[64] W. A. Woods, What's in a link: Foundations for semantic networks. D. G. Bobrow and A. M. Collins, ed., *Representation and Understanding: Studies in Cognitive Science*, p. 35-82, Academic Press, New York, N.Y., 1975.

[48] L. Schubert, R. Goebel, and N. Cercone, The Structure and Organization of a Semantic Network for Comprehension and Inference. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*, New York: Academic Press, 1979.

[49] C. E. Shannon, A mathematical theory of information. *Bell Systems Technical Journal*, 27, 379-423, 623-656.

[50] A. C. Shaw, A formal picture description scheme as a basis for picture processing systems, *Information and Control*, 14,pp.9-51. 1969.

[51] A. C. Shaw, Parsing of graph-representable pictures, *Journal of ACM*, 17(3), pp453-481. 1970.

[52] The MIND System: A Data Structure for Semantic Information Processing. Tech Report No. R-837-PR, The Rand Corporation, 1971.

[53] S. C. Shapiro, Generalized Augmented Transition Network Grammars for Generation from Semantic Networks, *American Journal of Computational Linguistics*, vol 8, no. 1, pp. 12-25, 1982.

[54] S.C. Shapiro and W.J. Rappaport SNEPS Considered as a Fully Intensional Propositional Semantic Network pp. 262-315 in Knowledge Frontier, Nick Cercone and Gordon McCalla (ed.), Springer-Verlag 1983.

[55] T. W. Graham Solomons, *Organic Chemistry*, 2nd Edition, Wiley & Sons (1980).

[56] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine.* Addison Wesley, Reading, Mass. (1984).

[57] J. F. Sowa, Semantic networks. S. C. Shapiro, ed., *Encyclopedia of Artificial Intelligence*, p. 1011-1024, Wiley, New York, 1987.

[58] C. Stanfill and D. Waltz, Toward memory-based reasoning. *Comm. of the ACM*, 29(12), 1213-1228 (December 1986).

[59] E. H. Sussenguth Jr., A graph-theoretic algorithm for matching chemical structures. *J. Chem. Doc.*, 5, 36-43 (1965).

[60] A. Turing, Computing machinery and intelligence. E. A. Feigenbaum and J. Feldman, eds., *Computers and Thought*, New York: McGraw-Hill (1963).

[61] C. S. Wilcox and R. A. Levinson, A self-organized knowledge base for recall, design, and discovery in organic chemistry. *Artificial Intelligence Applications in Chemistry. ACS Symposium Series*, 306 (1986).

[62] P. Willett, The evaluation of an automatically indexed, machine readable chemical reactions file. *Journal of Chemical information and Computer Sciences*, 20, 93-96 (1980).

[63] W. T. Wipke, H. Braun, G. Smith, F. Choplin, W. Sieber, Computer assisted organic synthesis. W. T. Wipke and W. J. Howe, eds, *ACS Symposium Series*, 61, 97-125 (1977).

[32] R. A. Levinson, A self-learning, pattern-oriented, chess program. In *Proceedings of Workshop on New Directions in Game-Tree Search*, T.A. Marsland (ed.), International Computer Chess Association (1989). Also in *International Computer Chess Association Journal*, Edmonton (January 1990).

[33] R. A. Levinson, Pattern Associativity and the Retrieval of Semantic Networks, Technical Report 90-30 available from the Baskin Center of Computer Science, Univ. of California, Santa Cruz(1990).

[34] T. Lipkis, A KL-ONE classifier. J. G. Schmolze and R.J. Brachman, ed., *Proceedings of the 1981 KL-ONE Workshop*, pp 128-145, Cambridge, Mass., 1982. The Proceedings have been published as BBN Report No. 4842 and Fairchild Technical Report No. 618.

[35] E. H. Lum, An implementation of the key reaction approach in computer chemical synthesis. M.S. Thesis, University of California, Santa Cruz (1990).

[36] M. Mansuripur, Introduction to Information Theory, Prentice-Hall (1987).

[37] M. Z. Nagy, S. Kozics, T. Veszpremi, and P. Bruck, Substructure search on very large files using tree structured data bases. *Chemical Structures: The International Language of Chemistry*, Wendy Warr, ed., Springer-Verlag (1988).

[38] J. G. Neal, A Knowledge-Based Approach to Natural Language Understanding, Technical Report 85-06, SUNY Buffalo Department of Computer Science, 1985.

[39] B. Nebel, Terminological reasoning is inherently intractable. *Artificial INtelligence*, 43:235-249, 1990.

[40] H. Poincare, *Science and Hypothesis*, Dover, New York (1952).

[41] A. S. Rao and N. Y. Foo, CONGRES: conceptual graph reasoning system. *Proc. of the 3rd IEEE Conference on Artificial Intelligence Applications*, Orlando, Florida, 87-92 (1987).

[42] L. F. Rau, Exploiting the semantics of conceptual graphs for efficient graph matching. *Proc. of the 3rd Annual Workshop on Conceptual Graphs*, John W. Esch, ed., pp. 3.2.4-1 to 3.2.4-10, (Aug 1988).

[43] F. Reza, *An Introduction to Information Theory*, McGraw-Hill (1961).

[44] B. Riff, Searching a partially-ordered knowledge base of complex objects. Master's Thesis, University of California at Santa Cruz (1988).

[45] R. J. Schalkoff, *Digital Image Processing and Computer Vision*, Wiley, p. 286. 1989.

[46] D. C. Schmidt and L. E. Druffel, A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *J. Assoc. Comput. Mach.*, 23, 433-445 (1976).

[47] J. G. Schmolze and T. A. Lipkis, Classification in the KL-ONE knowledge representation system. In *Proc. IJCAI-83*, 330-332 (1983).

[16] C. J. Fillmore, The Case for Case, in *Universals in Linguistic Theory*, Bach, E. and Harris, R. T., editors, Holt Rinehart and Winston, New York, 1968

[17] N. V. Findler, ed., *Associative Networks: Representation and Use of Knowledge by Computers*, Academic Press, New York, N.Y., 1979.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman (1979).

[19] B. J. Garner and E. Tsui, A self-organizing dictionary for conceptual structures. *Proceedings of Applications of Artificial Intelligence V*, John F. Gilmore, ed., SPIE Proc. 784, 356-363 May 18-20 (1987).

[20] F. Hayes-Roth and D. J. Mostow, An automatically compilable network recognition network for structured patterns. *Proc. IJCAI-75*, 246-251 (1975).

[21] G. Hendrix, Encoding knowledge in partitioned networks. *Associative Networks: The Representation and Use of Knowledge by Machine*, Academic Press, New York (1979).

[22] G. E. Hinton and J.A. Anderson, eds., *Parallel Models of Associative Memory*, L. Erlbaum (1989).

[23] E. Horowitz and S. Sahni, *Fundamentals of Data Structures*, Computer Science Press (1976).

[24] G. J. Klir, *Architecture of Systems Problem-Solving*, Plenum Press, New York (1985).

[25] J. L. Kolodner, *Retrieval and Organizational Strategies in Conceptual Memory: a Computer Model,* L. Erlbaum and Associates (1984).

[26] P. Langley and J. Wogulis, Improving efficiency by learning intermediate concepts. *Proc. IJCAI-89.*

[27] R. A. Levinson, A self-organizing pattern retrieval system for graphs. *In Proc. AAAI-84* (1984).

[28] R. A. Levinson, A self-organizing retrieval system for graphs. PhD Dissertation, Univ. of Texas at Austin (1985).

[29] R. A. Levinson, D. Helman, E. Oswalt, Intelligent signal analysis and recognition. In *Proc. 1st Int'l Conference on Industrial and Engineering Applications of Artificial Intelligence, ACM* (1988).

[30] R. A. Levinson, A self-organizing pattern retrieval system and its applications. Technical Report UCSC-CRL-89-21, University of California at Santa Cruz (1989). (To be published in *International Journal of Intelligent Systems.*)

[31] R. A. Levinson, Pattern formation, associative recall and search: a proposal. Technical Report UCSC-CRL-89-22, University of California at Santa Cruz (1989).

References

[1] G. W. Adamson, J. Cowell, M. F. Lynch, H. W. McLure, W. G. Town, M. A. Yapp. Strategic Considerations in the design of a screening system for substructure searches of chemical structure files. *J. Chem. Doc.*, 13, 153-157 (1973).

[2] A. Aho, A. Hopcroft, J. Ullman, *Data Structures and Algorithms*, Addison-Wesley (1983).

[3] F. H. Allen, S. Bellard, et al, The Cambridge crystallographic data centre: computer based–search, retrieval, analysis and display of information. *Acta Crystallogr.*, B35, 2331-2330.

[4] J. M. Barnard, Problems of substructure search and their solution. *Chemical Structures: The International Language of Chemistry*, Wendy Warr, ed., Springer-Verlag (1988).

[5] D. J. Bawden, Computerized chemical structure-handling techniques in structure-activity studies and molecular property prediction. *Chem. Inf. Comp. Sci.*, 23, 14-22 (1983).

[6] N. J. Belkin and C. J. van Rijsbergen, ed. Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval, ACM Press, 1989.

[7] C. Berge, *Graphs and Hypergraphs*, 2nd Edition, North-Holland, 1976.

[8] K. V. S. Bhat, Refined vertex codes and vertex partitioning methodology for graph isomorphism testing. *IEEE Trans. on Systems, Man, and Cybernetics*, 10(10), 610-615 (Oct 1980).

[9] R. Brachman and J. Schmolze, An Overview of the KL-One Knowledge Representation System, *Cognitive Science*, vol 9, no. 2, pp. 171-216, 1985.

[10] R. Brachman and H. J. Levesque, ed., *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, California, 1985.

[11] N. Cercone, On Representational Aspects of VLSI-CADT Systems Hassan Reghbati and pp. 451-470 in Knowledge Frontier, Nick Cercone and Gordon McCalla (ed.), Springer-Verlag 1983.

[12] D. G. Corneil and D. G. Kirkpatrick, A theoretical analysis of various heuristics for the graph isomorphism problem. *Siam J. Computing*, 9:2 (May 1980).

[13] P.G. Dittmar, N. A. Farmer, W. Fisanick, R. C. Haines, J. Mockus. The CAS online search system—general system design and selection, generation of search screens. *Journal of Chemical Information and Computer Sciences*, 23, 93-102 (1983).

[14] G. Ellis, Deterministic all-solutions retrieval from the generalization hierarchy. Technical Report, Dept. of Computer Science, University of Queensland, Australia (August 1989).

[15] G. Ellis, Efficient retrieval from the generalization hierarchy. *Proceedings of the 1st Australian Knowledge Engineering Program*, presented at the *AI and Creativity Workshop '89*, Melbourne, March 14-15th, 1989. Also available as Technical Report No. 114, Dept. of Computer Science, University of Queensland, Australia (May 1989).

thanks to Richard Snyder who has produced the pictures for the article and provided critical comments throughout the final revision.

In the chess application positions are represented as graphs based on attacks and defends relationships between pieces and squares. A number of the generalization nodes correspond to patterns well-known to good chess players such as fianchettoed bishop, various kingside castled positions, doubled rooks and various pawn formations. Other generalization nodes are not that well-known but may also have domain validity.

It is also worth taking a closer look at the meaning of node descriptors that have been developed in Method IV for conceptual graphs. They describe neighborhoods of concept and relation nodes. Since we ignore arc direction in calculating distances, then distance-1 neighborhoods give all relations connected to a given concept node or all concepts connected to a given relation node. Clearly, these are relevant notions. If we move to distance-2 neighborhoods of concept nodes we get the full picture of all relations that involve that concept (for example, node descriptor 5 in G_3 represents "an apple pie was eaten"). If a node was calculated through the iteration process then the information contained in a node descriptor becomes quite interesting(providing a meta-level view of the graph): A distance-2 neighborhood of a concept node gives the relations of that node to immediate adjacent concept neighborhoods!

