Parallelizing Subgraph Isomorphism Refinement for Classification and Retrieval of Conceptual Structures

James D. Roberts

UCSC-CRL-94-48
December 20, 1994

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

Major applications of graph-based knowledge representations will require quick response times on extremely large knowledge bases. Although algorithmic developments have provided tremendous improvements in speed, we believe implementation on parallel processors will be needed to meet long-term needs. This paper presents a new parallelization of a subgraph isomorphism refinement algorithm for performing projection tests and retrieving conceptual structures. The improved algorithm is faster, requires fewer processors, and is compatible with recent relation-based representations of conceptual structures.

The new parallelization takes advantage of the features of contemporary data-parallel parallel machines by exploiting bit-parallelism in wide data words. Processing numerous graphs on a single parallel array, it combines the strengths of prior parallel subgraph isomorphism parallelizations with multi-level indexed search. It incorporates lattice codes of the concept-type hierarchy to avoid a bottleneck in our prior parallelization and forms all node candidate binding lists in parallel. Simulation results of the behavior of the refinement algorithm with parameterized synthetic data sets are presented as are implications for hardware tailored to processing conceptual structures.

Contents

1 Introduction

Conceptual structures, a graph-based information processing methodology representing knowledge as sets of concept nodes and relation nodes [12], has been extensively researched over the last 10 years and is now being incorporated into a variety of applications as well as into ANSI standards. One of the methodology's key operations is testing for generalization; one graph is a generalization of another if it is in some way embedded in the other graph. In conceptual structures concept nodes match if the node in the subgraph is subsumed by that in the supergraph as specified in a separately maintained concept-type hierarchy. The graph embedded in the supergraph is called a projection of the subgraph. A knowledge base of conceptual structures can be organized into a partial order based on the subgraph relation, forming a generalization hierarchy. Locating a conceptual structure in the generalization hierarchy (or determining where it would be placed if it were to be inserted) identifies immediate generalizations and specializations as adjacent graphs in the partial order, thereby classifying as well as retrieving or inserting the structure.

Testing for projections is the most time consuming portion of classifying a conceptual structure into a database of graphs. Classification speed is crucial for systems that must handle a large number of queries or inference operations, particularly for anticipated knowledge bases of hundreds of thousands of structures. Speed improvements can be achieved by reducing the number of necessary projection tests, and by reducing the time required for each projection. Techniques such as multilevel indexed search, as incorporated into the PEIRCE Workbench and recently into KL-ONE, tremendously speed structure retrieval by minimizing the number of projection tests required and in the future by re-using matching information across graphs in the generalization hierarchy. We now believe that even with these techniques parallel processing will be necessary for future large-scale applications. This paper presents a revised algorithm for reducing the time of projection tests through parallel processing.

For the purposes of this paper we will consider projection as equivalent to subgraph isomorphism (SI). Negation and nested contexts introduce important differences between SI and projection, but the operations of SI remain central. Other differences are already incorporated into our SI algorithm. Although having exponential time in the worst case, subgraph isomorphism has an empirical expected-case time of $O(n^4)$ bit operations (where n is the number of nodes in the larger of the two graphs) using the SI refinement algorithm developed by Ullmann [13]. The polynomial expected-case time means that parallelization can provide a substantial speedup. Our new algorithm exploits both the parallelism of multiple processors and the parallelism of the multiple bits in a processor's data word.

1.1 Prior Parallelizations

Several others have proposed or implemented parallel algorithms for subgraph isomorphism; here we summarize those most relevant to our current work. Ullmann's original implementation used the bit-parallelism in the data word of a conventional serial processor to achieve $O(n^3)$ empirical time on graph isomorphism [13]. Implementations by Willett *et al.* on a data-parallel processor empirically show $O(n^2)$ time using n^2 single-bit processors, demonstrating an efficient parallelization of SI with substantial time improvement [14]. Lendaris has used an inherently parallel neural network approach to processing conceptual structures (CS), including join, simplify, and projection [5, 6]. His results demonstrate

neural networks performing important filtering of candidate graphs which must then be further processed for projection. Our own prior work focuses on the computationally efficient approach of using multi-level indexed search to extensively prune the number of graphs to be tested then speeding each test through parallel processing. Our prior paper ([4]) discusses modifications to Ullmann's SI refinement algorithm to accommodate the specific requirements and improve performance in projection tests on CS, analyses the advantages of a parallelized multilevel indexed search over exhaustive comparison to all graphs in the knowledge base for a bounded number of processors, and describes parallel compilation (encoding) of the concept-type hierarchy.

1.2 This Paper

This paper extends prior parallelizations, combining bit-parallelism, parallel processing, and multi-level indexed search. In particular the new parallelization exploits multi-bit parallel processors, more typical of contemporary data-parallel machines, allowing numerous CS to be processed on a single, small parallel array. We thereby modify our prior "processor farm" algorithm that uses multiple arrays for classification in a large database into an algorithm using a single array with work-load balancing. The resulting parallelization is faster while using fewer processors. This paper also introduces using subsumption codes to avoid the bottleneck of consulting a subsumption table rather than the less effective caching method we previously proposed. We also discuss graph storage and allocation, and present an algorithm for forming all SI node candidate binding lists in parallel. Additionally, this paper incorporates Levinson's recent work on a relation-based CS representation and summarizes refinement behavior on a parameterized synthetic data set.

