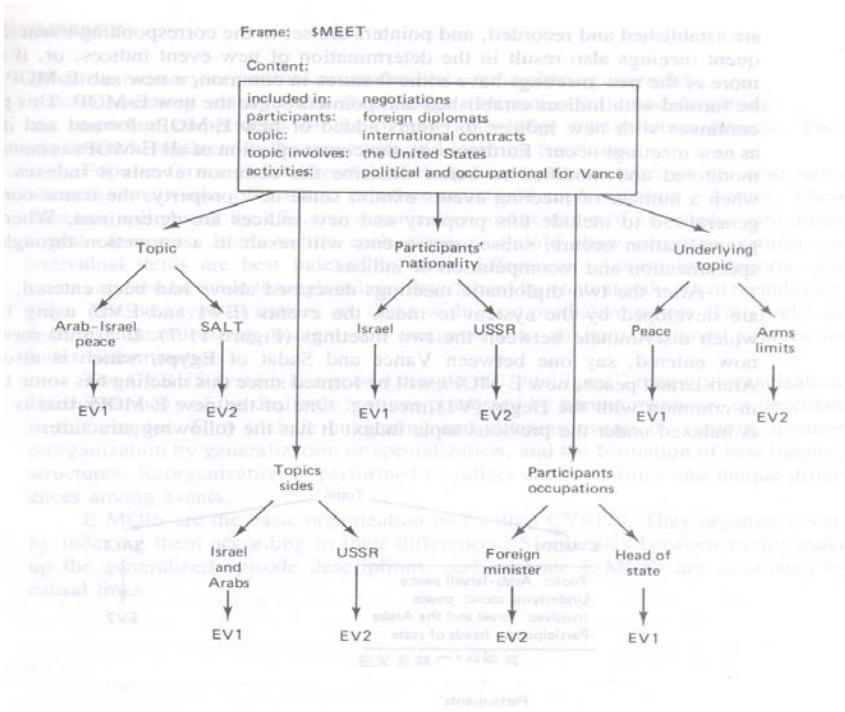


Fig An example of an EMOP with two indexed events EV1 and EV2

- For example, one of the meetings indexed was between Vance and Gromyko of the USSR in which they discussed SALT. This is labeled as event EV1 in the figure. The second meeting was between Vance and Begin of Israel in which they discussed Arab-Israeli peace. This is labeled as event EV2
- Note that each of these events can be accessed through more than one feature (index). For example, EV1 can be located from the $MEET event through a topic value of "Arab-Israel peace," through a participants' nationality value of "Israel," through a participants' occupation value of "head of state," and so on
- As new diplomatic meetings are entered into the system, they are either integrated with the $MEET E-MOP as a separately indexed event or merged with another event to form a new specialized meeting E-MOP.
- When several events belonging to the same MOP category are entered, common event features are used to generalize the E-MOP. This information is collected in the frame contents. Specialization may also be required when over-generalization has occurred. Thus, memory is continually being reorganized as new facts are entered.
- This process prevents the addition of excessive memory entries and much redundancy which would result if every event entered resulted in the addition of a separate event
- Reorganization can also cause forgetting, since originally assigned indices may be changed when new structures are formed
- When this occurs, an item cannot be located, so the system attempts to derive new indices from the context and through other indices by reconstructing related events
- The key issues in this type of the organizations are:
    - ✓ The selection and computation of good indices for new events so that similar events can be located in memory for new event integration
    - ✓ Monitoring and reorganization of memory to accommodate new events as they occur
    - ✓ Access of the correct event information when provided clues for retrieval