It makes sense that pieces of semantic networks that occur commonly are often useful when treated as units (chunks) in the domains in which they arise. That such units also improve retrieval efficiency is a less obvious fact, but worth studying, e.g. by cognitive scientists, knowledge engineers or philosophers of science. Langley and Wogulis [26] have recently supplied empirical evidence that the introduction of intermediate concepts can improve the efficiency of learning algorithms. Giving names to these pieces of semantic networks and processing these pieces as units is an application of "abstraction". We believe that the proper use of abstraction in semantic network retrieval systems may be as important as the exploitation of pattern-associativity. For example, by representing propositional nodes(see above) as individual nodes with special labels (and not graphs) much redundant computation may be avoided. We are currently exploring this avenue.

9 Concluding Remarks

In summary, we have presented four methods for the associative retrieval of semantic networks (while focusing mainly on conceptual graphs) and illustrated the ideas from which these methods have been derived. In particular we presented the principle of pattern-associativity and how it is exploited increasingly in the better methods. The first three methods have each been used in semantic network systems. The advantages of Method III over Methods I and II are argued both formally and informally. Our implementation of Method IV is not yet complete but the results in chemical systems and our experimental results make this a compelling research direction. At the very least we have shown the extent to which the principle of pattern-associativity can be taken. We shall not be surprised if there are yet one or more levels to go!

10 Acknowledgements

The paper has benefited from the constructive suggestions of three anonymous reviewers of the article and the editor. Gerard Ellis provided useful references and much encouragement. Jean McKnight formatted several drafts and Max Copperman helped with the introduction. Many

be extended to first compare the graphs (as in variation 1) while ignoring edges attached to partitions and then recursively working "inside-out" by first matching the most deeply-nested propositional nodes (graphs) and working out from there. A propositional node in the query graph can match any proposition node in a database graph that it is more-general-than.

3. *Logical operators such as negation or quantification are attached directly to a proposition node.* In this case matching can take place as in variation 2(viewing the operator as part of the label for the proposition node.) except that to insure consistency in matching a canonical form such as CNF (Conjunctive Normal Form) should be used for all database and query graphs.

As Method IV depends on refinement testing it can be applied to all network formalisms that use subgraph-isomorphism as the basis for subsumption testing. Here we applied Method IV to conceptual graphs, but it can be extended in a straightforward way to graphs with labelled or undirected edges [33] to handle other semantic network formalisms. (It is not obvious how to extend refinement and Method IV to partitioned-networks, however, but one possibility is to add to the O-type dus a "partition-distance" field that gives the minimal number of partitions that must be passed through from one node to the next and treatingthe box surrounding a partition as a node itself.) **But more important than the particular version of Method IV that has been presented is the application of pattern associativity that produced the method. Once this is well-understood it may be possible to derive similar systems that do not depend on isomorphism testing, but take advantage of the commonalities of subsumption tests, whatever the form.**

8 Domain Validity of Generalization Graphs

In each of Methods II-IV additional objects are added to the system to improve efficiency through indexing. In Method II screens are used. In Method III generalization graphs are created through self-organization. Finally, in Section IV we see that node descriptors (environments) are stored. In Methods II and III these generalization graphs are used because they are common to many graphs. The most useful node environments (Method IV) are also those that occur commonly. Indeed, generalization graphs are the most tangible manifestation of the pattern-associativity principle. One then wonders whether these graphs also have semantic validity in the domain applications in which they arise?

The domain validity of generalization graphs is one of the interesting things that has come out of our applications of semantic memory to chemistry [27,28,30,35] and chess [31,32]. The generalization graphs created through self-organization from chemical graphs have corresponded remarkably well to what chemists call "functional groups":

> ne great advantage of the structural theory is that it enables us to classify the vast number of organic compounds into a relatively small number of families based on their structures. The molecules of compounds in a particular family are characterized by the presence of a certain arrangement of atoms called a functional group. A functional group is the part of a molecule where most of its chemical reactions occur. It is the part that effectively determines the compound's chemical properties (and many of its physical properties as well). [55]

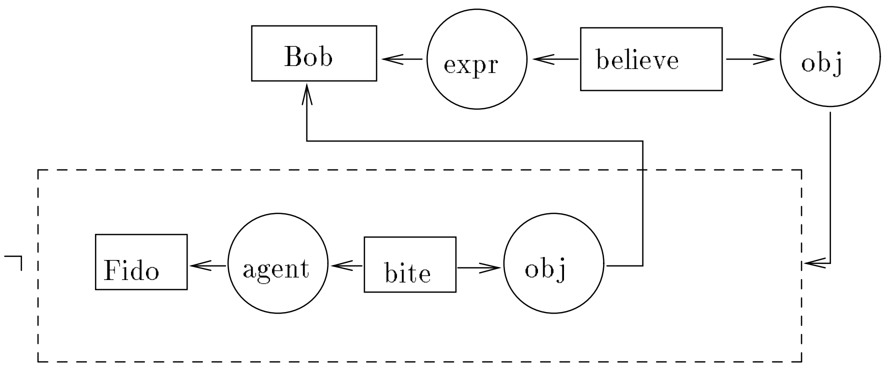


Figure 16: Example of a propositional node with a logical operator.

“Bob believes that Fido did not bite him.” (adapted from [56] p. 1015)

7 Extending to other semantic network formalisms

The previous development has been based on the use of conceptual graphs. An important question, then, is to what extent these techniques can be applied to other semantic network representation schemes? It is important to realize that Methods I-III assume the availability of an “oracle” (for conceptual graphs, a subgraph-isomorphism test) that could determine the subsumption relationship between concepts (graphs). The type hierarchies are used by the oracle and not by the retrieval algorithm itself. Thus, **the exciting conclusion is that Methods I-III can be used with any semantic network formalism as long as a subsumption operator is provided**. This was illustrated above when in Method IV the algorithms from Method III were used to insert node descriptors into the node descriptor hierarchy (using a comparison function other than subgraph-isomorphism). Although we have made no suggestions on how to improve the tractability of certain subsumption tests [39], Method III can be used to reduce the number of such tests that are required.

Some semantic network formalisms go beyond the general labelled graph framework by allowing “partitions”, “spaces”, or “proposition nodes” [21,52,48,57]. These are collections of nodes (and the arcs between them) in a semantic network. We will briefly look at the three main variations of these and how subgraph-isomorphism tests can be extended to handle them (variations 2 and 3 are illustrated in Figure 16):

1. *There are no arcs involving the partition box itself, but there may be arcs to and from nodes within the partition to nodes outside the partition.* By viewing the sets of nodes that make up the partition as “hyperedges” [7] the conditions for subsumption (and the corresponding test) can be extended in a straightforward manner. The Adjacency-Preservation condition (Section 2) is extended to include the adjacencies defined by hyperedges simply by changing the word “pair” to “set”: For a set of nodes that are adjacent in one structure, the corresponding set of nodes must be adjacent in the other structure, and further, the direction (if any) of the edge between the nodes must also correspond. This condition can be checked at exactly the same place it is normally done in a subgraph-isomorphism algorithm.

2. *The box making up the partition and not just the nodes within the partition may have incoming and outgoing edges.* For such proposition nodes the subgraph-isomorphism algorithm should

Graph	pred-count	succ-count	conclusion
G_1	3	1	generalization
G_2	0	5	specialization
G_3	1	0	incomparable
G_4	1	5	specialization
G_5	0	1	incomparable

Table 3: Results of Method IV retrieval example.

2.1 Increase pred-count by 1 for each database graph that has a characteristic descriptor that is a predecessor of q.

2.3 Increase succ-count by 1 for each database graph that has a characteristic descriptor that is a successor of q.

3. Return all graphs that have a pred-count equal to their number of nodes as predecessors of Q.

4. Return all graphs that have succ-count equal to the number of nodes in Q as successors of Q.

5. Graphs that are both predecessors and successors are exact matches.

END

Close matches are easily found using this system. This is done by using the succ-counts of the database graphs. Those matching a higher percentage of the query nodes are returned as close matches. Much success has been achieved using this simple technique in the chemistry domains. Rau [42] suggests a similar approach for conceptual graphs but from within a Method II system. Once these close matches are identified as a maximal common subgraph algorithm can be used to return the exact commonality if necessary.

Example 6.4:
Let's explore how our sample query would be processed using the node descriptor hierarchy (Figure 15). Each node descriptor for the query graph is inserted: Q_1 is found to be identical to 1, Q_2 would be inserted on the arc between 2 and 5, Q_3 on the arc between 3 and 6, Q_4 is identical to 7, and Q_5 is identical to 8. The pred-count and succ-count of each database graph are updated giving the results in Table 6.3.

This hierarchical node descriptor method may be used with Design Method III by placing a partial order over the database graphs as before. The hierarchical node descriptor method does the filter using the description hierarchy, then subgraph-isomorphism tests are done to ensure match (these should go fast since the graphs almost certainly match), but as in Phase II of Design Method III some expensive matches can be inferred for free using the hierarchy. Using Design Method IV the HTSS researchers have reported that retrieval time grows sublinearly: moving from 150,000 to 1,200,000 structures resulted in an increase in retrieval time of 50 percent [37]. Design Method IV is even more promising when we consider that each of the node descriptor searches down the hierarchy may be done in parallel.

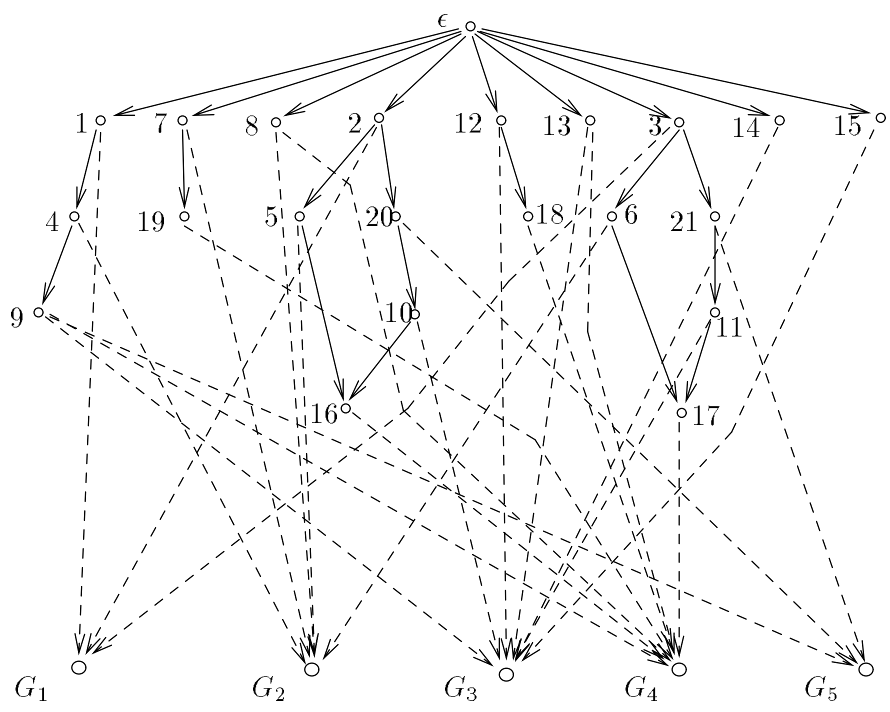


Figure 15: A node descriptor hierarchy for G_1 through G_5.

Ideally, any graph that has some node-descriptor in each predecessor set qualifies as a generalization, and any graph that has some node descriptor in each successor set is a specialization. However, theoretically things are more complex than this because a node may be inadvertently counted multiple times. For example, suppose graph X has a single node that is more-general-than all nodes in the query Q. This should not imply that X is more-general-than Q, but this is what the method would tell us. How can these difficulties be handled? One possibility is to keep track of exactly which nodes in the query graph map to exactly which nodes in the database graph. And then for each graph that remains a candidate as a specialization or generalization determine whether a 1-1 mapping is possible. Finding such a mapping is exactly the bipartite matching problem, unfortunately the complexity of solving this problem is $O(ne)$ where n are the number of nodes and e the number of edges in the matching problem. Relying on such an algorithm nearly defeats the purpose of eliminating the subgraph-isomorphism tests. Fortunately, the multiple node matches are unlikely to occur on conceptual graphs when a graph match is not present. The

resulting algorithm:

BEGIN(* Retrieval in Method IV *) With each database graph maintain two fields pred-count and succ-count that are initialized to 0.

