A Path-oriented Knowledge Representation System: Defusing the Combinatorial Explosion[1]

Stefan Feyock
Computer Science Department,
College of William & Mary
Williamsburg, VA 23185
feyock@cs.wm.edu

Stamos T. Karamouzis
National Research Council Associate
MS 152, NASA Langley Research Center
Hampton, VA 23681-0001
stamos@icat.larc.nasa.gov

John S. Barry
Lockheed Engineering & Sciences
MS 152, NASA Langley Research Center
Hampton, VA 23681
j.s.barry@larc.nasa.gov

Steven L. Smith
Lockheed Engineering & Sciences
MS 152, NASA Langley Research Center
Hampton, VA 23681
s.l.smith@larc.nasa.gov

Abstract

LIMAP is a programming system oriented toward efficient information manipulation over fixed finite domains, and quantification over paths and predicates. A generalization of Warshall's Algorithm to precompute paths in a sparse matrix representation of semantic nets is employed to allow questions involving paths between components to be posed and answered easily. LIMAP's ability to cache all paths between two components in a matrix cell proved to be a computational obstacle, however, when the semantic net grew to realistic size. The present paper describes a means of mitigating this combinatorial explosion to an extent that makes the use of the LIMAP representation feasible for problems of significant size. The technique we describe radically reduces the size of the search space in which LIMAP must operate; semantic nets of more than 500 nodes have been attacked successfully. Furthermore, it appears that the procedure described is applicable not only to LIMAP, but to a number of other combinatorially explosive search space problems found in AI as well.

Introduction

The set of Artificial Intelligence (AI) search/representation techniques referred to as "weak methods" generally represent problems in terms of states and operators. A typical problem description specifies one or a few start states, a test for discerning goal states, and a set of operators for generating successor states from current states. The state space is implicit, being generated by operator application as search progresses, and in many cases is potentially infinite.

[1] The research reported in this paper was supported in part by NASA Grant NCC-1-159 and NASA Contract NAS1-19000

There is a significant subset of practical AI problems, however, that involve a finite domain that is known and enumerable a priori, and unary or binary predicates over that domain; semantic net representations constitute an important instance. Well-known vector- and matrix-based representations can efficiently represent finite domains and unary/binary predicates and allow effective extraction of path information by generalized transitive closure/path matrix computations. We have developed an intelligent information tool, called LIMAP (Feyock and Karamouzis, 1991, 1992), which employs a set of abstract sparse matrix data types along with a set of operations on them as the basis for representing and manipulating finite enumerable domain (FED) problems. The present paper describes one such problem, and our experiences in attempting to apply the LIMAP system to its solution. We discuss in particular the solution we developed for overcoming the combinatorial explosion that occurred with increasing domain size, a technique that is applicable to a wide class of FED problems.

Problem Description

DRAPHYS

Semantic net representations constitute a significant subclass of FED Problems. We will begin by describing the DRAPHYS system, which employs a typical example of such a representation. DRAPHYS (Abbott, 1991) is a model-based system designed to reason about a physical system represented by its model; our group's research has centered on the modeling and diagnosis of jet engine faults. The model of the physical system is a semantic net consisting of a set of nodes, representing components such as compressors, turbines, and combustors, as well as a set of four kinds of arcs representing the relations *functionally-affects* and *physically-affects*[2], together with their inverses. Malfunctions are diagnosed by determining, for each component in the system, whether a possible propagation path exists linking that component to each symptomatic component. A possible propagation path is a path that satisfies the constraint that it contains no instrumented component whose sensor reading is normal.

Figure 2 and Figure 3, below, give pseudocode and LIMAP definitions of DRAPHYS's mode of operation. It is evident that DRAPHYS must determine, for each component, whether a path exists to each symptomatic component, and whether the path is in fact a possible propagation path. The original version of DRAPHYS accomplished this by means of backtracking search, a procedure that proved to be prohibitively time-consuming for large systems. Since LIMAP is a system that specializes in producing the set of paths between nodes of FEDs, it was decided to attempt to apply it to this problem.