Section 2 summarizes Ullmann's algorithm, its application to conceptual structures, and prior parallelizations. This is followed by a brief overview of multilevel indexed search and lattice coding. Section 3 shows that a relation-based representation of CS is readily incorporated into SI refinement, with performance benefits. Section 4 highlights the new multi-bit parallelization and Section 5 presents the details. Revisions to the multilevel indexed search parallelization are discussed in Section 6. Parameterized simulations on synthetic data sets are presented in Section 7 and ongoing work in hardware tailored to CS processing is summarized in Section 8. Section 9 summarizes the results presented in this paper in comparison to prior parallelizations and suggests directions for future work.

2 Background

This section outlines the algorithms that form the basis of the parallelizations presented in the remainder of this paper and briefly summarizes our prior work. We first discuss Ullmann's serial SI refinement algorithm and its parallelization by Willett *et al.*, as well as the modifications required to adapt the algorithm to CS. This is followed by a brief description of multilevel search and lattice coding as relevant to our new SI parallelization.

2.1 Subgraph Isomorphism Refinement

Ullmann's algorithm features a refinement procedure to extensively prune the search tree in a backtracking algorithm [13], speeding SI tests. Given two graphs G_a and G_b a SI test determines if G_a is a subgraph of G_b. Ullmann's refinement algorithm represents the graphs

and the binding between nodes as three Boolean matrices: A (the adjacency matrix of G_a), B (the adjacency matrix of G_b), and M (the 'match' matrix). The adjacency matrices must be symmetrical[1]. A bit $m_{ij} \in M$ is 1 if node $i \in G_a$ is a candidate for binding to node $j \in G_b$ (otherwise m_{ij} is 0). Figure 2.1 diagrams the overall structure of the algorithm. First, selected graphs are moved from main memory into the processors; those selected could include all graphs (exhaustive comparison) [14], those which pass filtering criteria (single-level indexing) [14, 6], or those passing multi-level indexing (initial graph SI tests filtering subsequent graphs) [9, 4]. Second, the match matrix M is formed. Third, the backtracking refinement search is performed. The refinement procedure is called at each branch of a backtracking search tree (including the root), significantly pruning the number of branches explored.

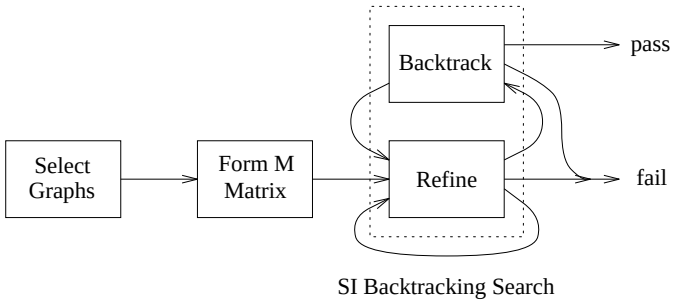

Figure 2.1: Flow Diagram of the Refinement Subgraph Isomorphism Algorithm

The initial match matrix is formed by iterating over all i and j and setting m_{ij} to 1 if the node type of i is the same as that of j and the arity of i is less than or equal to the arity of j, otherwise m_{ij} is set to 0, thus forming the set of all possible bindings. The M matrix is then refined to remove candidate bindings based on the topology of the graphs. This is not guaranteed to remove *all* invalid bindings. The refinement algorithm keeps a candidate binding m_{ij} iff $\exists(x)\ a_{ix} \wedge \neg(\exists(y) m_{xy} \wedge b_{yj})$. This logical criteria can be expressed by the matrix operations

$$M' = M \wedge \overline{A \times \overline{(M \times B)}}.$$

If at any time there is a row in M consisting of all zeros the refinement fails, as a node in G_a does not have a corresponding node in G_b. Otherwise, refinement iterates until the M matrix has not changed from the prior iteration. In the latter case the backtracking portion of Ullmann's algorithm determines if M represents a valid sub-isomorphism, checking that each row of M has exactly one 1 and that no column has more than a single 1. If so, we're done. Otherwise the backtracking portion recursively continues the search by arbitrarily selecting one binding for a node in G_a and calling refinement again. The full algorithm terminates when either an isomorphism has been found or there are no more bindings to try. In the latter case no subgraph isomorphisms exist.

As a simple example, consider the conceptual structures in Figure 2.2. The initial matrices for the graphs are:

[1] *I.e.* the graphs must be undirected; Ullmann shows modifications for directed graph isomorphism, but they are not applicable to subgraph isomorphism [13].

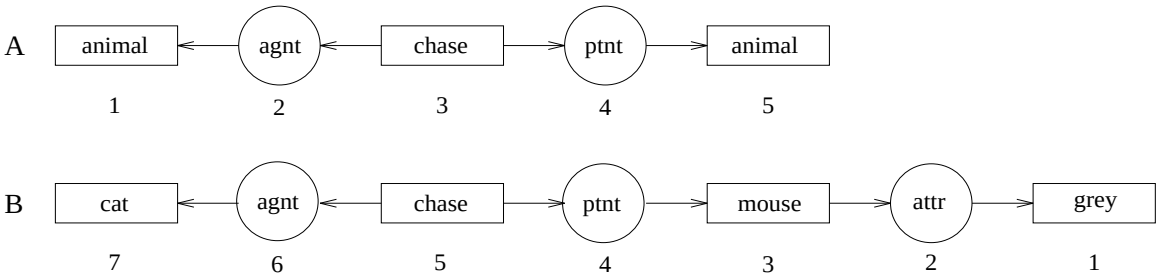

Figure 2.2: CS Refinement Example

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} B = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} M_0 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

In this example, after a single iteration of refinement the match matrix becomes

$$M_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

the algorithm immediately resolves the ambiguous mapping of *animal*, mapping $a_1 : b_7$ and $a_5 : b_3$ and no further work is necessary. Hughey *et al.* describe this and another example of subgraph isomorphism using Ullmann's algorithm in somewhat greater detail [4].