1. Node descriptors are calculated for the query graph Q.
2. For each node descriptor q:
    - 2.1 Find q's place in the hierarchy using Phase I and Phase II of Method III but with the node-descriptor comparison test.

Node Descriptor	Node Number
G_1:	
{(S,person,0)(O,agent,-1)(O,eat,2)}	1
{(S,agent,0)(O,person,1)(O,eat,-1)}	2
{(S,eat,0)(O,person,2)(O,agent,1)}	3
G_2:	
{(S,girl,0)(O,agent,-1)(O,eat,2)}	4
{(S,agent,0)(O,girl,1)(O,eat,-1)(O,manr,2)}	5
{(S,eat,0)(O,girl,2)(O,agent,1)(O,manr,1)(O,quickly,2)}	6
{(S,manr,0)(O,agent,2)(O,eat,-1)(O,quickly,1)}	7
{(S,quickly,0)(O,manr,-1)(O,eat,2)}	8
G_3:	
{(S,Sue,0)(O,agent,-1)(O,eat,2)}	9
{(S,agent,0)(O,Sue,1)(O,eat,-1)(O,obj,2)}	10
{(S,eat,0)(O,Sue,2)(O,agent,1)(O,obj,1)(O,pie,2)}	11
{(S,obj,0)(O,agent,2)(O,eat,-1)(O,pie,1)(O,cont,2)}	12
{(S,pie,0)(O,eat,2)(O,obj,-1)(O,cont,1)(O,apples,2)}	13
{(S,cont,0)(O,obj,2)(O,pie,-1)(O,apples,1)}	14
{(S,apples,0)(O,pie,2)(O,cont,-1)}	15
G_4:	
{(S,Sue,0)(O,agent,-1)(O,eat,2)}	9
{(S,agent,0)(O,Sue,1)(O,eat,-1)(O,obj,2)(O,manr,2)}	16
{(S,eat,0)(O,Sue,2)(O,agent,1)(O,obj,1)(O,pie,2)(O,manr,1)(O,quickly,2)}	17
{(S,obj,0)(O,agent,2)(O,eat,-1)(O,pie,1)(O,cont,2)(O,manr,2)}	18
{(S,pie,0)(O,eat,2)(O,obj,-1)(O,cont,1)(O,apples,2)}	13
{(S,manr,0)(O,agent,2)(O,eat,-1)(O,obj,2)(O,quickly,1)}	19
{(S,quickly,0)(O,eat,2)(O,manr,-1)}	8
G_5:	
{(S,Sue,0)(O,agent,-1)(O,eat,2)}	9
{(S,agent,0)(O,Sue,1)(O,eat,-1)}	20
{(S,eat,0)(O,Sue,2)(O,agent,1)}	21
Q:	
{(S,person,0)(O,agent,-1)(O,eat,2)}	q_1
{(S,agent,0)(O,person,1)(O,eat,-1)(O,manr,2)}	q_2
{(S,eat,0)(O,person,2)(O,agent,1)(O,manr,1)(O,quickly,2)}	q_3
{(S,manr,0)(O,agent,2)(O,eat,-1)(O,quickly,1)}	q_4
{(S,quickly,0)(O,manr,-1)(O,eat,2)}	q_5

Table 2: Node descriptors for G_1 through G_5 and Q.

Now that we know how to build node descriptors for each graph we can move to building the desired database of node descriptors (partial order by more-general-than) and associated graphs. Initially, the database is made up of only a single node descriptor ϵ that is defined to be more general than any other node descriptor and thus will remain at the top of the hierarchy. All node descriptors to be stored in the system are given a unique number and are to occur only once in the hierarchy. Each node descriptor points directly to the graphs from which it has been derived.

Comparing node descriptors: Node descriptor q is to be considered more-general-than node descriptor r iff there is a 1-1 mapping from dus in q to matching dus in r. (The idea is that q is more-general-than r if they could bind in a subgraph-isomorphism test). Two S-type dus (S,v1,d1) and (S,v2,d2) match iff v1 is more-general-than v2 in the type hierarchy and d1$\geq$d2. Two O-type dus (O,l1,d1) and (O,l2,d2) match iff l1 is more-general-than l2 in the type hierarchy (if labels) or if l1 is more -general-than l2 in the node-descriptor hierarchy (if node descriptors).

Note that it is only when comparing dus that the type hierarchies need be consulted. Again we recommend that the type hierarchies be compiled into tables of pairs for faster processing. Similarly we recommend that the numbers from the descriptor hierarchy also be compiled into pairs where one descriptor is more-general-than another.

In general, determining whether one node descriptor is more-general-than another is the bipartite matching problem and thus is $O(ne)$ in the worst case where n is the number of nodes in the descriptor and e is the number of possible du matches. But in practice finding the matching is usually trivial.

```
BEGIN (* Insert Graph G into the method IV database *)
    1. Build node descriptors for each node in G as described above.
    2. For each label field of O-type dus that is itself a node descriptor call this routine recursively to
       insert the node descriptor, replace the label field in the O-type du with the number returned
       for the descriptor.
    3. Insert each descriptor into the hierarchy where it belongs(if it does not already exist). This
       can be done using Phase I and Phase II of Method III except the graph isomorphism test is
       replaced with the much simpler node descriptor comparison described above.
    4. Add pointers from the node descriptors to G.
    5. Return a unique number for the node descriptor or an existing one if the descriptor had
       previously been inserted.
END
```

For example Figure 15 shows a node descriptor hierarchy where graphs G_1 through G_5 and their nodes (Table 2) have been inserted.

The top-level node descriptors that come from a particular graph will be called "characteristic descriptors" of that graph in the following algorithm. It is only the characteristic descriptors that point to a given graph, though other descriptors may have paths through a characteristic descriptor to the given graph. Retrieval takes place by first generating all node descriptors in the query graph and then finding predecessors and successors of each node descriptor in the descriptor hierarchy.

O-type du can be made more specific by replacing it with the current node descriptor of the node from which the label has been derived.

Example 6.2:
The Node Descriptor for Node 1 in Example 6.1 can be made more specific by substituting in the node descriptors for nodes 2 and 3 giving {(S, bread, 0), (O, {(S, betw, 0), (O, bread, -1), (O, jam, 1), (O, betw, 2)}, 1), (O, {(S, jam, 0), (O, bread, 2), (O, betw, -1), (O, betw, 1), (O, bread, 2)}, 2)}.

6.3 Method IV: Design Organization

By precompiling the *node descriptors* for each database graph the node descriptor comparisons based on the query graph can in essence be done in parallel since through pattern-associativity shared node descriptors in each database structure are processed only once.

Every node in every database graph(and every query graph) is to be represented as a node descriptor as above. But how specific should these descriptors be made? The following algorithm gives the necessary details: (Two nodes are in the same equivalence class if they have the same node descriptor. Thus, as descriptors become more specific, equivalence classes may become smaller and more numerous.)

```
BEGIN(* Generate Node Descriptors *)
  1. Represent each node as a set of dus as described above. Label fields in O-type dus are the
     actual label from the node pointed to (a pointer to this node should be temporarily stored in
     the du).
  2. REPEAT
       2.1 Record current node descriptors and equivalence classes.
       2.2 Replace all labels in the O-type dus with the new node descriptor for the associated
           node. (This need only be done for nodes that are not currently in singleton equivalence
           classes)
     UNTIL equivalence classes of nodes have not changed from the previous iteration.
  3. Return node descriptors from the previous iteration.
END
```

Except for very unusual graphs, the resulting node descriptors are such that two nodes with the same descriptor are truly symmetric in the given graph. For many graphs only one iteration is required.

Example 6.3:
The equivalence classes after the first iteration (Example 6.1) are: 1,5,2,3,4. Thus nodes 1 and 5 are the only nodes that are expanded(refined) on the next iteration. We gave the expansion for node 1 above. The expansion for node 5 is {(S, bread, 0), (O, {(S, betw, 0), (O, betw, 2), (O, jam, -1), (O, bread, -1)}, 1), (O, {(S, jam, 0), (O, bread, 2), (O, betw, -1), (O, betw, 1), (O, bread, 2)}, 2)}. Since the descriptions for nodes 1 and 5 no longer match, each node is in its own equivalence class and hence the iteration process stops with each node being described with its final descriptor.

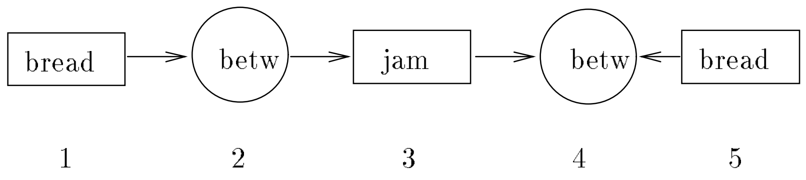


Figure 14: Used in examples 6.1, 6.2 and 6.3.

"betw" is the triadic relation "between". It is not fully specified in this graph (see [56], page 72).

Node	Node Descriptor
1	{(S,bread,0),(O,betw,1),(O,jam,2)}
2	{(S,betw,0),(O,bread,-1),(O,jam,1),(O,betw,2)}
3	{(S,jam,0),(O,bread,2),(O,betw,-1),(O,betw,1),(O,bread,2)}
4	{(S,betw,0),(O,betw,2),(O,jam,-1),(O,bread,-1)}
5	{(S,bread,0),(O,betw,1),(O,jam,2)}

Table 1: Initial node descriptors for Figure 14

.

It is the calculation of these "self-loops" that allows the system to correctly handle the anomalous chemical queries mentioned above.

2. There is one du for each other node in the graph: (O,v,d)
   - O = other
   - v = node label
   - d = Shortest path distance to node *ignoring edge direction* and thus viewing all edges as bidirectional. This will always be a positive integer, except for nodes at distance 1, where 1 stands for the existence of a forward edge (and possibly a backward edge) to that node and -1 for no forward edge (only a backward edge).

Such descriptions can be calculated for all nodes using a single call to an all-pairs shortest path algorithm[2]. The descriptors can be read directly from the resulting distance matrix(that gives the shortest distance between all pairs of nodes). The distance matrix heuristic method has performed very well in practice [46], though Corneil [12] shows that it (and many other good practical heuristics) are ineffective on a theoretical (unrealistic) subclass of graphs known as c-subgraph regular. We would be very surprised indeed to find a realistic example of conceptual graphs in which these descriptions are not sufficient. In the counter examples from the graph theory literature almost all node and edge labels are set to be the same. In fact, for most applications we suggest not storing atoms that reflect distances of more than 2 or 3. Not only are the larger distances unneccesary but they greatly increase the storage and matching requirements.

Example 6.1:
We calculate node descriptors for Figure 14 (before further iterations as below), reflecting distances up to length 2. The results appear in Table 1.

But how can node descriptors be made more specific (as is required in successive iterations of refinement)? **The key, and somewhat profound, notion is that the node label field of an**

In refinement, a different approach is taken based directly on applying the syntactic property-inheritance principle (see Section 4) to the nodes of graph Q: What is true of a node q in Q must be true "all the more so" of a node in r that it is bound to. Thus, in general, the more specific we make the description of a node in Q the fewer possible bindings with nodes in R need be considered. If the descriptions are made extremely specific(more "refined") then the likelihood of eliminating all possible bindings is increased, and if some possible bindings remain, we can be nearly, (but not absolutely) certain that they are correct. If after making the descriptions highly-specific, all nodes in Q still have non-empty binding lists we can be nearly (but not absolutely) certain that Q is a generalization of R.

