Discrete Applied Mathematics 19 (1988) 45–63
North-Holland

GRAPH EMBEDDING IN SYNCHEM2, AN EXPERT SYSTEM FOR ORGANIC SYNTHESIS DISCOVERY*

Joseph D. BENSTOCK** and Donald J. BERNDT
Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794, USA

Krishna K. AGARWAL
Department of Computing and Information Sciences, Trinity University, San Antonio, TX 78284, USA

Received 15 October 1985
Revised 30 April 1986

Graph embedding (subgraph isomorphism) is an NP-complete problem of great theoretical and practical importance in the sciences, especially chemistry and computer science. This paper presents positive test results for techniques to speed embedding by modeling graphs with subroutines, precalculating edge tables, turning recursion into iteration, and using search-ordering heuristics.

The expert system SYNCHEM2 searches for synthesis routes of organic molecules without the online guidance of a user, and this paper examines how embedding information helps to implement the central operations of SYNCHEM2: selection, application, and evaluation of chemical reactions. The paper also outlines the architecture of SYNCHEM2, analyzes the computational time complexity of embedding and related problems in graph isomorphism and canonical chemical naming, and suggests topics and techniques for further research.

1. Introduction

Graph embedding algorithms determine whether a given *guest* graph can be embedded within a given *host* graph; some embedding algorithms in addition store the mapping(s) of the isomorphism(s) between the guest and subgraphs of the host. Such algorithms are of great theoretical and practical importance in a number of the sciences. We present positive test results for techniques to speed embedding by modeling graphs with subroutines, precalculating edge tables, turning recursion into iteration, and using search-ordering heuristics.

As a central process in the expert system SYNCHEM2 at SUNY/Stony Brook, an AI project under the direction of H. Gelernter, embedding assists in organic synthesis discovery, an important and difficult problem in the natural sciences for

* This research was supported by Eastman Kodak Co. and NASA Research Grant #NAG 3-651.

** Present address: Computer Resources International A/S, Vesterbrogade 1A, DK-1620 Copenhagen V, Denmark.

which SYNCHEM2 has generated results of interest to chemists and biochemists [7,19,20]. SYNCHEM2 searches for synthesis routes for a given organic molecule without the online guidance of a user, and uses embedding information to help select, apply, and evaluate chemical reactions. Various algorithms for substructure search, molecular pattern matching, pretransform and posttransform testing, and heuristic subgoal evaluation will be presented and discussed in the context of SYNCHEM2.

Embedding is an NP-complete problem related to important problems in graph isomorphism and canonical chemical naming that currently have unknown computational time complexity and that bear upon the deep questions of containment in the polynomial hierarchy of problem complexity classes, including P and NP. We will analyze the time complexity of these problems and the relationships between them.

Graph embedding is a central operation in all the following applications involving pattern matching and feature detection: canonical chemical nomenclature and indexing, chemical substructure searching, and synthesis discovery [2, 7, 10, 33, 36, 39, 42, 43], aspects of developmental biology [14], and modeling memory recall in cognitive psychology [7, 32]. Applications are found in electrical network design and optimization [43], image processing involving template matching and feature recognition [7, 8, 11, 16, 17, 34, 35, 37, 38, 40], and speech processing using templates and networks of phonemes [7]. Embedding techniques find widespread use in the following subfields of computer science: database management and information retrieval [6, 7, 8, 11, 30], program transformation (including synthesis, optimization, and automated programming) [2, 5, 7, 10], computer learning and knowledge representation [7, 15], analogical reasoning using graph homomorphism [12, 22], state representation and pattern recognition in game trees [2, 6, 10], as well as study of programming languages, data and control flow, and concurrency [9, 14].

The paper consists of six sections: (1) Introduction, (2) Architecture of SYNCHEM2, (3) Using embedding in SYNCHEM2, (4) Computational time complexity of embedding and related problems, (5) Techniques to speed embedding, (6) Summary and directions for further research.

2. Architecture of SYNCHEM2

Other introductions to the SYNCHEM project are found in [7, 19, 20].

2.1. Control strategy

Problem reduction by retrosynthesis

The problem reduction strategy SYNCHEM2 uses for finding synthesis routes for an input organic molecule (called the *target* molecule) is to select reactions that

retrosynthesize it into compounds that are simpler in the sense that they are commercially available or are expected to be more readily synthesizable. These generated compounds are referred to as *subgoal conjuncts* in artificial intelligence and *precursors* in chemistry.