The matrix operations are readily parallelizable with all candidate bindings (bits in the match matrix) processed simultaneously. Willett *et al.* present a parallelization for the DAP-610, an older parallel machine with 4096 single-bit processors [14]. Their coding optimizes the matrix expressions, eliminating the transpose for one of the Boolean matrix multiplications. They present three parallel implementations of Ullmann's algorithm. In the first, each processor stores 1 bit from each of the matrices and the parallel processor computes one SI test at a time. In the second, each processor stores the entire matrices and the array performs up to 4096 SI tests simultaneously. In the third, a hybrid, graphs are processed using the second implementation until some number of graphs are completed after which the remaining graphs are processed using the first implementation. Their experimental results on a small organic chemistry database show that the third method is clearly faster. Willett *et al.* propose a fourth method that uses the second method, adding new graphs as earlier graphs complete processing.

2.2 Subgraph Isomorphism and Conceptual Structures

We make several modifications in applying SI refinement to conceptual structures [4]. First we must account for edge direction; Ullmann's algorithm is for undirected graphs and cannot be adapted to SI in DAGs. In CS, edge direction is determined by the relation type so that we treat relations as labeled edges and check argument numbers to verify direction in the backtracking portion of the algorithm. Treating relations as labeled edges rather than

nodes also has a major speed advantage, as empirical times are $O(n^4)$ for serial and $O(n^2)$ for Willett *et al.*'s parallel implementations of Ullmann's algorithm, where n is the number of nodes. Additionally, nodes may bind if concept node of G_a is subsumed by that of G_b. Node subsumption is determined by a separate concept-type hierarchy, typically based on class inheritance. We now initially set m_{ij} to 1 iff $a_i \preceq b_j$ and |relations adjacent to a_i| $\leq$ |relations adjacent to b_j|. Additional criteria and filters, such as minimum-cycle length, can also be applied in forming the initial match matrix.

In testing projection between CS, there does not have to be a 1:1 mapping between the concept-type nodes and relations in G_a to those in G_b. This is readily handled by relaxing the mapping criteria applied in the backtracking portion of the algorithm.

2.3 Multilevel Indexed Search

Multilevel indexed search (MIS) [9] can significantly reduce the number of SI tests necessary. As an independent survey shows Levinson and Ellis' methodology to be the fastest known structure search pruning [2], it forms the framework and motivation for our parallelization efforts. Rather than providing a single level of indexing as in conventional databases, Levinson's Method III MIS creates a partial order over the knowledge base using the more-general-than relation. Objects are screened by predecessors in the poset and in turn screen successors. Smaller and simpler objects prune out more time-consuming comparisons on larger objects. Method III empirically prunes by a factor of $O(N/\lg^2 N)$, where N is the number of objects in the knowledge base, and also supports conceptual clustering, generalization, and machine learning. We hypothesize that at any given point in the execution of MIS, $O(\lg N)$ graphs are available as candidates without risking unnecessary comparisons. Current work on multilevel indexed search explores ways of reducing the work in performing a SI test on a given graph by exploiting binding and other information from tests on its predecessors.

2.4 Lattice Codes

The transitive links in a poset can be compiled into codes for each element, avoiding both link chasing and large look-up tables. As a basic example, the poset is represented as an adjacency matrix and the reflexive-transitive closure of the matrix is computed. Each row of the resulting matrix is the code for its respective element in the partial order [1]. These simple codes are N bits long where N is the number of elements in the poset. More formally, the poset is plunged into a lattice and the lattice is then encoded; more advanced encoding algorithms approach the lower bound of $\Omega(\lg N)$ length codes for ideal poset topologies [1, 3]. Lattice codes are useful in speeding operations on a CS concept-type hierarchy and can also be used to further prune comparisons in a Method III graph classification, provided the "query" graph is known to be in the knowledge base. In addition to reducing storage, shorter codes reduce execution time. The time to compute subsumption, greatest lower bound, or least upper bound between two concept types or two graphs in the hierarchy is proportional to the length of the code [3].

On a parallel machine, one lattice code can be simultaneously compared against numerous others. This permits operations such as identifying all descendents of an element, or even the intersection or union of the descendents of several elements, in time proportional to the length of the codes times the number of root elements. With full data parallelism

(one processor for each item) time for these lattice-code operations are independent of the number of elements in the poset or the lengths of the inheritance paths.

3 A Relation-Based Representation for CS

Levinson's Universal Data Structure (UDS) combines the features and benefits of neural networks, semantic networks, relational databases, and conceptual structures in a single representational framework [7]. Its relation-based representation of graphs is of particular relevance to the new work presented in this paper. In UDS, conceptual structures are transformed such that the relations and their adjacent concept-types in the CS become nodes in the UDS graph. Edges in the UDS graph represent the bindings between the concept-type arguments of their relations. Figure 3.1 shows the UDS representations of the CS shown in Figure 2.2. A node has the form *[relation, arg1, arg2,... argN]*, representing higher-order relations as a single node. Each node is in the UDS hierarchy, having a single subsumption code that can be used in forming candidate bindings in subgraph isomorphism tests.