Success can not be guaranteed without a node-by-node comparison as in backtrack search, but in practice in the chemical domain this has never proven necessary. The technique has never erred in real systems. However, some hypothetical anomalous queries have been presented in the chemical literature [4]. The description technique presented below also handles these. We expect these techniques to work at least as well for conceptual graphs due to the increased label variety in these networks that reduces the number of bindings that need be considered.

> It has been shown empirically that a refinement algorithm usually solves isomorphism problems on random graphs in $O(n^2)$ operations where n is the number of nodes in each graph [12].

The refinement method lends itself readily to parallelism: on each iteration the description of a node may be updated independently of other nodes to be updated on that iteration.

6.2 Method IV: Overview and Node Description Scheme

Informally: Method IV maintains a hierarchical partial ordering(by more-general-than) of node descriptions such that descending the hierarchy(to more specific descriptions) is akin to iterating in a refinement test. At the bottom of the hierarchy are stored the original database graphs(assume an arbitary flat ordering of these for now). There are pointers to each database graph G from the most specific descriptions in the hierarchy that correspond to some node in G. Thus there are paths from each description to database graphs that have a node that satisfy that description. By finding the place of the description of each query graph node in the hierarchy it is possible to find all specializations and generalizations of the query. Specializations are all those graphs that can be reached by all query nodes and generalizations are all those graphs that have paths from each of their nodes to query nodes (if they were to be inserted in the hierarchy). ¡picture¿

Obviously, the key to Method IV processing is in the node description method. There are many such methods used by refinement algorithms. The following information is usually stored, though any individual system uses only a subset of this information. Each node (concept) is represented as a set of *description units("dus")* called a *node descriptor*:

1. Each node descriptor has one du of the form: (S,v,d)
   - S = self
   - n = node label
   - d = length of the shortest non-trivial cycle node is on, 0 if none. (since conceptual graphs are bipartite, d must be even)

1. Applying a maximal common subgraph algorithm to the answers to close match queries.
2. Taking a query graph and stripping off parts of it until it is a subgraph of many others.
3. Applying a "join" operation(take two graphs and combine them to form a more specific graph that retains shared structure from the original graphs) to general graphs in the ordering.

Some systems require that generalization nodes be formed and stored between any two database graphs. We have found it more useful to estimate by using an information-theoretic heuristic based on query expectancy [28,30] whether it will be useful to add the node. This heuristic estimates for a typical query whether an isomorphism test is likely and if it does occur how much information about other graphs in the database it is likely to provide. G. Ellis [14,15] has tried to make the hierarchy "more balanced", i.e. not giving a graph too few or too many immediate predecessors or successors. In practice, we have seen that the application of our heuristic leads to such balanced orderings though the heuristic tends to fill the general levels of the hierarchy first since these graphs are smaller and hence have cheaper comparison tests.

5.4 Parallelization of Design Method III

Phase I of Design Method III is easily parallelized. Each processor takes the next available object from the list, compares it to Q and updates S and the list as before. The only possible inefficiency is that two objects may be used in comparisons such that the answer to one may eliminate the need for the other. Thus in addition to removing objects from the list the processors should be terminated that are working on removed objects and freed for other computation. Alternatively, a processor could not work on an object that has a predecessor in a processor.

In Phase II the upward chaining from each immediate predecessor can be done in parallel, and the breadth first search over the successors of the last immediate predecessor may also be parallelized.

6 Method IV: Hierarchical Node Descriptor Method

Design Methods I-III assumed that the subgraph-isomorphism tests were to be done as a unit and further did not depend on which technique (backtrack search or refinement) was used to perform these tests. Design Method IV is based directly on the refinement model of subgraph-isomorphism. Here we will go over Method IV in detail, but first it is helpful to have an intuitive understanding of the refinement method.

6.1 Refinement

Recall that the purpose of a subgraph-isomorphism test is to find bindings for the nodes in a query graph Q and those in the database graph R that satisfy the node consistency and adjacency preservation constraints(see Section 2). We said in Section 2 that both backtracking and refinement tests start out by generating possible binding lists for each node in the query graph and that this is the only place that the type hierarchies need be consulted when performing these tests. Backtracking then goes on to effectively explore the space of all possible binding combinations.

Proof:
When are comparisons required by these systems? In Method III comparisons are required for those objects X that satisfy any of the following:

i.) All of X's predecessors are known to be in Q. (Phase I)

ii.) All of Q's predecessors are known to be in X. (Phase II).

In Method II a comparison is required for objects X that satisfy any of the following:

iii.) X is a screen

iv.) All of X's screens are known to be in Q. (possible generalization)

v.) All of Q's screens are known to be in X. (possible specialization).

Note that since screens in DB2 do not have their own screens statement iii is just a special case of statement iv and could be omitted. Now since all screens of an object are predecessors of that object: statement i implies statement iv and statement ii implies statement v. Thus every method III comparison is also required in Method II. QED. (Above it was shown that many specialization tests required in Method II are eliminated for free in Phase II of Method III. It should also be clear that Method III does not necessarily require comparisons on all screen objects from DB2 since they are not necessarily on the first level.)

This theorem is stronger than others that we have previously published. The symmetry of the proof might lead one to believe that further insight is available. This is indeed the case:

1. The object relationships inferable from DB2 are a subset of those inferable from DB3.
2. The algorithm used in Method III is not restricted to fully-specified partial orders. The algorithm used in Method II is a special case of this algorithm applied to two-level orderings!

Theoretically, little is known yet about the average run time of insertion into partial orders. In empirical tests of Method II vs. Method III, retrieval of predecessors and successors in Method III was more than twice as fast as Method II on databases of 630 and 521 concepts respectively [28,30]. The Method II database was created by eliminating all intermediate nodes (those with both predecessors and successors) from the Method III hierarchy some of which were valid answers to queries. This is impressive considering that Method III also produced 33 percent more structures as answers per query (because of and in spite of the fact that the database contained more information).

Finally, it should be noted that close matching in Method III should usually be more accurate than in Method II since it is based on more specific features: immediate predecessors as opposed to screens.

5.3 Self-Organization in Method III

"Self-organization" is the name we give to Method III systems that add "generalization graphs" to the database expressly to improve retrieval efficiency (though other uses are possible, see Section 8). These graphs are found by examining the database graphs for graphs that are subgraphs of many other database graphs but are not yet themselves in the database. There are many methods for coming up with these nodes including:

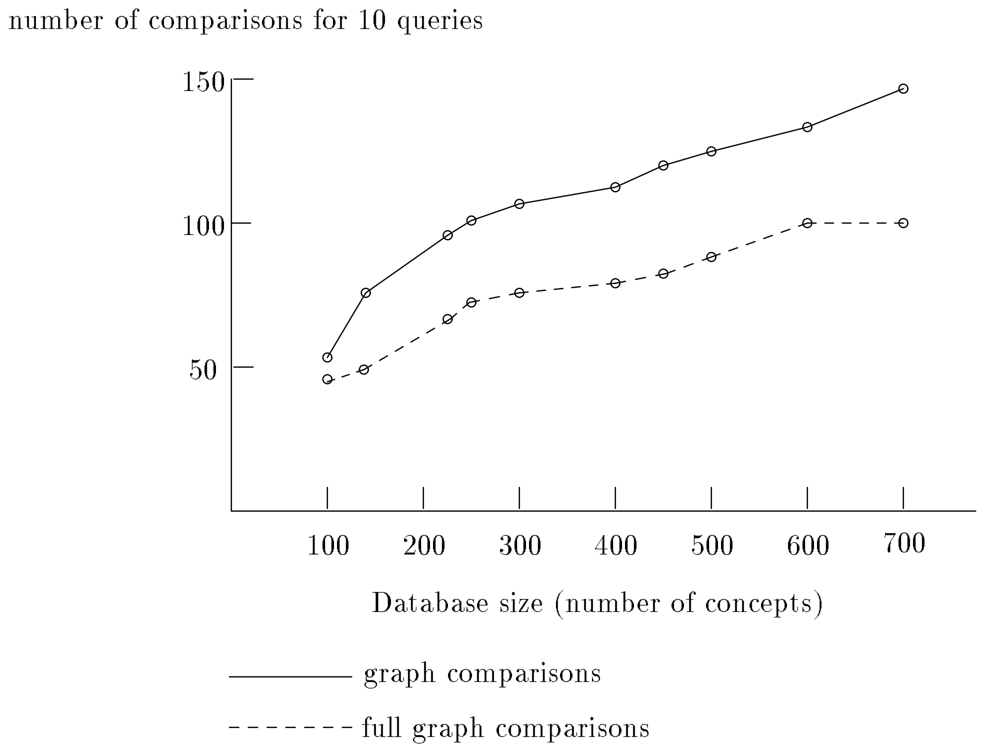


Figure 13: Retrieval time vs. database size

does comparisons on the immediate successors (and some others). Two other things point to the deficiencies of this approach: the immediate predecessor information from Phase I is not taken into account and by starting at the other end of the hierarchy the system is required to do comparisons on the most complex objects!

We have explored alternative algorithms to these that do not query the partial order in a bottom-up or top-down fashion but instead use an information-theoretic heuristic that attempts to maximize the ratio of expected information gained to comparison cost and using a few levels of lookahead [44]. We've had only limited success with these algorithms: only improvements of about 15-20 percent despite a large amount of off-line pre-processing.

5.2 Comparison of Design Methods II and III

Above, intuitive arguments have been given that Design Method III produces more efficient associative retrieval than Design Method II. Can something more concrete be said? We shall prove the following theorem:

Theorem:
Let DB2 be a two-level database to be used with Method II. Let DB3 be a Method III partially-ordered database made up of the objects and screens from DB2. Then on every query Q, Method III (on DB3) does a subset of the comparisons done by Method II (on DB2) to determine generalizations and specializations.

```
(8) For each successor X of Y in order by size (as in step (1) above) do
        If X is in I and X is a successor of Q (isomorphism test) then
            S := S ∪ {X}
            Eliminate successors of X from the rest of the for loop.

(9) Return S.
```

In our example, S is initialized to $\emptyset$. Y is taken to be G_1 and I is set to the successors of G_7: G_2, G_4. Y's (G_1's) successors that are not smaller than the query are: G_2, G_4, G_3, and G_6. G_2 is compared to Q and is a successor (and is added to S) thus G_4 is found to be a successor for free (being a successor of G_2) and is not processed further. Since G_3 and G_6 are not in I they are eliminated without isomorphism testing.

If we actually wish to insert Q into the hierarchy, the IP and IS sets of other objects have to be updated. This is done in Phase III (Figure 12(b)).

Phase III. (update immediate predecessor and successor sets of other items)

```
(10) For each x in IP(Q) do
         S(x) := IS(x) ∪ {Q} − IS(Q)

(11) For each x in IS(Q) do
         P(x) := IP(x) ∪ {Q} − IP(Q)
```

Thus, Phase II does not do an isomorphism test on a database object unless it contains each member of IP(Q) (the screens for Q). Note how the original database objects are being used as screens in Phases I and II. The big savings of Method III over Method II comes from the fact that only the immediate successors of Q need to be determined using isomorphism tests. All other successors (specializations) are determined for free. Since the objects eliminated in this way are usually the most complex, many expensive tests have been eliminated.

Close matches can be found using this algorithm much as they are in Phase II. The approximation is based on the number of immediate predecessors of Q that are contained in an object. This count corresponds exactly to the count which has been calculated in Phase II. To verify such an approximation one may of course use a maximal-common subgraph algorithm.

There are other algorithms for insertion of objects into partially-ordered sets, we recommend the one here due to its simplicity and efficiency. In practice only a small fraction of the database objects need to be compared with Q using isomorphism tests: 10 or 20 structures at the most on a database of 680 objects for example. Further, we have seen that as database size grows the increase in retrieval time is sublinear and quite possibly logarithmic. (See Figure 13.)