Problem representation using AND/OR *graphs*

The search space of the problem is represented by an AND/OR *Problem Solving Graph* (*PSG*) as in Fig. 1, in which a molecule is represented by a *compound node*, that when selected for development, spawns alternative *subgoal nodes* [7, 31]. Compound nodes and subgoal nodes alternate in the PSG and are connected by reaction edges. Subgoal nodes (whose children are one or more reacting compounds) are by definition 'OR' nodes because the establishment of *any* subgoal node solves the parent compound. Compound nodes, on the other hand, are 'AND' nodes because *all* sibling compound nodes must be established to solve their parent subgoal. Available compounds are 'terminal' nodes in the PSG, and normally are not further developed. A *cycle of subgoal generation* is completed when every feasible reaction schema from the knowledge base that meets the dynamic selection criteria has been applied to a particular compound. The local tactic of SYNCHEM2 is to progressively reduce the 'distance' between leaf nodes of the PSG and a list of available compounds by selecting and applying reactions until all leaf nodes represent available compounds.

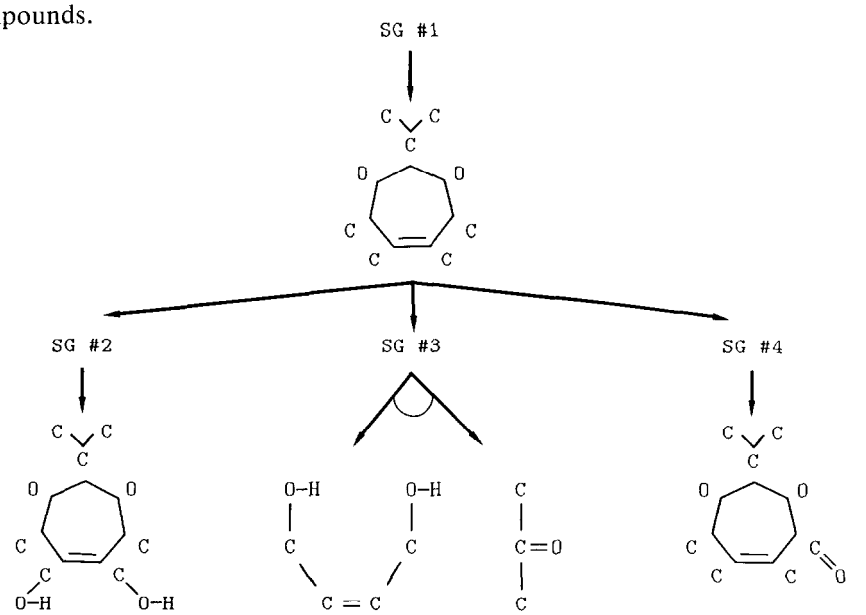

Fig. 1. A Problem Solving Graph. Adapted from [24]. The target compound at the root is developed into three 'OR' subgoal nodes, SG #2, 3, and 4. Subgoal nodes have one or more reacting 'AND' compound nodes as children. The goal of SYNCHEM2 is to develop all leaf nodes into available compounds. Note that chemical convention omits depiction of hydrogen atoms on carbons when they clutter drawings; thus all carbons have four edges, although in drawings some edges may be implicit.

Heuristic evaluation functions

An exhaustive search of the problem space is computationally infeasible ('combinatorially explosive') for any but the most simple syntheses. As the search generates and evaluates subgoals, it must correspondingly limit the growth of the PSG and prune it. Thus the search is guided by figures of merit, which are heuristically computed values used to predict the success of program alternatives. The three types of merit, compound, reaction, and subgoal, are integers in the range $[0, \ldots, 100]$. Merits are computed using information about the current chemical environment of a given node in the PSG to adjust the default values from the knowledge bases [20, 24].

Compound merits are used to predict the likelihood of successfully completing a pathway from a particular compound all the way down to available compounds and reflect the maximum subgoal merit of subgoal children. (Available compounds are assigned a merit of 100 on the scale $[0, \ldots, 100]$.) *Reaction merits*, working in the opposite direction, estimate the cost of successfully reacting compounds to solve their parent subgoal. Reaction merits are a function of the dynamic parameters *ease*, *yield*, and *confidence*, which are also rated $[0, \ldots, 100]$. The default parameter values associated with each reaction are estimates of usefulness under standard conditions, supplied by chemists who have worked on the knowledge bases. The *subgoal merit* of each subgoal is a dynamic evaluation function of its constituent child compound merits, its reaction merit, and after updating, the subgoal merits of descendants. After each cycle of subgoal generation, affected subgoal and compound merits throughout the PSG are updated by recomputing them iteratively, starting from newly generated subgoals and moving up the PSG to the target compound at the top. The merit of a subgoal is generally somewhat worse than the merit of its worst unsolved child compound.