---

[2] Component A functionally affects component B if A's malfunction can affect B's operation via a functional causal path; component A physically affects B if A's malfunction can affect B's operation via a physical path, such as fragments from A piercing B. A typical unary relation occurring in this model is *is-instrumented(A)*

The LIMAP Knowledge Representation System

LIMAP is designed to represent semantic nets, which of course are an important class of representations based on FEDs. LIMAP differs from other net-based representation systems in its emphasis on the efficient storage of large sparse nets, and on the provision of a second-order query capability oriented toward queries involving paths. Its design was motivated by the observation that queries of the form "is there a relation R such that nodes x and y are in relation xRy?" "is there a path form x to y? a path fulfilling constraint C? where can I go from x? how can I get to x?" arise frequently in AI in general, and in diagnostic problems like those addressed by DRAPHYS in particular. The following section provides a brief overview of LIMAP's query language and design; details can be found in (Feyock & Karamouzis 92).

LIMAP Overview

The LIMAP DDL/DML

The LIMAP implementation model is based on a representation that employs Boolean and symbolic vectors and adjacency matrices[3] to represent unary and binary predicates, as well as an efficient transitive closure computation capability that allows Boolean or symbolic path matrices to be computed and manipulated.

As is the case for an ordinary first-order database system, LIMAP capabilities are invoked via a language interface that consists of two parts. One is the data definition language (DDL) for specifying both the data the system is to contain as well as "meta-data," that is, information about the structure and constraints that govern the data contained in the system. The other is the data manipulation language (DML), the subset of the language concerned with the specification of queries and updates on the data. We will categorize the LIMAP functions accordingly. A brief summary of the LIMAP DDL and DML follow; Feyock and Karamouzis (1991) contains a complete listing.

DDL operations: The basic DDL operations are:

> DEFREL < name > < specification > < type > < representation >
> < specification >:: = (< number >) | (< number > < number >)
> <type> :: = boolean | symbolic
> <representation> :: = sparse | dense
>
> and
>
> DELREL <name>,

to define or delete a relation, respectively. DEFREL defines a relation by creating a new array according to the values of the parameters, and binding this array to <name>;

[3] An adjacency matrix is a binary matrix representation of a graph. Entry ij is 1 iff a link joins node i and node j.

<specification> stipulates whether the array will be a vector or matrix, as well as the index range(s); <type> specifies whether the declared relation will be represented by a Boolean or symbolic array, while <representation> allows the user to choose a sparse or dense array representation.

DML operations: The major DML operations are:

STORE	relname	value [row] column	Store value
RETRIEVE	relname	[row] column	Retrieve contents
TCLOSE	relname		Transitive closure
PATHS	relname	row column	All paths
MULT	relname	relname	Multiply
TRANSPOSE	relname	relname	Transpose

STORE and RETRIEVE perform the indicated operation on the specified array position, in accordance with the array's type and representation, while MULT and TRANSPOSE typify a variety of standard matrix operations made available by LIMAP. Except in DEFREL it is transparent to the user whether the array representation is sparse or dense. This transparency extends to the other attributes of the array wherever possible.

Calculating Paths

The TCLOSE and PATHS operations form the core of LIMAP's path manipulation capability. TCLOSE computes the transitive closure of a semantic net. If G is a semantic net then the transitive closure G* of G is a network containing an edge <a, b> if and only if G contains a path (of length 0 or greater) from a to b. TCLOSE employs Warshall's Algorithm (see, e.g., Horowitz & Sahni, 1976) for computing G* given an adjacency matrix that represents a network G. Intuitively, the algorithm scans the matrix top to bottom, left to right. If a 1 is encountered, say in row i, column j, then row i is replaced by row i OR row j, and the scan continues from position ij.

A straightforward extension, described in (Feyock and Karamouzis, 1991), of Warshall's Algorithm to symbolic adjacency matrices produces a matrix, termed the path symbolic adjacency matrix (PSAM), whose ij entry contains the set of all paths from node i to node j. The extension, shown in Figure 1, consists of storing the paths created by appending all paths in (i,j) to all paths (j,k) into the cell (i,k). PATHS[i,j] retrieves the set of all paths from node i to node j by referencing entry ij of the resulting PSAM, thus enabling quantification over paths.