KL-ONE's classification algorithm [34] is somewhat different: in phase I an object is compared to the query as soon as one of its predecessors match Q (A depth-first approach as opposed to the breadth-first approach described here). Our studies have shown that the predecessor information gain for free by this method (usually simple comparisons) do not pay for the additional predecessor tests (usually more complex) required by this method. Other variations may be feasible though, such as comparing an object as a predecessor as an IP when a certain proportion of its immediate predecessors have succeeded. Since Phase I is not the expensive phase the differences here are not that significant. Some systems that maintain a partial order have Phase II work exactly as Phase I but from the other end of the hierarchy. This is not as efficient as the method here since at the minimum all successors of Q (and some others) must be queried, whereas the Phase II here only of Q in the ordering are its generalizations and its successors are its specializations. Thus the retrieval operation is essentially the same as an insertion operation. The immediate predecessor and immediate successor sets are found in two phases. Phase II makes use of the immediate predecessors found in Phase I. Both phases attempt to use the information in the hierarchy to minimize the number of isomorphism tests.

Phase I: (find IP(Q), the immediate predecessors of Q)

```
(1) List all database objects from smallest to those of the same size as Q. Sets of objects that
    are the same size need to be ordered to reflect their relationship in the partial order (if any),
    that is an object cannot be succeeded by its generalization.

(2) S := ∅.

(3) While there is a member X in the list
    If X is a predecessor of Q (isomorphism test) then
        S := S ∪ {X} − IP(X)
        Remove X from the list.
    Else
        Remove X and all successors of X from the list.

(4) Return S.
```

Ordering the database objects as in step (1) produces a topologically sorted list, i.e. a total ordering that embeds the original partial ordering by more-general-than. Since all database objects will be preceded by their predecessors in the list they only will make it to the front of the list if their predecessors have been found to be predecessors of Q. Thus, the proper screening is taking place. Although we have published this algorithm in several places this is the first time we have obviated the need for maintaining predecessor counts in Phase I. This simplification also leads to a simple parallel implementation (see below). Now let us return to our example, Figure 12. One ordered list for this database would be G_1, G_5, G_7, G_2. First G_1 is compared to Q and succeeds as a generalization and is placed in S. G_5 fails but has no successors to remove from the list. G_7 succeeds (and is added to S) and G_2 fails, completing Phase I with S = $\{G_1, G_7\}$.

Phase II. (find IS(Q), the immediate successors of Q)

```
(5) S := ∅.

(6) Y := some element of IP(Q)

(7) I := intersection of the successor sets of each element of IP(Q) except Y
```

We suggest the following implementation of step 7:

```
(7') For each z in IP(Q) except Y do
         For each successor s of z do
             Increment count(s)
         For each item s do
             If count(s) = |IP(Q)| − 1 then I := I ∪ {s}
```

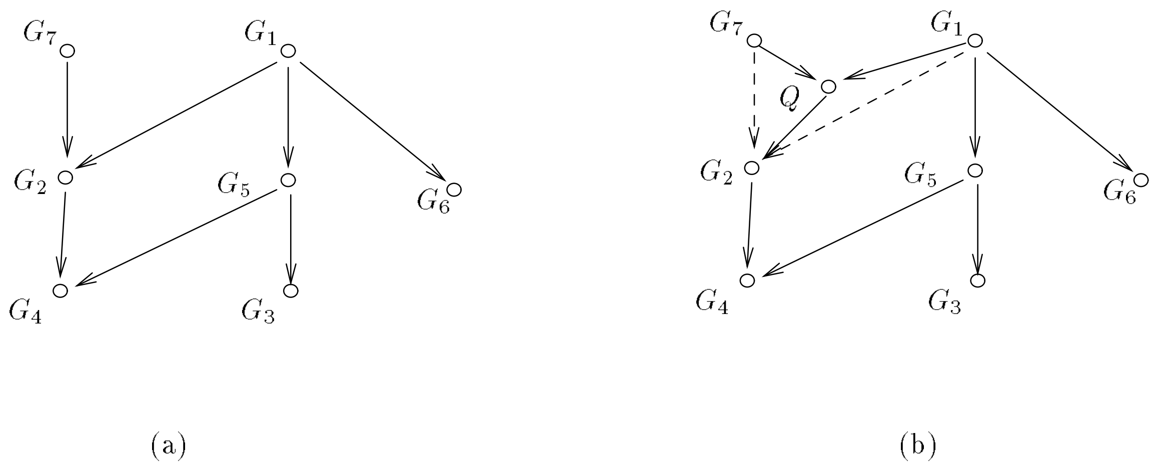


Figure 12: A Method III organization of Figure 10.

(a) is the database before insertion of the query object Q, and (b) is after insertion.

$\{G_1, G_7\}$ is IP(Q) and $\{G_2\}$ is IS(Q).

other and database objects with each other, but only on the relationship of screens with database objects. How can this inter-screen and inter-object information be useful? Suppose for example that database object R is known to be a generalization of database object S. Now once we determine that R is a specialization of Q, we know that S is also without performing further isomorphism testing! Similar reasoning applies to screens: If screen X is a generalization of screen Y, then if X is found not to be a generalization of Q, than clearly Y isn't either.

Following this line of thought, since we are interested in the interrelationship amongst all objects and screens, the distinctions between these become blurred and we are no longer left with separate levels, instead we have a multi-level partial order: Design Method III. In this section we shall prove that, ignoring pointer chasing, Method III is superior query-for-query than Method II. This should make sense since Method III uses a superset of the information used by Method II.

5.1 Method III Retrieval

In this method all database objects are placed in a partially-ordered hierarchy by the relation more-general-than. Because of transitivity only the immediate predecessor (generalizations) arcs and immediate successor (specialization) arcs need be stored(as in the Hasse diagram of any po-set (see Figure 12(a))). Other objects besides the original set may also be stored in the database to provide further indexing (see section on Self-Organization). The system's algorithms make no distinction between "screens" and "non-screens", however. (Though it is, of course, possible to add classifications to database objects so that some are filtered from the system's answers.) In essence, an object is screened by its predecessors in the ordering and screens its successors! This is another application of the syntactic-property inheritance principle.

Exactly how is this organization used for efficient associative retrieval? First, notice that to answer specialization/generalization queries it is sufficient to find where the query object Q fits in the partial order (i.e., Q's immediate predecessors and immediate successors). The predecessors

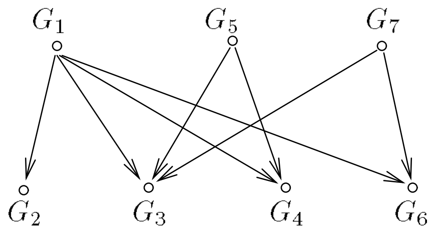


Figure 11: A Method II organization of Figure 10.

3. For those bottom level objects that are pointed to by each member of S, and are not smaller than S, do a subgraph-isomorphism test to determine if they are specializations of the query Q.

4. For those bottom level objects that have all of their screens from the database in S, and are not larger than S, do a subgraph-isomorphism test to determine if they are generalizations of Q.

5. For those bottom level objects that have many but not all of their screens in S, do a maximal-common subgraph test to determine close matches with Q. Any object that is both a generalization and specialization of Q is isomorphic (exact match) to Q.

6. To insert an object add bidirectional pointers from(to) the object to(from) screens that it contains.

One sample Method II database based on Figure 10 is shown in Figure 11. (To illustrate the technique less than optimal screens have been chosen.) Retrieval proceeds as follows: first, screens G_1, G_5 and G_7 are compared to Q with G_1 and G_7 being found to be generalizations of Q with G_5 failing. Thus, S = $\{G_1, G_7\}$. The bottom level objects G_2 and G_4 are pointed to by both G_1 and G_7 (the members of S) and hence, each is compared as specializations to Q, both succeeding. Since G_2 has its only screen (G_1) in S it is also compared as a generalization of Q and fails. Thus the algorithm returns G_1 and G_7 as generalizations and G_2 and G_4 as specializations. Note that G_3 and G_6 were eliminated without isomorphism tests.

Method II is definitely an improvement over Design Method I, since the screening step (step 1) is fast, and usually most of the database objects are eliminated without further subgraph-isomorphism tests. In essence, through pattern associativity the global property screen stage of the subgraph-isomorphism tests on the top level objects are now done simultaneously.

Parallelism can be added to this method in the obvious way: With k processors, in step 1 each processor applies the next available screen to the query object and follows the pointers to adjust screen tallies for the main database objects. The remaining isomorphism tests work the same way: Each available processor performs the next one. Again parallelism is possible due to the independence of the isomorphism and property tests.

5 Design Method III: Multi-level Partial Order

So Design Method II is a great improvement over Method I, but what additional improvements are possible? Note that Method II does not use information about the relationship of screens with each

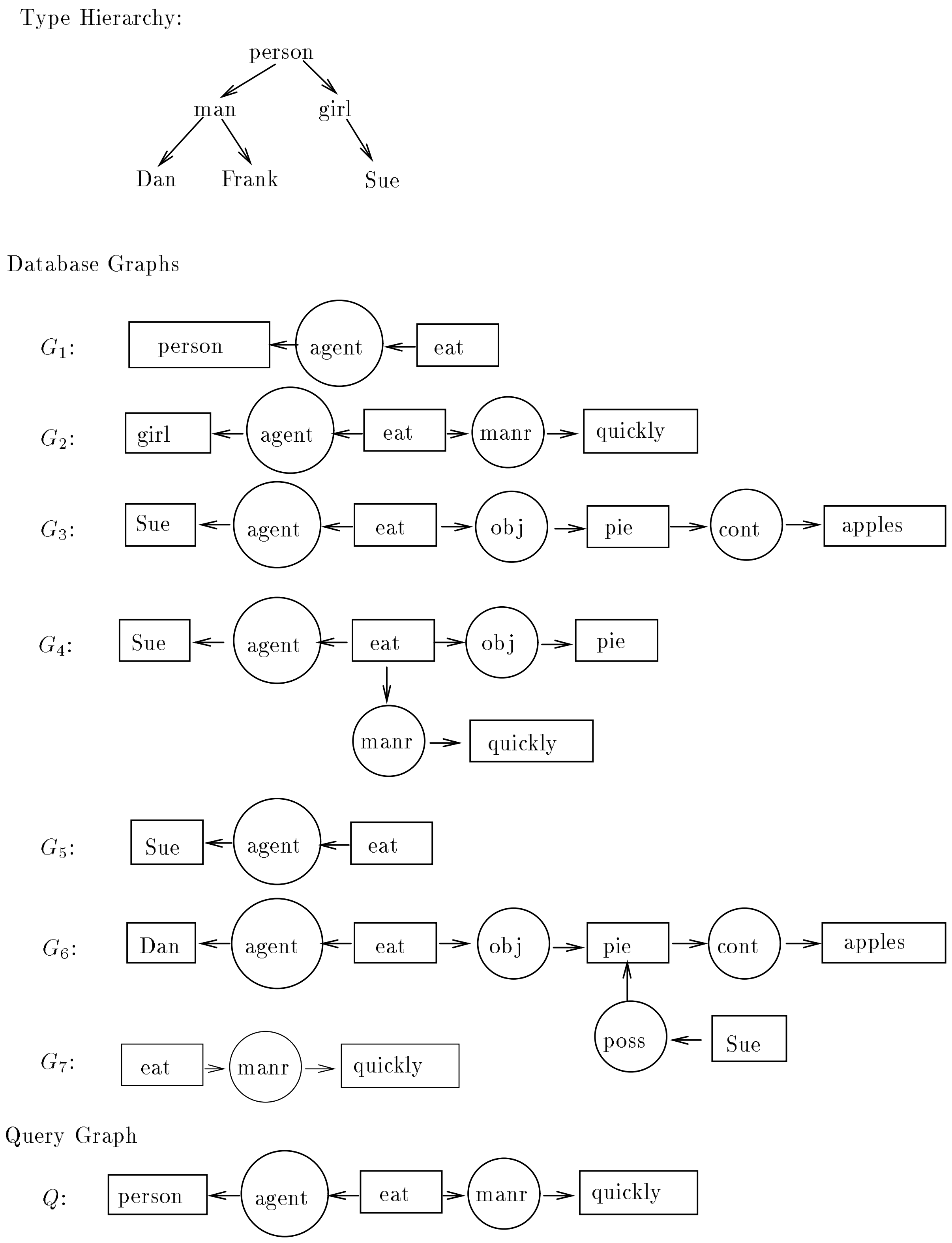

Type Hierarchy:
person
man
girl
Dan
Frank
Sue
Database Graphs
G1: person agent eat
G2: girl agent eat manr quickly
G3: Sue agent eat obj pie cont apples
G4: Sue agent eat obj pie manr quickly
G5: Sue agent eat
G6: Dan agent eat obj pie cont apples poss Sue
G7: eat manr quickly
Query Graph
Q: person agent eat manr quickly

Figure 10: Type hierarchy, database and query graph.
This example is adapted from [56] pp.92-93. "manr" stands for "manner", "cont" stands for "contains" and "poss" stands for "possesses". G_1 and G_7 are generalizations of the query Q, G_2 and G_4 are specializations of Q and G_3, G_5, and G_6 are incomparable.

database objects are sequentially ordered, and each processor works on the next database object for which a test has not yet been performed. Such parallelism is possible since the results of the subgraph-isomorphism tests do not depend on each other.