Effort distribution

The *Effort Distribution* algorithm dynamically apportions search time for each compound node so that the program can develop the search space rationally. Search status flags for compound nodes (open, solved, just solved, stuck, available, unsolvable) are updated and distributed throughout the PSG after each cycle of subgoal generation [24]. SYNCHEM2 has the capability of allowing the user to control the degree to which the search concentrates on syntheses that are either inexpensive, short, diverse, different from those in the literature, or have well-understood chemistry, etc.

2.2. Knowledge bases

The knowledge bases consist of four available-compounds libraries (two of which are currently active), and four reaction libraries.

Compound representation and libraries

Davis, Boivie, and Agarwal have designed *SLINGs* (SYNCHEM Linear Input Graphs) for SYNCHEM2 [13], a compact linear canonical representation for molecules, for user input/output as well as internal storage and communication between program modules. Compounds are also represented by the *Extended Topological Representation* (*XTR*), a data structure that includes a connection matrix and that computes and stores additional information on demand, such as atomic degree, number of cycles, and so on.

SYNCHEM2 has a knowledge base of about 10,000 available compounds, most of which are from the *Aldrich Catalog-Handbook of Organic and Biochemicals.*

Reaction representation and libraries

SYNCHEM2 selects reactions from about 1200 reactions in its knowledge base by determining the *attributes* of a compound and using these attributes as keys to reactions in the knowledge base. Attributes are characteristic chemical features, functional groups, or patterns of infeasible structures. The main *reaction library* is divided into 30 currently active chapters; each chapter is associated with a unique attribute and a chapter screening test. A reaction with more than one attribute is arbitrarily assigned to a chapter. Reactions within chapters are arranged in order of decreasing probable utility.

Reactions are stored in the reaction library as reaction schemata, which may be viewed as graph rewriting rules. A reaction schema consists of a *goal pattern* graph and its retrosynthetic derivative, the *subgoal pattern* graph, which have isomorphic node sets. The goal pattern or the subgoal pattern may be disconnected. To determine whether a reaction may be applied to a compound, i.e., a reaction schema ap-

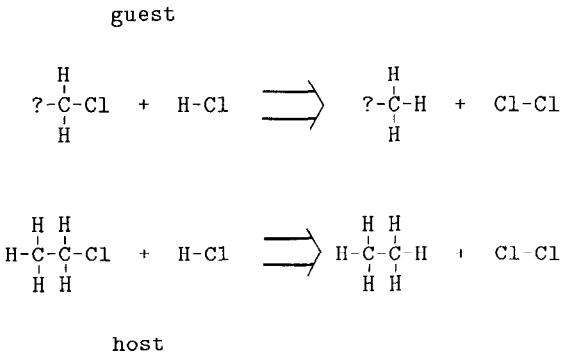

Fig. 2. A reaction schema and application. The guest graph in the reaction schema on top embeds in the host graph, showing that the reaction schema can be applied to the host graph, as on the bottom. The left and right sides of the reaction schema are called the goal pattern and subgoal pattern respectively. The fragment variable nodes labeled '?' correspond to subgraphs, called fragments, in the host and its retrosynthetic derivative.

plied to a host graph, SYNCHEM2 considers the goal pattern graph as the guest graph and searches for embeddings of it within the host graph. Fig. 2 shows a guest graph that embeds in a host graph, which indicates that the reaction schema can be applied to the host graph.

3. Using embedding in SYNCHEM2

3.1. Definition of chemical graph embedding

Because a goal pattern graph stands for a class of compounds, it is convenient to use *fragment variable* nodes to stand for classes of compound substructures.

Definition. A *fragment variable* node (or variable node, for short) is a guest graph node that, according to its label, maps to exactly one of the following attribute classes: any element at all, any element except hydrogen, any halogen atom, or an alkyl carbon. Similarly, a variable edge maps to either a single, double, or resonant bond.

Definition. A *constant* node is the opposite of a variable node and is labeled by a definite element.