> Let M be an N×N path matrix, i.e., a matrix each of whose entries contains a set of paths (represented as lists). Initially M[i, j] = {(i j)} i.e., the singleton set containing a one-step path from i to j, iff there is a link from node i to node j; if not, M[i, j] = NIL. Then after the following loop is executed, M[i, j] contains the set of all paths from node i to node j.

```
for k := 1 to N do                    ; scan array from top down
  for i := 1 to N do          ; scan array from left to right
    if (i ≠ k & M[i, k] ≠ NIL) then
      for j := 1 to N do M[i, j] := M[i, j] ∪ (M[i, k] || M[k, j])
```

The || operater is defined as follows:
If $p = (v_1, \ldots, v_k)$ and $q = (v_k, \ldots, v_r)$ then $p \,||\, q = (v_1, \ldots, v_k, \ldots, v_r)$

Figure 1: Path Matrix Computation

Control Structures

The distinction between procedural and nonprocedural predicate calculus specifications blurs if the underlying domain is finite, since the FORALL and EXISTS quantifiers map in an obvious way to loops ranging over the domain elements. It has been our goal to give the LIMAP DML as non-procedural a character as possible. In particular, LIMAP notation is an adaptation of the (function-less) predicate calculus, with extensions to allow data retrieval in addition to data specification. Perhaps surprisingly, we have found that minimal modifications of the control macros described in (Charniak et al., 1987) were suitable for the task of expressing the required quantifications. Here is a summary of the general form of the control structure implemented by these macros:

(FOR ((< $variable_1$ >: IN < set_1 >)
(< $variable_n$ >: IN < set_n >))
[:WHEN <when-expression>]
< FOR-keyword > < $expression_1$ > . . . < $expression_n$ >)

The construct (<$variable_i$>: IN <set_i>) causes the variable to iterate over the elements of the set, which may be specified as a list, a vector, or a matrix row or column. Unless a false when-expression is present, the FOR-body is evaluated and a result is produced as governed by the FOR-keyword. Iteration then proceeds to the next set of variable values.

FOR keywords	
:ALWAYS	true if all the values of body are true
:FILTER	produce a list of the non-NIL values of body
:FIRST	produce the first non-NIL value of body
:SAVE	produce a list of all values of body.

While the description of these constructs is procedural in form, the effect when programming in this notation is that of writing FORALLs and EXISTs, with the proviso that any variable values that are found to "EXIST" are collected in accordance with the FOR keyword and returned as value. The following section contains an example application of LIMAP.

DRAPHYS in LIMAP

Figure 2 summarizes the operation of the DRAPHYS diagnostic system.

```
for each C in set-of-components do
  if "C has failed" is a valid hypothesis
  then add C to set-of-valid-hypotheses;
end for;

component C is a valid hypothesis iff  there is a POSSIBLE PROPAGATION PATH
from C to every symptomatic sensor

A path is a POSSIBLE PROPAGATION PATH iff every instrumented
component on the path has at least one symptomatic sensor

If set-of-valid-hypotheses contains
   one element: done
   more than one element: DRAPHYS waits for more symptoms to develop and
    disambiguate the diagnosis.
```

Figure 2: Basic DRAPHYS Operation

LIMAP allows this procedure to expressed concisely. The code shown in Figure 3 creates the set of valid hypothes:

```
(defun determine-hypotheses (components symptomatic-sensors)
; components =(def) set of all components to be
; considered as hypotheses
   (for (c :in components)
                :when (is-valid-hypothesis c) :filter c) )

(defun is-valid-hypothesis (c symptomatic-sensors)
  (for (s :in symptomatic-sensors)
                :always (exists-bad-path c s) ))

(defun exists-bad-path (c s)
  (for (p :in (paths 'engine c s) ) ; paths from c to s
    :first (for (component :in p) :always (not-known-ok component) ))  )

(defun not-known-ok (c)
  (or (null (instrumentation c)) (symptomatic c)))  ; symptomatic is a boolean vector

(defun instrumentation (c)  ;  returns list of sensors associated with c
  (for (s :in components)
        :when (and (is-sensor s) (retrieve 'engine c s)) :save s))
```