Clearly, there is more structure to be taken advantage of! In the methods that follow three different levels of pattern-associativity are used to create more efficient associative retrieval:

1. Common substructures or properties in database objects.

2. More-general-than relation between database objects.

3. Shared computation amongst individual subgraph-isomorphism refinement tests

Design Method II will use 1, Method II: 1 and 2 and Method IV:1,2 and 3.

Figure 10 gives a type hierarchy, database of conceptual graphs and a query graph that we will use to illustrate Methods II-IV.

4 Design Method II: Two-level Ordering

In Design Method II a new set of objects ("screens") are added to the database to provide an indexing or "screening" level to the original set of objects. The use of this set of objects is founded on the same reasoning that supports the global property filter that starts the subgraph-isomorphism tests: If a property is true of a graph Q, it must be true "all the more so" of a graph R it is a specialization of. We shall call this application of pattern associativity "the syntactic property-inheritance principle". To be more technical: the properties that satisfy this principle must be preserved by isomorphism. (That is, not depend on the way in which the nodes may have been numbered.) For example:

a. Every subgraph of Q is a subgraph of R

b. If Q has m cycles(edges, nodes, cliques), R has at least m cycles(edges, nodes, cliques).

c. If nodes a and b are distance d in Q, then their isomorphic nodes are no further than distance d in R.

Subgraphs (or properties) that are common to many objects are used to produce the screen level of the two-level order. These are either found through statistical methods or are supplied to the system based on human knowledge of useful indexes[1,5,13,62]. Simple screen objects (1-3 edges) are best since the time to determine if they occur in an arbitrary structure is minimal compared to more complex comparisons required by the system.

For each screen object, s, bidirectional pointers are stored to all second level objects that s occurs in. Associative retrieval takes place as follows:

1. Find which screen objects S occur in the query structure Q. This is usually done by applying subgraph-isomorphism tests on a screen vs. the query. For very simple screens (such as number of edges or nodes) other tests are possible. This step usually goes rapidly due to the simplicity of the screen objects.

2. For each bottom level object determine which members of S occur in them using the database pointers.

a solution in a reasonable amount of time, they seem to work well in many practical situations. [12]

Conceptual graphs are bipartite since there are no edges between pairs of concept nodes or pairs of relation nodes, but only between a concept node and a relation node (or vice versa). Unfortunately, the subgraph-isomorphism problem when restricted to bipartite graphs remains NP-complete [18]. In some applications of conceptual graphs the set of edges out-going from each concept node are restricted to go to at most one relation node of a given label or certain relations and concepts may be confined to connect with only certain other relations and concepts. In these cases, the subgraph-isomorphism test may be done in polynomial-time. Still the techniques that follow are valuable even in this restricted case as they reduce the number of these tests that must take place. In Section 8 we discuss how these methods may be adapted to other semantic network formalisms including those in which the more-general-than relation (subsumption) between concepts can not be determined with a subgraph-isomorphism algorithm.

Here we will describe only those aspects of subgraph-isomorphism testing that are essential to understanding the four design methods, the evolution of pattern associativity, and how type hierarchies are incorporated. Further details and efficient implementation methods can be found in [2,4,8,23,33,46].

Call the two graphs being compared Q (for query) and R. The object is to determine if Q is a generalization of R. The result of a successful test is a set of bindings that satisfy the node consistency and adjacency preservation conditions. There are two main methods for testing if such a set of bindings exist: *backtracking* and *refinement*. Both methods start out by the same. A set of possible bindings for nodes in Q are generated. Then compute for each node in both graphs, what the possible bindings are for those nodes. These bindings are determined by first verifying that the node in R has arity at least that of the node in Q and then searching the type hierarchies for a direct path from two labels, where the label from Q is equal to or more-general-than the label from R. As searching these paths may be costly (especially when no path exists), an alternative is to compile the information directly into a hash table (for all queries) that will tell directly if two labels correspond. The tradeoff, of course, is a greater storage requirement, and greater computation at system start-up, insertion or compile time. **The beauty is that once these "candidate binding lists" are created the type hierarchies need never be consulted again during an isomorphism test.** The type hierarchies are maintained as a separate data structure from the structure that organizes the database graphs and may take any desired form(e.g. lattice, tree, or acyclic graph). Methods I-III may use any correct subgraph-isomorphism algorithm(backtracking or refinement) since comparisons are considered independently of other processing. In Section VI we will take a closer look at refinement since it is essential to understanding Method IV. In Method IV the database is compiled into a form that allows, in essence, many refinement tests to be done concurrently.

3 Design Method I: Arbitrary Flat Ordering

This is the simplest design possible: place no additional structure on the stored objects at all! The system simply receives the query graph Q from the user and considers for each database object whether it is a specialization, generalization or close match with Q. Thus with N database objects N subgraph-isomorphism tests are done regardless of the query type. With k processors the

Sections 3-6 describe the database design methods and their use of pattern-associativity in depth. Section 7 describes how the methods can be extended to handle other semantic network formalisms in addition to conceptual graphs. Section 8 considers the domain validity of common patterns used to index structures in these methods. Section 9 is the conclusion.

2 The Comparison of Conceptual Graphs

The fundamental comparison operation needed takes two semantic networks A and B and determines if A is a generalization of B. (B is a specialization of A.) Recall that we will be restricting most of the paper to the conceptual graph family of semantic netwroks thus we shall assume that the word "graph" refers to directed graphs in which nodes are labelled and edges are not. Further, the labels on the nodes (for both concepts and relations) are partially-ordered by the relation more-general-than. The structures that store this relation over labels we call type hierarchies. Nodes without referent fields (used to denote specific individual objects) [56] should always be considered to be more-general-than the same node with a referent. From the graph viewpoint, the comparison operation is a subgraph-isomorphism test. Thus, A is a generalization of B iff there is a 1-1 correspondence between all nodes and edges in A and a subset (possibly not proper) of the nodes and edges in B such that each of the following conditions hold:

1. The labels on the nodes in A are equal to or more-general-than the label on their corresponding node in B. (Node-consistency)

2. For a pair of nodes that are adjacent in one structure, the corresponding pair of nodes must be adjacent in the other structure, and further, the direction of the edge between the nodes must also correspond. (Adjacency-preservation)

A is a specialization of B iff B is a generalization of A. A=B iff A is both a specialization and generalization of A. Loosely, we would like A and B be to be close matches iff there is a large graph C (not necessarily connected or existing in the database) that is a generalization of both A and B. We shall not be more formal and specific about close matches as determining their relevance often requires domain and problem (goal)-specific knowledge. The retrieval methods will select a good list of candidates to which further criteria can then be applied.

To understand the four associative retrieval design methods it is important to have knowledge of good algorithms for determining subgraph-isomorphism. The subgraph-isomorphism problem is NP-complete. So, unless a characterization of semantic nets in a given domain can be exploited, in the worst case a calculation exponentially proportional to the number of nodes in the structures is possible. In general, a brute force comparison of all possible node and edge bindings (mappings from nodes on one graph to the other) is unfeasible. In fact, it still remains an open problem as to whether a polynomial-time algorithm for the special case, the graph-isomorphism problem exists, and whether it is NP-Complete [18]. However, there are some algorithms that generally do quite well for both graph-isomorphism and subgraph-isomorphism:

> In the past decade the graph isomorphism problem has received a great deal of attention in both the practical and theoretical computing literature. The development of computer algorithms for the graph-isomorphism problem has been stimulated by such diverse applications as chemical identification, scene analysis and construction and enumeration of combinatorial configurations. Although these algorithms do not guarantee

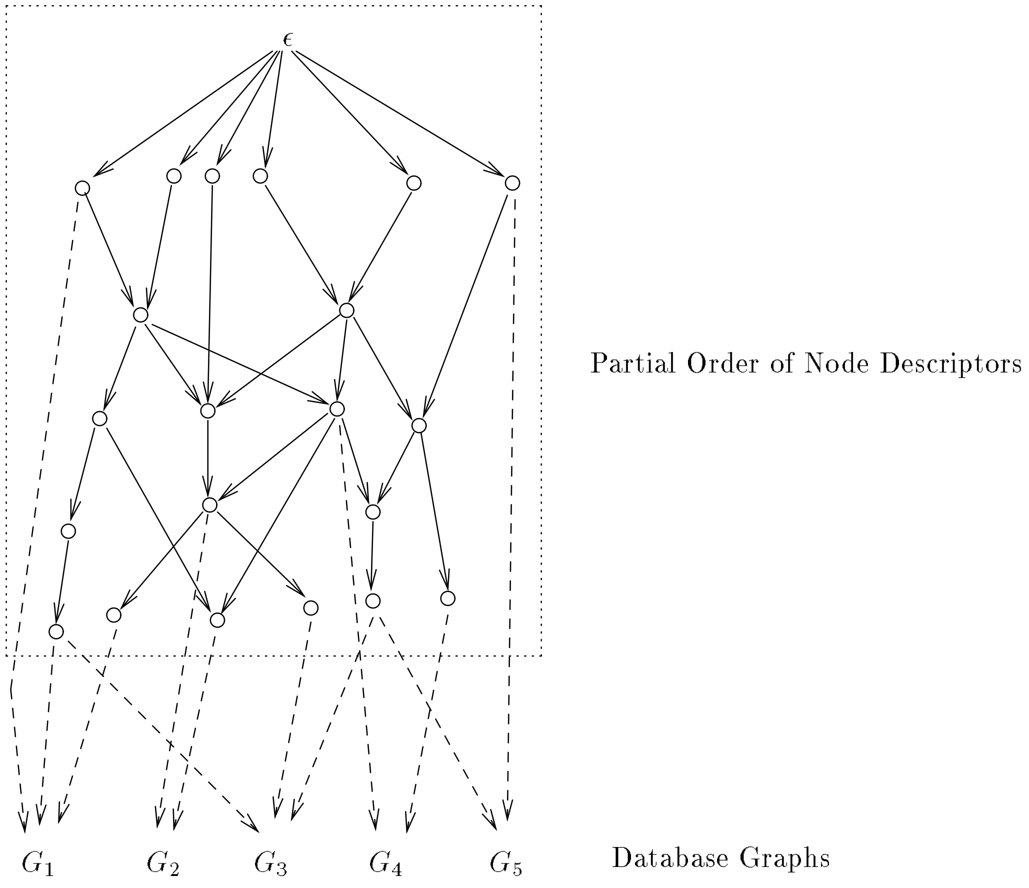


Figure 9: A Method IV hierarchy.