Definition. A *chemical graph embedding* of a guest graph in a host graph is a 1-to-1 mapping of guest nodes into host nodes such that: (1) adjacency is preserved, (2) node and edge labels are preserved, (3) stereochemical orientation is preserved, and (4) images of variable nodes are part of connected subgraphs in the host corresponding to their label and containing no images of constant nodes [39].

These subgraphs in the host and its retrosynthetic derivative are called *fragments* and are an instance of the attribute class the labeled variable node represents. Intuitively, fragments are what is 'left over' in the host graph after images of constant nodes are accounted for. See Fig. 3.

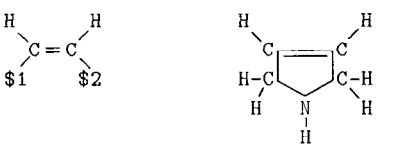

Fig. 3. Variable nodes and fragments. From [39]. The guest graph embeds in the host graph. The hydrogen and carbon atoms in the guest map to the obvious atoms in the host; the two variable nodes labeled '$' both correspond to the same fragment in the host graph, an eight node subgraph constituting the rest of the host. '$N', where N is an integer, is represented as R_N in chemistry literature, a molecular structure variable.

3.2. Selecting reactions

Attribute determination

In order for the program to know which types of reactions to select for a particular compound, it must first determine the attributes of that compound. It uses SUBSRCH, a procedure designed by Boivie, for finding all attributes within a compound from a set of predefined substructures. SUBSRCH operates by conducting a binary search of the attribute table and comparing table entries to neighborhoods recursively 'grown' around each compound atom.

SUBSRCH operates by examining the neighborhood of each atom in a compound, using breadth-first search (BFS, see Section 5.3), and identifying each neighborhood as: (1) part of a potential attribute warranting further search, (2) an instantiation of an attribute, or (3) a substructure not warranting further search, unlike any in the table (in which case SUBSRCH backtracks). The SUBSRCH procedure is more efficient than general embedding algorithms for locating sets of predefined substructures and identifying attributes that are: (1) sensitive and that interfere with reactions, (2) readily available or readily synthesizable, or (3) chemically invalid [10].

Screening reactions

Once SUBSRCH has determined the attributes of a compound, SYNCHEM2 conducts screening *pretransform tests* on them to select reactions from the reaction library. Pretransform tests for a reaction use chemist-specified combinations of conditions of the following four types: the compound (can't/must) have (any/all) of the attributes on a list associated with the reaction. An example of a pretransform test would be that a Grignard reaction on lactones or aldehydes can't have any carboxyl or nitrile groups, and must have any secondary or tertiary alcohol group.

3.3. Applying reactions

If all the pretransform tests for a reaction on a compound are satisfied then SYNCHEM2 attempts the following embedding using Sanders's MATCH algorithm: it takes the goal pattern graph of the reaction schema as the guest graph, and the graph representing the compound as the host graph, and generates maps for all embeddings of the guest within the host [39].

The difference between SUBSRCH and MATCH is that, while SUBSRCH locates a 'root' node for each instance of an attribute within the host, MATCH determines the exact location and scope of the guest graph within the host graph, generating a 1-to-1 map from nodes of the guest graph to nodes of the host graph. Because a reaction can take place at several sites in a molecule, but sometimes such chemistry is redundant, MATCH has the option of generating only one embedding on any given procedure call.

MATCH is called recursively with a guest graph and a host graph and the most recently matched pair of nodes. It then finds all possible extensions of existing par-

tial embeddings using a *dual depth-first search* (*DDFS*), and returns the maps of all complete embeddings. The pseudocode for the MATCH algorithm is given below. Note that a *frond* or *back edge* is an edge that completes a cycle in a graph traversal [41].

The Dual Depth-First Search MATCH Algorithm. Node a in the guest already matches node α in the host and MATCH attempts to extend the embedding to nodes b and β.

```
procedure MATCH (a, α: nodes);
  begin
    for each unmarked neighbor b of a do
      begin
        mark b;
          for each unmarked neighbor β of α do
            begin
              if (node labels of b and β match and
                  fronds of b and β match and
                  edge labels of edges (a, b) and (α, β) match) then
              begin
                mark β; image(b) := β;
                  if (all nodes of guest graph are marked)
                    then record embedding
                  else call MATCH(b, β);
                  unmark β; image(b) := 0;
              end; {if nodes, fronds, and edges match}
          end; {for each unmarked neighbor of α do}
        unmark b;
    end; {for each unmarked neighbor of a do}
  end; {procedure MATCH}
```

If the guest graph of the reaction can be embedded within the host graph representing the compound, then SYNCHEM2 applies the reaction to the host graph using the procedure SUBGENR, producing a *subgoal* graph consisting of one or more compounds, as in Fig. 2.