Figure 3: DRAPHYS in LIMAP

The Size Barrier

The key statement in the Figure 3 code is the line

```
(for (p :in (paths 'engine c s) ) ; paths from c to s
```

occurring in the *exists-bad-path* function. (paths 'engine c s) is a reference to the PSAM for relation *paths*, and exemplifies the capability to quantize over paths that is a major strength of LIMAP. It is also the source of a combinatorial explosion in terms of storage requirements when system size grows. Whereas DRAPHYS precomputes no paths, and thereby incurs unacceptable runtimes for large problems, LIMAP precomputes all paths, and encounters space limitations as the domain size grows. In particular, replacing DRAPHYS' backtracking path search with LIMAP's PSAM capability on the 23-component jet engine model originally operated on by DRAPHYS significantly improved the run time performance of the diagnostic reasoner. Increasing the size of the model to over 100 components, however, produced a PSAM matrix that, despite its sparse representation, was so large that paging overhead made its use computationally infeasible. It was evident that precomputing and storing all possible paths between nodes was unworkable in terms of space. An approach that avoided both the inordinate time requirements of DRAPHYS and the excessive space consumption of LIMAP was required.

Node Matrices

The solution that was developed was to modify the extended Warshall's Algorithm depicted in Figure 1 so that M[i, j] would store not the entire set of all paths from node i to node j, but only the nodes occurring on those paths.

Let M be an N×N node matrix, i.e., a matrix each of whose entries contains a set of nodes. Initially M[i, j] = {i, j} i.e., the set containing the two nodes occurring on the one-step path from i to j, iff there is a link from node i to node j; if not, M[i, j] = NIL. Then after the loop in Figure 4 is executed, M[i, j] contains the set of all nodes occurring on paths from node i to node j. More precisely: after executing the node matrix computation algorithm, M[i, j] = {n | ∃path p from node i to node j, and n occurs on p}

```
for k := 1 to N do                    ; scan array from top down
  for i := 1 to N do          ; scan array from left to right
    if (i ≠ k & M[i, k] ≠ NIL) then
      for j := 1 to N do M[i, j] := M[i, j] ∪ (M[i, k] || M[k, j])
```

Figure 4: Node Matrix Computation

It is important to note that if it is necessary to establish only the *existence* of a path between i and j, the node matrix M is as efficient as the adjacency matrix: a path exists iff M[i, j] is not NIL. If an actual path between nodes i and j is required, the original backtracking search method is employed, but with the constraint that the search consider

only nodes in the set M[i, j]. If the net is sufficiently sparse (as is the case in our application), it is feasible to generate the set of all possible paths between i and j by the same approach.

Performance Improvement

The time required to compute the set of all paths between two nodes of a digraph is known to be exponential in the number of nodes. The extended Warshall's Algorithm employed by LIMAP is $\partial(n^5)$, while the node matrix computation is $\partial(n^4)$. The amount of time saved was found in practice to increase the size of the problems that could be attacked by nearly an order of magnitude. At least as important, however, is the fact that restricting the search to nodes in M[i, j] can achieve a radical reduction in the size of the search space, depending on the degree of sparseness of M. Since a node matrix records only the set of nodes occurring on any path between i and j, rather than the paths themselves, it is to be expected that node matrices require significantly less storage than path matrices. A further source of performance improvement results from the fact that sets allow the familiar representation in terms of bit strips (bit i is on iff i is in the set) that allows set union to be implemented as bitwise OR, a highly efficient operation on essentially all machines. Each matrix entry is then a single binary number of N bits. Not only matrix storage requirements but also matrix creation time are greatly reduced compared to the PSAM, since the time-consuming link operation is no longer needed. Furthermore, if the net is sparse, then most bits of most bit strips will be 0, allowing a number of well-known compression techniques to be applied to the entries of the node matrix. The amount of storage required is $N^2 \cdot s \cdot k$ bits, where s is the expected number of distinct nodes on paths between arbitrary nodes, and k is compression overhead. In the worst case (no compression) the amount of space required is $N^2 \cdot N.\log_2(N)$ bits. We thus have representational parsimony on two levels: the sparse-matrix representation facilites afforded by LIMAP, and the bit strip compression technique employed at the level of the matrix entries. Table 1 summarizes the time and space savings achieved by the node matrix technique[4].