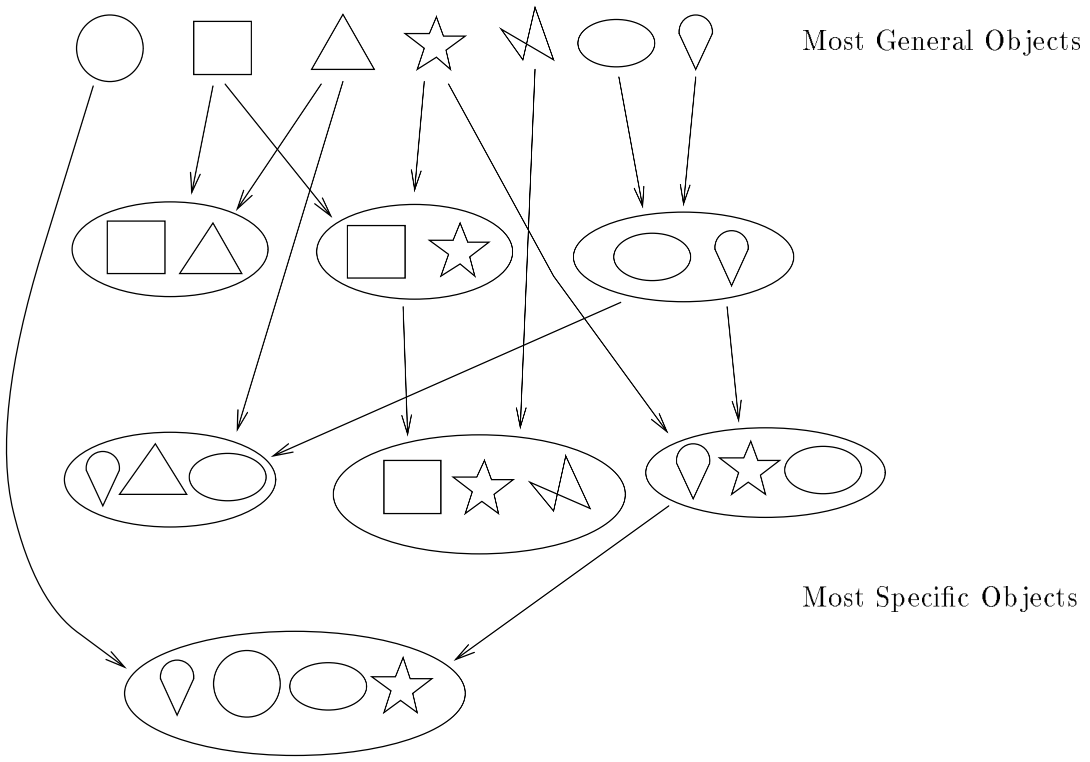


Figure 8: A multi-level partially-ordered hierarchy.

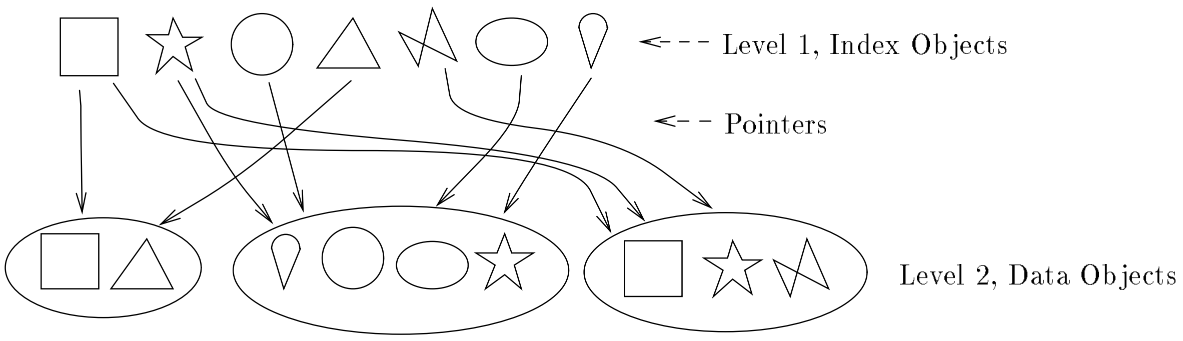


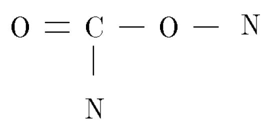
Figure 7: A two level database.

retrieval system [27,28,30], a method for organizing chess patterns stored as semantic networks [31,32,61], and (since the method does not actually require the use of conceptual graphs) have also applied it to the retrieval of DNA protein sequences in genetics and to radio signal classification [29]. This method stores *all* subgraph relationships that exist between the index structures and domain structures of the two-level method, thus creating a multi-level partially-ordered hierarchy (Figure 8). This additional pattern associativity information leads to significant performance improvement over the two-level method. It will be shown that for every query structure, the graph comparisons required by Method III are always a subset (often proper) of the graph comparisons required by Method II. Empirical evidence also supports these conclusions [28,30].

Design Method IV is based on the design of HTSS, a commercial system for chemical structure retrieval developed in Hungary. While the inspiration for the design of Method IV has come from this system, to our knowledge only a brief high-level discussion of it is available [37]. Thus the discussion, details and application of this method to conceptual graphs has been developed completely independently and undoubtedly differ from the details of that system. Method IV takes pattern associativity one step further than Method III: The commonalities (pattern-associativity) in the potential subgraph-isomorphism tests are considered and exploited. The type of subgraph-isomorphism test used is based on "refinement" [4,8,12,59] (some people call this "relaxation" [4]: Successively refine the node *descriptors* (based on connectivity and label information) until it is clear that one graph is or is not a subgraph of the other). HTSS stores a tree of relevant node descriptions in increasing specificity and finally connects them to the domain structures that satisfy them. (HTSS stores the descriptors in a tree, we will use a partial order to take advantage of the methods and understanding developed with Method III (as in Figure 9)). A query is then done by tracing each node description in turn down the tree to determine which structures they occur in. Practice, has shown that this organization is more than sufficient for most applications: 15000 structures on an IBM-PC/AT can be queried in 15 seconds. On an IBM mainframe (3090/150) the average retrieval time for a substructure in a database of 1,200,000 compounds is less than ten seconds. This method has not been applied to conceptual graphs or semantic networks. Here we show how to go about doing that. An implementation of Method IV ideas is underway. Due to the structure of conceptual graphs we expect even better performance than was achieved in the chemistry domain.

Before proceeding to describe each of these methods in detail, we need to establish some background information on the comparison of two conceptual graphs. This is covered in Section 2.

A molecule:



Its conceptual graph:

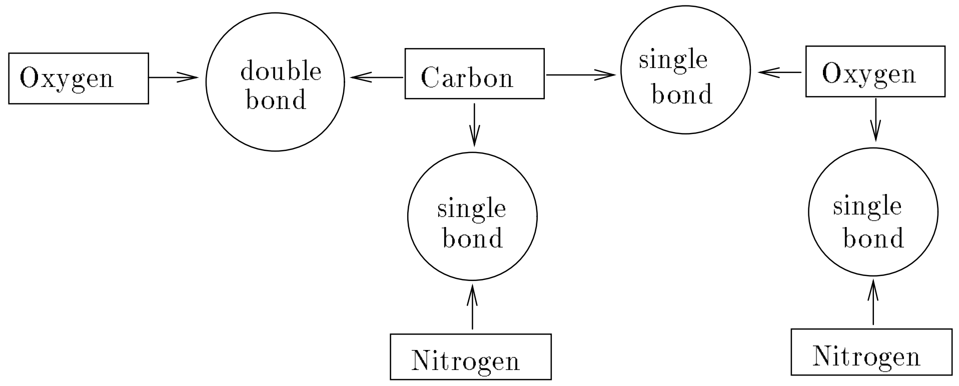


Figure 5: A molecule expressed as a conceptual graph.

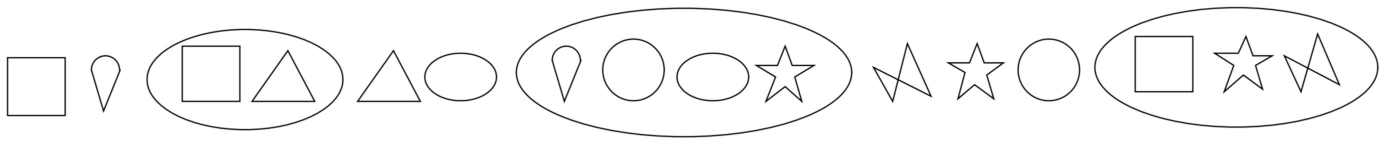


Figure 6: An arbitrary, flat-ordered database.

at all (see Figure 6 for an example. In this introductory section for simplicity we use sets of shapes rather than graphs). Since the arbitrary flat ordering uses no pattern associativity information, on a database of N items exactly N subgraph-isomorphism tests are required for each query and possibly an additional N tests to determine close matches.

Design Method II, the two-level ordering method is the method that has most often been used in chemical retrieval systems [1,5,62]. The idea is to add a second "level" of commonly occurring substructures as indexes into the original group of structures (Figure 7). Because one level of "indexing" seems justified it has been used internally in many AI knowledge representation implementations as well. Rau has recently applied a two-level method to the retrieval of conceptual graphs [41,42]. Database implementors have for years been using two-level schemes in a more restricted form, known as "inverted files" (these are files in which a large amount of index structure is created that points into relational records. Because of the simple structure of these records, if the indexing is complete the records need not be stored at all) [2,23]. Method II is an increase in pattern associativity over Method I since commonalities amongst the original structures are used to form the indexing level. For any given query, subgraph tests are only required on a subset of the original structures.

Design Method III, is a method based on a multi-level partially ordered hierarchy that we have developed for complex object retrieval [27,28,30]. Others have also applied this approach to semantic networks [9,15,19,34]. In our own work we have used Method III to produce an improved chemical in which there is no consensus world view (e.g. crime reports, political opinions, etc.). Finally, they may even provide a useful mechanism for organizing multiple single-net knowledge bases (like Sneps), associative retrieval being used to select the most relevant ones.

1.5 The Four Design Methods

In this paper we will not focus on any domain application but will show how the principle of pattern associativity can be used as the basis for the design of systems for the associative retrieval of semantic networks. By associative retrieval we mean retrieval in which stored data are retrieved by content and not location or inference. Specifically, suppose we start with a database of N semantic networks, each corresponding to one or more facts or events in a problem-solving domain. Then given a new query graph Q, the problem is to determine which of the N graphs are specializations, generalizations, and close matches.

The four design methods to be explored are:

I. Arbitrary flat ordering.

II. Two-level ordering.

III. Multi-level partial ordering.

IV. Hierarchical node descriptor method.

These design methods are purposely ordered by increasing pattern associativity. As we move from Design I to Design IV marked gains in efficiency for the "specifications of Q" query are obtained. Also, more accurate answers to "generalization" and "close match" queries also come about at no cost to efficiency. The only cost in moving from Designs I to IV is an increase in algorithmic and implementation complexity. Although we will be concentrating mainly on sequential implementations, we will also show that each of the design methods has a clear-cut parallel version with nearly k-fold speedup for k processors.

Our experience with these designs is based on research in chemical structure retrieval systems, in which very large databases of chemical graphs are used to support associative retrieval as defined above. The analogy between chemical structures and conceptual graphs is straightforward: atoms correspond to concept nodes in a conceptual graph, and bonds to relation nodes. For example, see Figure 5. The major difference is that conceptual graphs also allow generalization (type) hierarchies on their node and edge labels. We shall see that the methods smoothly handle type hierarchies.