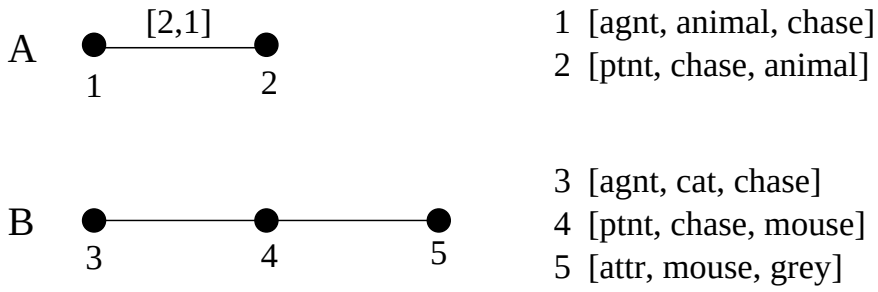

Figure 3.1: UDS Relation-Based Graph Representation

The refinement subgraph isomorphism algorithm can be applied directly to the UDS representation without change, but with several benefits. For the UDS graphs of Figure 3.1, the refinement matrices are:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} B = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

The initial match matrix immediately shows the subgraph isomorphism as the *agent* relations can only bind to one another. As seen in this example, representing CS at the relational level usually reduces the number of nodes for the SI algorithm. The number of refinement iterations are reduced due to the greater specificity of the nodes. Combined, these significantly speed SI tests and in turn classification in a database.

4 The New Multi-Bit Multi-Graph Parallelization

Having covered the background information, we now present our new parallel implementation. Previous parallelizations, including our own previous work, assumed massively parallel machines with a single-bit processor such as the DAP-601 or the CM-2. Recent machines feature much wider data words in each processor, typically 32 bits. Since nearly all bit operations are independent of one another at any step of the refinement, it is possible

to assign as many bits to a processor as it can handle simultaneously rather than place a single bit of each matrix in each processor. This increases speed by making much better use of the available resources and permits simultaneous comparison of many graphs even on a relatively small number of processors. This also reduces the cost of a machine by reducing the number of processors necessary.

In addition to converting from a single-bit to a multi-bit processor parallelization, our new parallelization covers graph storage, a parallel algorithm for forming the M match matrices, and revises the parallelization of Method III to account for a significant number of graphs being tested on a single parallel array. As noted, recent work on UDS can be directly incorporated. The overall structure is the same as in Figure 2.1. We use Method III (or its successors) to select several graphs for testing, move them into processors, form the M matrices, then execute the SI refinement. In this paper we cover only refinement in detail; the backtracking portion is assumed to be essentially the same as that of Willett *et al.*

The following descriptions involve two assumptions for the purpose of simplifying discussion. First, we assume the processors are connected in a 2-dimensional mesh topology, typical of most contemporary data parallel machines (higher dimensional topologies such as hypercubes can readily emulate lower-dimensional meshes). Second, we assume that the number of bits in the processor data word is at least as large as the number of nodes in a graph. Otherwise each row of the matrices must reside in multiple processors. The implementation details that arise when this condition is not met can be dealt with in a straightforward manner, primarily by additional communication steps during matrix multiplication.

The following sections cover our modification and parallelization of the refinement portion of Ullmann's SI algorithm. Refinement represents the bulk of execution time in SI, that in turn represents the bulk of time in Method III MIS classification.

5 Details of the Multi-Bit Refinement Parallization

Adapting the SI refinement algorithm to contemporary multi-bit processors, we combine the features of Willett *et al.*'s different methods and achieve additional benefits. In the new parallelization, we assign an entire row of each of the matrices to a processor and execute the refinement algorithm for many graphs on a single-data parallel array, giving a group of processors a new graph once they have completed their prior SI test. We exploit the bit-parallelism of b-bit processors so that a given SI test requires $1/b$ as many processors with no slowdown. Having fewer processors and the full matrix row in a single processor greatly reduces communication time. As with Willett *et al.*'s first method, having multiple processors per graph speeds refinement. As with their second method, having several graphs on a single parallel array minimizes wasted processors when many more processors are available than are needed for a single graph comparison. Although not yet implemented in our simulations, we propose a load-balancing approach similar to Willett *et al.*'s third and fourth method to avoid the poor processor utilization which would result if the implementation were to wait until all graphs finished before proceeding to the next group of graphs, integrating the load balancing with Method III.[2] Finally, specific to CS, we

[2]Our proposed implementation is similar to Willett *et al.* 's third method in that each graph is assigned to multiple processors as in their method one.

use lattice codes of the concept-type hierarchy, performing a subsumption test on the codes in forming the M match matrix. This avoids the bottleneck of all processors consulting a single concept-type hierarchy and allows forming all rows of the match matrices of all graphs simultaneously.

5.1 Graph Storage and Assignment to Processors

Since only $O(\lg^2 N)$ graphs are actually tested, where N is the total number of graphs in the knowledge base, each processor stores a large number of graphs in off-chip memory but processes only a tiny fraction of them. Although during computation each processor is assigned a single row of a graph's matrices (corresponding to a single node), each processor's memory holds the full adjacency matrix, node subsumption codes, and relation type labels for a complete graph. Graphs are assigned to processors for storage randomly, and the odds of a conflict in spreading the graph to nearby processors is quite small. If there is a conflict, the graph can be moved to another region of the processor array or held back until the nearby processors have completed the first graph they were assigned.