Model	LIMAP	Node Matrix
23-component engine:	Time: nominal Space: nominal	Time: nominal Space: nominal
166-component hydraulic system:	Time: 5 hours Space: 3M	Time: 30 minutes Space: 50K
198-component oil system:	Time: 12 hours Space: 5M	Time: 35 minutes Space: 61K
490-component fuel system:	Attempt abandoned; infeasible due to paging problems	Time: 1 hour Space: 65K

Table 1

[4] The "nominal" entries for the 23-component engine model reflect the fact that most circumstances tend not to be noticed - or measured - until they become irksome.

It is apparent that the task addressed by DRAPHYS is a typical FED problem, and that the node matrix technique we have described is applicable to FED representations in general. LIMAP, which was developed to accomodate the requirement for quantifying over paths as well as individuals in FED representation, runs into space limitations the space limitations illustrated by Table 1 when processing larger models. If (as is frequently the case) the net is sparse, the node matrix technique allows significantly larger systems to be represented in feasible amounts of space, while retaining sufficient speed to allow quantization over paths. The example graph (tree, in this case) depicted in Figure 5 illustrates the striking reduction in search effort that the node matrix technique can achieve.

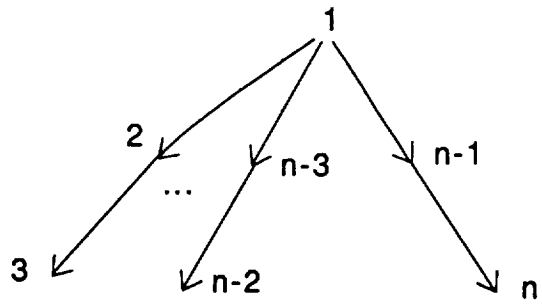

Figure 5

Suppose the task is to find a path from node 1 to node n. It is evident that depth-first backtracking search will explore all of nodes 1 to n. If a node matrix is used, then the [1,n] entry will contain {1, n-1, n}, the set of nodes occurring on the path(s) from 1 to n. Search in this 3-node space is trivial. While this is admittedly an extreme example, we have found that the use of a node matrix increases the size of the problem that can feasibly be attempted by nearly an order of magnitude.

Conclusion

An earlier paper (Feyock and Karamouzis, 91) described the LIMAP system that we developed in response to the need for a higher-order logic capability in FED problems, particularly the need to quantize over relations and paths in diagnostic systems. Practical experience with this system showed that while it coped well with models of small to moderate size, large representations resulted in unacceptable storage requirements. In this paper we have presented technique that can lead to drastic reductions in search space size, allowing models of more than 500 nodes to be processed. The procedure described is applicable not only in the context of LIMAP, but to a number of other combinatorially explosive search space problems found in AI as well.

References

Abbott, K, H. *Robust Fault Diagnosis of Physical Systems in Operation*, NASA Technical Memoradum 102767, NASA Langley Research Center, Hampton, Virginia, 1991.

Charniak, E., et al. *Artificial Intelligence Programming* (2nd Ed.). Lawrence Earlbaum Associates, 1987.

Feyock, S, and S. Karamouzis, "Design of an Intelligent Information System for In-Flight Emergency Assistance," in *Proceedings of the 1991 Goddard Conference on Space Applications of Artificial Intelligence*, Greenbelt, MD, May 1991.

Feyock, S, and S. Karamouzis, "A Path-Oriented Matrix-based Knowledge Representation System," in *Proceedings of the Fourth International Conference on Tools with Artificial Intelligence*, Arlington, VA, November 1992.

Horowitz, E., and Sahni, S. *Fundamentals of data structures*. Computer Science Press, 1976.