The cost of associative queries on chemical databases is dominated by the number of graph comparison (subgraph-isomorphism) tests that need to be carried out to answer individual queries. Thus our comparison of methods will count these and not consider the cost of tracing pointers (though this cost does increase somewhat with each method). In what follows it should be remembered that an individual isomorphism test compares a query structure to a single semantic net, whereas each of the four design methods compares a query structure to a database of networks and usually requires many isomorphism tests. As we move from methods I-IV the overhead due to isomorphism testing is reduced due to the increased exploitation of pattern associativity. Design Method I, the arbitrary flat ordering method is discussed here mainly for the sake of argument. However, this is essentially the design used in the Cambridge Crystallographic Data Base [3], which expects queries to be done in batch mode! No additional structure is placed on the database objects

1.3 Conceptual Graph Queries

There are three major types of queries we would like an associative retrieval system to support: specialization queries, generalization queries and close match queries. Here we give examples of these. They will be formally defined in Section 2.

1.3.1 Specialization Queries

Typically we expect the database system to return specializations of the query. An article on the effects of commercial fishing on dolphins is a specialization of the effects of commercial fishing on sea mammals, since dolphins are a specialization of mammals. The type hierarchy is consulted when comparing graphs.

1.3.2 Generalization Queries

Requests requiring generalization are less common than requests requiring specialization. Usually someone will have a subject area and want articles that fall into that area - specializations. Occasionally, however, someone may have a particular article and want to know how it is classified by the system. That will require generalizing. The article on the effects of commercial fishing on dolphins can be generalized in several ways in a single step: the effects of commerce (of any sort) on dolphins, the effects of fishing (of any sort) on dolphins, the effects of commercial fishing on sea mammals. A node S in a stored graph will be a generalization of a node Q in a query graph if Q is a subtype of S.

1.3.3 Close Match Queries

Both generalization and specialization are used in generating close matches. If someone requests close matches to articles on the effects of commercial fishing on dolphins, a system might generalize dolphins to sea mammals, then specialize sea mammals to whales, and return any articles on the effects of commercial fishing on whales. By the same method, it would return articles on the effects of perfume production on whales (perfume production being a subtype of commerce, which is a supertype of commercial fishing). The closeness of the match can vary. For example, the above query may be matched to an article on the relationship between crop rotation techniques and gopher population if the amount of generalization and specialization is not checked.

1.4 Appropriate Applications

Note that since the information in the articles must retain pointers back to the articles from which it has come and since information may be mutually contradictory across articles it is natural to treat the graphs derived from the articles individually and not build up a "consensus reality" network as is done in many semantic network systems in which an entire knowledge base is represented as one net (e.g. Sneps [54]).

In this paper four different designs of associative retrieval systems of semantic networks are presented. In addition to bibliographic databases the methods here will be most applicable for domain applications in which aspects of the world need to be considered individually such as databases of CAD designs [11], parsed-image databases [50,51], machine vision [45] and domains

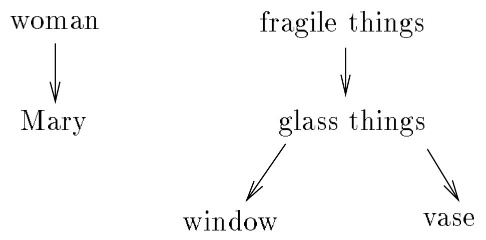


Figure 2: A type hierarchy.

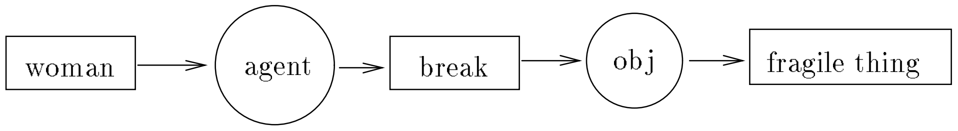


Figure 3: Query graph: “Did a woman break something fragile”.

mammals and the library contains an article on the effects of commercial fishing on dolphins, the system should return it. The system must therefore “know” that dolphins are sea mammals. This knowledge is encoded in a type hierarchy that is separate from the conceptual graphs themselves. We shall consider a type hierarchy to be a partial order by more-general-than over concept nodes and relation nodes. In the example above , “Mary” is a subtype of “woman” and, “window” is a (not necessarily direct) subtype of “fragile things” (see Figure 2).

Given the type hierarchy, conceptual graphs may be used to make certain types of inferences with little difficulty. If the graph given in Figure 1 is stored in the database then a system can answer “yes” to the question “Did a woman break something fragile?” (see Figure 3, and the stored graph, (being more specific) can be returned as a proof (and if necessary can be processed further to find that the woman is Mary) . If the conceptual graphs describe the content of articles, the same specialization-finding mechanism can be used to return a graph in the face of the query “Is there an article on ...”. For example, “is there an article on a woman breaking something fragile?”

It is outside the intended scope of the paper to consider the advantages and disadvantages of using conceptual graphs instead of traditional keyword approaches for text retrieval, but it is worth noting one advantage since it is an application of pattern-associativity: the ability to exploit the relationships between concepts. For example, suppose we are interested in papers on drugs that cause diseases, with the keyword approach the query “FIND drugs AND diseases” would return a large number of articles, most of which are irrelevant, whereas with conceptual graphs the desired query can be formulated easily (see Figure 4).

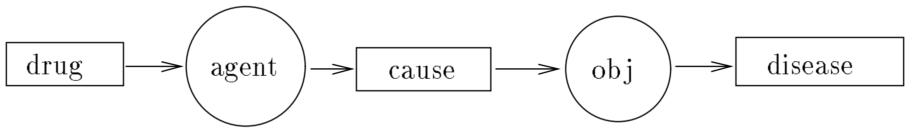


Figure 4: Bibliographic conceputal graph query.

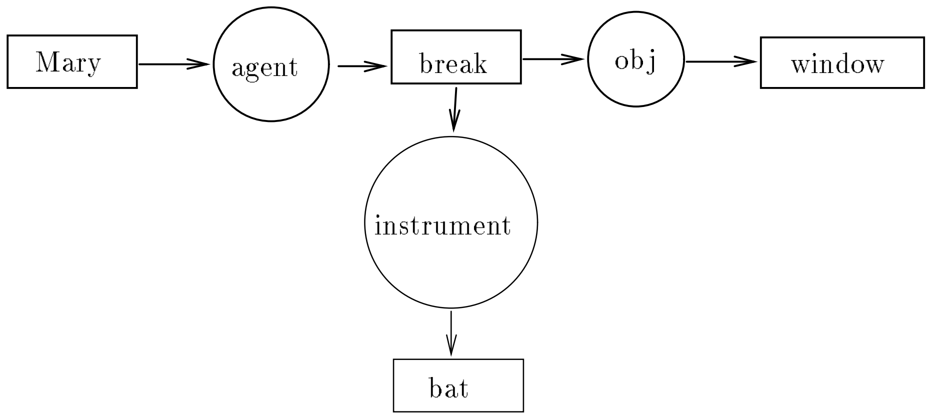


Figure 1: Conceptual graph for “Mary broke the window with the bat”.

“obj” stands for “object”.

articles, other attributes are gleaned from the request to guide the search(such as which journals to consider, earliest date interested in, which library, etc.). Most often, the search will be for specializations of the query, but it may be for generalizations or close matches. The graph and search-guiding attributes collectively are called a query. For our purposes we will assume the query is just a graph. Extending the process to include attributes is not difficult. Given a query, the database search operations will return the article(s) that match the query. To do so, the database will need to be provided with a comparison function that compares the query graph to graphs in the database. For our purposes, the comparison function will be a subgraph-isomorphism test.

1.2 Conceptual Graphs

In what follows we will assume that the semantic network family we are using is Conceptual Graphs [56]. In Section 8, we discuss how the retrieval techniques may be extended to other families. We will take Sowa's convention that conceptual graphs are graphs in which nodes are labelled and edges are directed and unlabeled. The nodes can be divided into two classes: concept nodes and relation nodes(which refer to relationships between concept nodes). In an equivalent formulation of conceptual graphs they are viewed as graphs in which nodes are labelled with concepts and directed edges are labelled with relation types. We choose the first formulation as it simplifies the discussion, but these techniques can be easily adapted to the latter case [33].

As an example conceptual graph (one that is based on case theory [16]) consider the sentence “Mary broke the window with the bat”, the action is one of breaking, Mary is the agent, the window is the object, and the bat is the instrument. This would be represented in Sowa's formalism as shown in Figure 1.

1.2.1 The Type Hierarchy

In addition to the relations between concepts that are expressed by a conceptual graph (as a representation of a sentence, a discourse, or facts about the world), there are implicit relations between concepts. If a user is looking for articles on the effects of commercial fishing on sea

Sometimes it is simplicity which is hidden under what is apparently complex; sometimes on the contrary, it is simplicity which is apparent, and which conceals extremely complex realities... No doubt, if our means of investigation became more and more penetrating, we should discover the simple beneath the complex, and then the complex from the simple, and then again the simple beneath the complex, and so on, without ever being able to predict what the last term will be.

— Henri Poincare [40]

1 Introduction

This paper has several thematic objectives:

1. To present a practical overview of the main methods of comparing semantic networks and organizing them for associative retrieval.
2. To present the main principle that is being exploited in the evolution of these methods to higher performance.
3. To present a new organizational scheme based on further evolution of this principle.
4. To outline parallel implementations of these methods.

The principle of Pattern Associativity is informally stated as follows: **The more that is known about how pieces of information relate to each other, given the ability to efficiently exploit this knowledge, the more effective is a problem-solving system.** We shall not formalize this notion here. Pattern Associativity relates to the recognition, representation, and economical exploitation of what others have called "mutual information" or structure [24,36,43].

Indeed representing and exploiting interrelationships is the principle behind the semantic network knowledge representation formalism. The advantages gained by the semantic network formalism over traditional logic representation occur at both the conceptual and implementation level. At the conceptual level, logically related items can be viewed as a unit. At the implementation level logically related items are physically in close proximity and thus support faster access. Although logical propositions can be structured somewhat similarly no such organizing principle is implied by the methodology.

1.1 Examples of associative retrieval: bibliographic databases

Suppose the task is to retrieve articles from a library based on their content [6]. Each article will have an associated semantic network. The graph must describe the article and must include a concise description of the content of the article, the author, title, and other bibliographic information (including how to find the article in the library). We will not concern ourselves here with how the graphs are created (in particular how natural language is parsed into conceptual graphs (cf.[38,53]) but how these graphs should be organized to efficiently answer typical queries.

To use the database, a person may formulate a request in English describing the desired article or articles. This request is translated into a semantic network by a similar mechanism to that which derives a graph from an article. In addition to producing a graph describing the desired

Pattern Associativity and the Retrieval of Semantic Networks

Robert Levinson
Department of Computer and Information Sciences
University of California Santa Cruz
Santa Cruz, CA 95064 U.S.A.
(408)459-2087
ARPANET:levinson%saturnucscc.ucsc.edu
UUCP:ucbvax!ucsc!saturn!levinson

Abstract

Four methods for the associative retrieval of semantic networks are described. These methods differ from those traditional approaches, such as SNEPS, in which an entire knowledge base is treated as a single network. Here the knowledge base is viewed as an organized collection of networks and is most appropriate for applications (such as bibliographic retrieval) in which pieces of knowledge need to be treated individually. Method I is an arbitrary flat ordering of database graphs, Method II a two-level ordering, and Method III is a full partial order. Method IV is a novel method known as "hierarchical node descriptor method" that is based on the "refinement" method of subgraph-isomorphism. A "pattern associativity" principle explains the development and effectiveness of each of these methods. Moving from Method I through Method IV there is a steady increase in both pattern associativity and efficiency. A theorem is proven that establishes the superiority of Method III over Method II despite the fact that Method II is the method most often used. A brief discussion of how parallelism may be incorporated also accompanies the description of each method. Most of the paper applies these methods to conceptual graphs and a later section shows how the techniques can be extended to other semantic-network formalisms. The paper concludes by showing how generalization graphs constructed through pattern associativity may also have semantic validity in the domains from which they have been derived.

Topics: Algorithm Design, Associative Retrieval, Conceptual Graphs, Databases, Graph Isomorphism, Knowledge Representation, Parallelism, Semantic Networks.