3.4. Evaluating reactions

After applying a reaction, SYNCHEM2 canonically names the generated compounds and conducts a series of *posttransform tests*, using SUBSRCH and MATCH, to evaluate the application to the particular compound in the chemical environment existing at that node in the PSG. The default values for the ease, yield, and confidence parameters of the reaction are adjusted at this stage and combined to calculate the adjusted reaction merit. Posttransform tests can cause a subgoal to be rejected (e.g.,

due to an invalid attribute having excessive bond strain), the reaction procedure to be modified (e.g., changing the reagent), or protection procedures to be specified for sensitive groups. In each case, the specific site of the embedding and the current chemical environment are considered.

Posttransform tests may be concerned with, for example, electronic and stereochemical effects, steric hindrance, proximity of functional groups to the reaction site, ring structures and sizes, alkynes in a ring, allenes in a small ring, relative stabilities of carbonium ions, leaving groups, migratory aptitudes, the largest attribute in the molecule, symmetry properties of the goal and subgoal molecules, etc. [24].

After posttransform testing, compound merits of newly generated compounds are evaluated using several criteria, calling SUBSRCH to identify significant attributes and heuristic procedures to estimate compound merit. Compound merit is computed in a procedure that considers the number and bonding of carbon atoms, the ratio of number of functional groups to number of carbon atoms, ring system complexities, and metastable oxidation states. In general, simpler compounds are rated preferentially.

4. Computational time complexity of embedding and related problems

4.1. The polynomial hierarchy

A program for a deterministic Turing machine (DTM) consists of: (1) a finite state read/write head; (2) an infinite tape of cells; (3) a finite set Σ of input tape symbols; (4) a finite set Q of states, including a start state q_0 and two halt states q_Y and q_N, and (5) a transition function that directs the tape head in a given state at a given cell to read the symbol in the cell, to write a symbol over it (perhaps the same one), and to move either left or right. A program for a nondeterministic Turing machine (NTDM) has a transition function that directs the tape head in a given state at a given cell to read the cell, and to nondeterministically choose a pair, consisting of a symbol to write and a head movement, from a set. A nondeterministic program 'guesses' a solution of a decision problem by choosing the right such pair at each step, and then 'verifies' whether it is actually a solution. An example of this concept is the following informal description of an NDTM for graph isomorphism: it guesses a mapping between the vertex sets of two graphs, and then proceeds to verify whether the mapping is 1-to-1, onto, and adjacency-preserving.

An input string $x \in \Sigma^*$ is a finite sequence of symbols and we say (for a DTM or an NDTM) that *program M accepts x* iff it begins at the first tape cell, reads the string x, and halts in the accepting state q_Y. The *language accepted or recognized by program M* is

$$L_M = \{x: x \in \Sigma^* \text{ and } M \text{ accepts } x\}.$$

There is a natural correspondence between recognizing languages and solving decision problems.

A polynomial time program is one in which the number of computation steps is bounded by a polynomial function of the input length. The complexity class P is defined to be

$$\mathrm{P} = \{L: \exists \text{ a polynomial time DTM program recognizing } L\}.$$

The class NP is defined to be

$$\mathrm{NP} = \{L: \exists \text{ a polynomial time NTDM program recognizing } L\}.$$

The class co-NP is defined to be

$$\text{co-NP} = \{L: \exists \text{ a polynomial time NTDM program recognizing } \Sigma^* - L\}.$$

The P = NP problem is open because, while verifying solutions can be done in polynomial time for many problems, it is not known whether guessing solutions can also be done in polynomial time for those problems. To rate the relative complexity of languages, it is useful to define a polynomial transformation between languages L_1 and L_2 as a function f computable by a recursive (always halting) polynomial time DTM program such that $x \in L_1$ iff $f(x) \in L_2$. We then write $L_1 \leq_{\mathrm{m}} L_2$, and say that L_1 *many-one Turing reduces* to L_2. This expresses the relation that if the 'harder' language L_2 is in P, then so is L_1. '$\leq_{\mathrm{m}}$' is a partial order. L_1 and L_2 are *polynomial time equivalent* iff $L_1 \leq_{\mathrm{m}} L_2$ and $L_2 \leq_{\mathrm{m}} L_1$. We then write $L_1 \equiv_{\mathrm{m}} L_2$. The class NP-complete (NPC) consists of the 'hardest' problems in NP:

$$\mathrm{NPC} = \{L: L \in \mathrm{NP}, \text{ and } \forall L' \in \mathrm{NP}, L' \leq_{\mathrm{m}} L\}.$$

A polynomial algorithm found for an NPC problem would be translatable to a polynomial algorithm for *any* NP problem, by direct construction using the reducing polynomial transformation. It is felt that there is a strong possibility that NPC problems are intractable, with optimal solutions taking an exponential amount of time, although good approximate solutions may sometimes be found in polynomial time.