The first step, then, is to move the selected graphs from off-chip memory into the processors' registers. The following gives a high-level description for the simpler case of forming the initial group of graphs to be processed. Here we present the case where the query graph G_b is known to be larger than the knowledge base graphs to be tested (a test in Phase I of Levinson's Method III MIS). Although other cases are somewhat more complex, simulation shows that this data retrieval phase is only a few percent of total refinement time.

1. Divide the array into equally sized subsections. Each section is 1 processor wide and $|G_b|$ processors tall.
2. Select one graph to be tested from among those stored in each array subsection that is in the MIS queue of graphs.
3. Distribute the selected graph to the other processors in the sub-array so that each processor receives one row of the A adjacency matrix, and the corresponding node lattice code. This is done for all selected graphs simultaneously.
4. Broadcast and store the rows of B in the processors, one row simultaneously for all active knowledge base graphs.

5.2 Forming the Match Matrix for Comparing Two CS

Figure 5.1 gives the pseudocode for forming the M matrices. The algorithm processes all nodes in all the selected knowledge base graphs simultaneously. In addition to this parallelism, the algorithm has the advantages that there is no communication between the processors and no consultation of a central concept-type subsumption table. Node lattice codes and relation labels for the query graph are broadcast from the controller to all processors.

As the refinement algorithm is called many times per SI test, and iterates several times per call, its time dominates over that of forming the M matrix even when the latter includes additional filters.

Even so, some criteria are better handled in the refinement portion of the algorithm rather than in forming M. For example, one could test that the relations of each node are a subset of those of the query graph. A subset algorithm on a serial machine could

```
/***** FORM M MATRICES *****/

/** data in registers, each processor **/
par code(n) Anode /* n-bit row of matrix A */
par subsumptioncode acode /* subsumption code for Anode */
par int arels /* number of Anode's edges (relation) */
par code(m) Bnode /* m-bit row of matrix B */
par code(m) Mrow /* corresponding row of match matrix */

/** Test Candidate Bindings **/
foreach nodex ∈ G_b
      Mrow(nodex) ← 1
      if |arels| ≰ |brels| /* brels broadcast from controller */
            Mrow(nodex) ← 0
      else if acode ⋠ bcode /* bcode broadcast from controller */
            Mrow(nodex) ← 0
```

Figure 5.1: Parallel Algorithm for Forming Candidate Binding (Match) Matrix

use hashing to test membership in the larger set, iterating over the nodes in the smaller set for time $O(a)$ where a is the size of the smaller set (the arity of a node of G_a in our application). We can not use this method on a data-parallel array and still process all nodes of all graphs simultaneously, and must instead iterate over all elements of the larger set (the arity of a node in G_b), giving time $O(a_m A_m)$, where a_m is the maximum arity of any node in any of the database graphs and A_m is the maximum arity of any node in the query graph. In the worst case this is $O(n^3)$, asymptotically worse than the expected-case time of parallel SI refinement. Here it is clearly better to just check edge labels and directions in the backtracking portion of the code once a tentative SI mapping has been found. Rather than performing time-consuming operations for all possible node mappings, it is better to perform those operations only on the mappings in an otherwise valid subgraph isomorphism.

5.3 Multi-Bit Parallel Refinement

With the A, B, and M matrices established in the processors, the refinement procedure itself simply iterates the matrix operations as given in Section 2.1 until the M matrix is unchanged or contains one or more rows of all zeros. Details of the communications and bit-manipulations required in the Boolean matrix multiplications, including the transposition, are machine dependent.

The allocation of one row of the matrices to a processor offers several advantages. With this layout, the "summing" portion of the Boolean vector-product is replaced by a zero-test on a single data word, avoiding time consuming inter-processor communication. This is particularly important as each parallel multiplication requires n iterations of this operation. The layout also allows testing if any row of a matrix is all zeros or if the matrix has changed since the last iteration in just one or two instructions. Further, with all rows of the matrices of a given graph allocated to a column of the parallel array, the Boolean matrix transpose requires only a regular, efficient broadcast of each row of the matrix to the other processors in the subarray, which then select the appropriate bits for their destination matrix row.

Since one graph can be significantly larger than the other, such as the query graph G_b in this section's discussion, many processors will have rows of the B matrix but not of A and M matrices, nor the intermediate matrices R and T. This does not significantly hurt performance, however, as all processors are active in the matrix operations during most of the steps.

The new multi-bit implementation of SI refinement significantly complicates implementation details over our prior method, however. In addition to the load balancing described in above, it is necessary to execute the backtracking portion of Ullmann's on multiple processors in the data parallel array as in Willett *et al.*'s methods two through four.

6 Implications for Classification into a Database

In addition to speeding SI tests, the multi-bit parallelization allows performing multiple tests simultaneously on a single data-parallel array. By incorporating multi-level indexed search (MIS) we can minimize, and in some cases eliminate, unnecessary graph comparisons.

6.1 Improvement in Multi-Level Indexed Over Exhaustive Search

The multi-bit implementation strengthens our prior analysis demonstrating the advantage of MIS over exhaustive parallel search [4]. With b-bit processors we now need $1/b$ as many processors, or we can process b times as many graphs simultaneously. Updating the prior analysis to the multi-bit implementation is straightforward. Except under conditions where nested parallelism (comparing multiple graphs and parallelizing the refinement matrix operations) is not possible, the new MIS implementation is always faster than exhaustive search and a factor of b faster than the single-bit implementation. Taking the number of processors as the space measure, MIS always has a much better space-time product than exhaustive search. If memory is used as the space measure, exhaustive search has a space-time product a factor of $\lg N$ better as memory cost dominates over processor cost. However the million or more processors required[3] is impractical for the foreseeable future. With memory cost dominating, the parallel applications have better space-time products than their serial equivalents.

6.2 Modification to Prior Multi-Level Indexed Search Parallelization

It is, however, necessary to reconsider the approach to parallelizing MIS. Previously, a parallel array of a thousand processors could handle only one to a few graphs; with a dozen or more graphs available at any stage in the MIS comparisons would be farmed out to several processor arrays [4]. This processor-farm model included implicit load balancing (each array requesting more work when it was done) and there were more graphs queued for comparison than the available processors could handle so that a graph need not be compared until it was certain that the comparison was necessary. On the minus side, communication was heavy as the full knowledge base graph and concept-type subsumption data had to be sent from wherever the MIS management was executing. Although having all graphs on a single parallel array significantly reduces communication outside the array and the use of concept-type subsumption codes greatly reduces communication within the array,

[3] A knowledge base with 64 K graphs averaging 16 nodes each would require a million processors for exhaustive parallel search, one with 1 M graphs averaging 32 nodes each would require 32 million processors.

the new implementation affects the implementation of the multilevel indexed search itself. With 32-bit processors, an array of 1024 processors could handle thirty 32-node graphs simultaneously, but at any given time the MIS algorithm would have identified only one to two dozen graphs requiring comparison. One must thereby choose between not using all the processors or speculatively comparing graphs.

Performing speculative graph comparisons using an earlier version of Phase I of Levinson's Method III appears to be the better choice. In the more recent implementations of Method III, the queue of candidate graphs begins with only the top of the poset (generalization hierarchy) and candidates are added as the poset is traversed, limiting not only the number of SI tests but also the number of graphs "visited." By initially enqueueing all graphs smaller than the query graph and then dequeuing them if a predecessor fails the older version guarantees a large queue of candidate graphs. Although there will be some graph comparisons that are later determined to be unnecessary, this is expected to give a faster parallel implementation as far fewer processors would be idle. It should also be possible to integrate the MIS dequeuing with the refinement load balancing; if a currently executing graph is dequeued it can be abandoned and another graph started. The data parallel array can also handle much of the queue and other MIS management. This speculative comparison approach is currently under investigation.

7 Simulation Results of Refinement Behavior

We have begun evaluating the new parallelization by simulating its parallel execution on a serial workstation. A probabilistic analytical model for predicting the likelihood of a bit in the M match matrix changing from 1 to 0 (and in turn the probability of a SI test failing or succeeding) based on the densities of the A, B, and M matrices would be overly complex due to the dependencies between bits in the matrices. We have thereby run simulations of the refinement procedure, emulating two data parallel machines: the proposed 16-bit MISC architecture [11] and the commercially available 32-bit MasPar MP-2 [10]. The simulations use parameterized synthetic data sets to study the effect of different graph characteristics on refinement behavior.

Our simulations go beyond the simple case described in Section 5, allowing graphs to have more nodes than bits in the processor data word (requiring multiple columns in the array), and graphs with fewer than half the processors in one column of the array (allowing multiple graphs in a single column). These add only a small number of inter-processor communications and additional operations.

Graphs in the knowledge base are generated as random adjacency matrices A with $|G_a|$ within a range of specified sizes. The probability of any bit in an A being on is specified by the parameter $p(A)$. Ones on the diagonal are disallowed, prohibiting self-cycles in the graphs, and the resulting graph is tested to see if it is connected. If an A adjacency matrix does not correspond to a single connected graph, it is discarded and another is generated randomly. Query graphs are generated in the same way.

Our results include several experiments varying parameters as shown in Table 7.1. As before, $|G_x|$ is the size of a graph and $p(X)$ indicates the probability that a bit in the adjacency matrix of graph X is one. The "mid1" experiment approximates the chemical database used by Willet *et al.* Both the "mid1" and "mid2" experiments have adjacency matrix densities corresponding to 1 or 2 edges per node; the others range from 2 to 9 edges per node. Figure 7.1 plots the portion of graphs remaining to be processed (neither failed

Experiment	$\|G_a\|$	$\|G_b\|$	$p(A)$	$p(B)$	$p(M)$
small	8	12	0.20	0.30	0.50
mid2	16	24	0.13	0.15	0.40
mid1	8–19	19–25	0.10	0.10	0.45
large	24–26	30–31	0.15	0.30	0.25

Table 7.1: SI Refinement Experiment Conditions

nor unchanged) at each iteration averaged over 4 queries and Figure 7.2 plots the cumulative number of graphs that have failed. All the examples show similar behavior in the number of iterations needed to complete most of the graphs, and only the "small" experiment shows a major difference in the total number of graphs that fail.

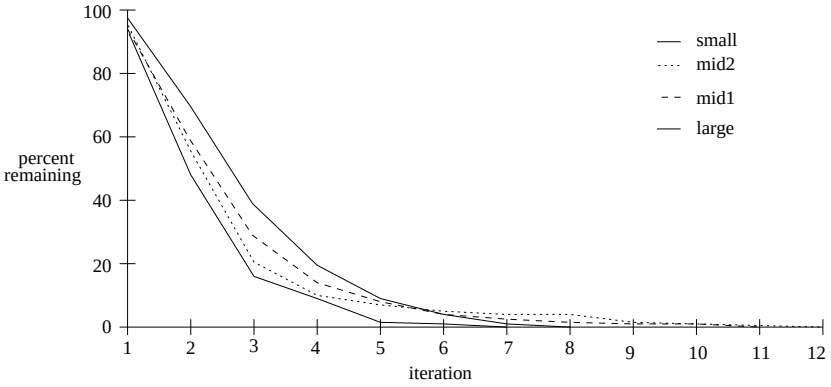

Figure 7.1: SI Refinement: Graphs Remaining

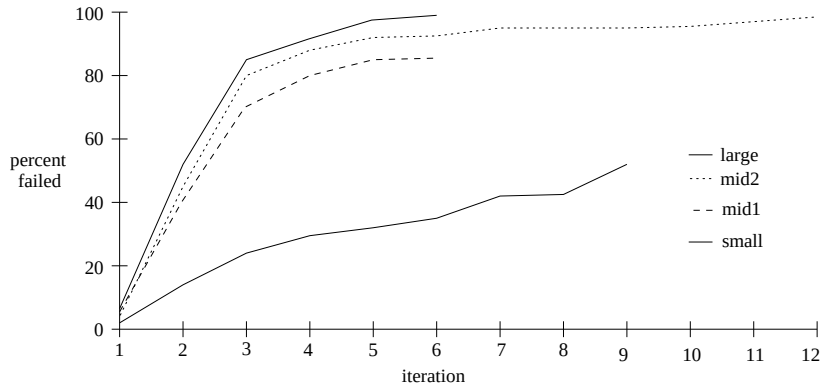

Figure 7.2: SI Refinement: Cumulative Graph Fails

In the context of a MIS knowledge base, the graphs selected for comparison are more likely to pass the SI than a random selection due to the filtering effect of the multilevel indexing. This is approximated in the experiments by the large values for $p(M)$. This is especially appropriate for the "mid1" experiment as most atoms in an organic chemistry database are either carbon or hydrogen (the latter typically ignored).

Small changes to $p(A)$ and $p(B)$ from those of Table 7.1 have only a small effect on behavior. Reducing $p(M)$ to 0.20 results in all graphs failing within 1 or 2 iterations

whereas increasing $p(M)$ to 0.60 for the "mid2" example results in as many as 17 iterations to complete all graphs and less than 50% failing refinement. Increasing $p(M)$ to 0.40 for the "large" experiment results in none of the graphs failing with only 2 to 4 iterations required until all M matrices stop changing. The preliminary experiments of Table 7.1, Figure 7.1 and Figure 7.2 are intended to represent a middle ground of typical applications.

Examining the simulation experiments individually, where processing continues until all graphs are complete at any given time an average of 70% of the processors have completed their graph. Although 30% is not an unreasonable processor utilization, it indicates that the load balancing effect of adding new graphs as others complete is an important issue. The simulations indicate that a single iteration of refinement requires some 1,000 instructions on a well-suited processor architecture whereas distributing graphs and forming the initial match matrix requires over 2,000 cycles. Combining these simulation results we conclude that it would be appropriate to bring in new graphs every 5 to 10 iterations of refinement depending on application characteristics.

Thus far we have discussed refinement iterations. As depicted in Figure 7.3, Willett et al.'s experimental data for exhaustive search in a chemical database shows that nearly all SI tests are resolved in 3 to 7 calls to refinement (each of which includes several iterations) with a few (less than 1%) requiring 20–50 calls. By bringing in new graphs during refinement iterations, we effectively load balance at each call to refinement; the housekeeping required by this approach remains an open question at this stage in our research.

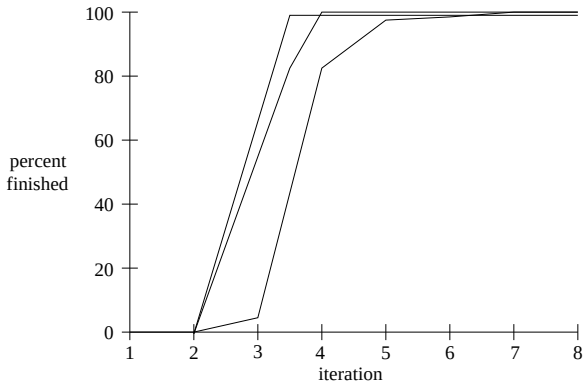

Figure 7.3: SI Calls to Refinement [Willett]

8 A Hardware Implementation in Progress

In the course of this work we have noted several characteristics of the algorithm that have influenced our design of the MISC Machine, a proposed architecture for artificial intelligence, including conceptual structure (CS) processing. In particular, 90% of simulated subgraph isomorphism (SI) refinement execution time is spent in Boolean matrix multiplication. Tailoring hardware for this operation provides significant speedup for a relatively minor hardware investment. A comparison of simulated SI execution on the MISC and MasPar MP-2 (not presented in this report) indicate that MISC would be 40 times faster on the refinement portion. It is, however, unfair to compare projected performance of a machine utilizing VLSI technology expected to be common 5 years from now with the existing commercially available MP-2. Adjusting MP-2 results to projected technology still gives

MISC an order of magnitude advantage in performance at roughly the same hardware requirements. Although the MP-2 can be configured with as many as 16 K processors, increasing the number of processors does not substantially improve performance as MIS greatly prunes the number of candidate graphs. Since the number of graphs that can be processed simultaneously without risking wasted effort is hypothesized to be $O(\lg N)$, the size of a CS knowledge base is likely to be determined more by memory capacity than number of processors for most parallel machines, including MISC. As a result of these observations, we foresee a small MISC array configured as workstation coprocessor board as an ideal platform for future CS applications. The workstations could be networked into a distributed system for extremely large knowledge bases, multiple-domain applications, and other division of labor.

9 Conclusions

We have developed a new parallelization of Ullmann's subgraph isomorphism refinement algorithm featuring multiple bits per processor and multiple graphs per processor array. The new algorithm incorporates graph storage and allocation, the formation of the candidate binding matrices in parallel, and criteria specific to conceptual structures. It avoids the potential bottleneck of consulting a concept-type subsumption table by using subsumption codes as concept-type labels. We also present a revised parallelization of multi-level indexed search that exploits the characteristics of the new refinement algorithm. Analysis shows that with these improvements the new parallelizations provide a significant speedup over prior parallelizations, make efficient use of contemporary parallel processors, are clearly superior over exhaustive parallel SI tests, and require only a small data parallel machine. Results of the refinement algorithm's behavior on parameterized synthetic graphs are also included.

A combination of empirical results and analysis show that the new method provides a significant speedup over prior parallelizations.[4] Ullmann's original implementation used the bit-parallelism of a single processor to achieve an empirical $O(n^3)$ expected-case time where $n \leq b$ (n is the number of nodes in the graph and b is the number of bits in the processor). Willett *et al.*'s first implementation uses n^2 single-bit processors to achieve $O(n^2)$ empirical expected-case time; their other methods require longer time but process a large number of graphs simultaneously. Our parallelization has $O(n^2)$ time per group of graphs being simultaneously processed. We hypothesize that with multi-level indexed search $O(\lg N)$ graphs can be compared simultaneously without unnecessary comparisons (where N is the total number of graphs in the knowledge base). In this case we would have an amortized expected-case time per graph of $O(n^2/\lg N)$ using only $O(\lg N n^2/b)$ processors. Empirical results on serial implementations of multi-level indexed search ([8] and others) indicate that a total of only $O(\lg^2 N)$ SI tests are necessary.

Although it remains to be seen if these empirical expected-case behaviors and resulting analyses hold for conceptual structure applications, the promise of fast retrieval and classification in extremely large knowledge bases with a single, small data-parallel array is exciting.

Our broad goal for future work is the parallel implementation of a conceptual structure database system on an existing parallel machine. Key steps along the way include an

[4] In all cases, the space-time product of number of processors, number of bits per processor, and expected-case execution time is $O(n^4)$ indicating effective parallelizations.

updated data-parallel implementation of the backtracking portion of Ullmann's algorithm for multiple graphs on a single array, load balancing to keep processors busy as some SI tests complete before others, and integration of the refinement SI algorithm with multi-level indexed search algorithms, including UDS. Work should also include further experiments with parameterized synthetic graph data sets and characterization of graphs in specific CS applications.

Acknowledgments

Thanks to Robert Levinson and Richard Hughey for their many comments and suggestions and for their proofreading an earlier version of this report.

References

[1] Hassan Ait-Kaci et al. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January 1989.

[2] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jurgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA, October 1992.

[3] Gerard Ellis. Efficient retrieval from hierarchies of objects using lattice operations. In *Conceptual Graphs for Knowledge Representations*, pages 274–293, New York, 1993. Springer-Verlag.

[4] Richard Hughey, Robert Levinson, and James D. Roberts. Issues in parallel hardware for graph retrieval. In *Proceedings of the 1st International Conference on Conceptual Structures*, pages 62–81, 1993.

[5] George G. Lendaris. Representing conceptual graphs for parallel processing. In *Conceptual Graphs Workshop*, 1988.

[6] George G. Lendaris. A neural-network approach to implementing conceptual graphs. In Timothy E. Nagel et al., editors, *Conceptual Structures, Current Research and Practice*, chapter 8, pages 155–188. Ellis Horwood, New York, 1992.

[7] Robert Levinson. UDS: A universal data structure. In W. M. Tepfenhart, I. P. Dide, and J. F. Sowa, editors, *Conceptual Structures: Theory and Practice*, pages 230–250. Springer-Verlag, New York, 1994. Lecture Notes in AI 835.

[8] Robert A. Levinson. *A Self-Organizing Retrieval System for Graphs.* PhD thesis, University of Texas at Austin, 1985.

[9] Robert A. Levinson and Gerard Ellis. Multi-level hierarchical retrieval. *Knowledge-Based Systems Journal*, 5(3), September 1992.

[10] John R. Nickolls. The design of the Maspar MP-1: A cost effective massively parallel computer. In *Proceedings of COMPCON*, pages 25–28, February 1990.

[11] James D. Roberts et al. Hardware for PEIRCE. In *Proceedings of the International Workshop on PEIRCE: A Conceptual Graphs Workbench*, 1992,1993.

[12] John F. Sowa, editor. *Principles of Semantic Networks.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[13] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, January 1976.

[14] Peter Willett, Terence Wilson, and Stewart F. Reddaway. Atom-by-atom searching using massive parallelism: Implementation of the Ullmann subgraph isomorphism algorithm on the distributed array processor. *Journal of the Chemical Information Computation Society*, (31):225–233, 1991.