We can show a given problem to be NPC by showing that it contains a known NPC problem as a subcase with a recursive (always halting) polynomial time transformation reducing the known problem to the given one. (Intuitively, if the new, harder, problem were polynomially solvable, then its known NPC subcase would be polynomially solvable as well). Ladner showed that if $\mathrm{P} \neq \mathrm{NP}$, then there exists a dense hierarchy of problems strictly between P and NPC [18]. The class NP-Incomplete is defined to be

$$\mathrm{NPI} = \mathrm{NP} - (\mathrm{P} \cup \mathrm{NPC}).$$

Fig. 4 depicts the containments in the polynomial hierarchy under this assumption that $\mathrm{P} \neq \mathrm{NP}$, in which case NPI is nonempty.

Accounts of Turing machines and complexity classes are found in [3, 18, 21].

4.2. Time complexity of subgraph isomorphism

Embedding is NP-complete for chemical and general graphs. Compounds can be modeled by graphs with labeled edges and nodes. Thus the problem of determining whether one compound is 'part' of another can be restated as that of determining, in a pair of labeled graphs, whether a graph G can be embedded in a graph G', i.e., whether G is isomorphic to a subgraph of G'. This is the labeled variant of the *Subgraph Isomorphism* problem (SGI).

To see that SGI is NPC, we use the standard technique of noting that it is NP (because it is possible to verify a solution in polynomial time), and that it contains a known NPC problem, CLIQUE, as a subcase. Intuitively, this shows that SGI is in the class NPC because it is 'at least as hard' as its subcase, known to be 'at least as hard' as any problem in NP. CLIQUE asks whether a graph contains a complete subgraph of size $\geq k$. CLIQUE is a subcase of SGI because an algorithm for SGI can be iteratively called as a subroutine to attempt to embed the complete graphs of size $\geq k$ in CLIQUE's input graph. That iteration takes polynomial time. We conclude that CLIQUE $\leq_m$ SGI, and that SGI is NPC. A polynomial reduction from unlabeled graphs to graphs with only one label shows that SGI $\leq_m$ labeled SGI. Thus the chemical application we are interested in is modeled by labeled SGI, which is NPC.

4.3. Relationship of SGI to graph isomorphism and chemical naming

The *Graph Isomorphism* problem (GI) asks whether a graph G is isomorphic to a graph G'. (SGI asks whether there exists a 1–1 map between the graphs, while GI asks whether there exists an onto 1–1 map between the graphs.) The isomorphisms of a graph onto itself are called *automorphisms* or *symmetries*. In synthesis discovery embedding, nontrivial automorphisms and interactions between goal pattern, subgoal pattern, and host graphs often generate redundant compounds and waste the substantial time required to canonically name, store, evaluate, and develop them. Agarwal discusses several heuristics for eliminating redundant maps (some before generation and some after) [2], but notes that the heuristics bear further unification and extension. Because heuristics to speed an algorithm can themselves suffer from combinatorial explosion, applications must be time analyzed carefully.

SGI and GI algorithms can be used as subroutines to help solve the chemical canonical naming problem, which is of central concern whenever dealing with a chemical database. The graph naming problem (GRAPHNAME) determines a canonical name for a compound. SYNCHEM2 uses the SLING representation and algorithms [13]. The Morgan algorithm is another popular and effective system [29].

4.4. Time complexities of graph isomorphism and chemical naming

The graph isomorphism problem is of considerable theoretical interest because,

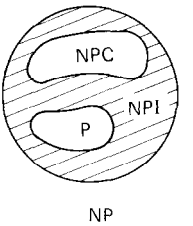

Fig. 4. The polynomial hierarchy of complexity classes, assuming $P \neq NP$. From [18, p. 154].

although it models many fundamental problems, its exact complexity is not currently known. A polynomial algorithm for SGI could be easily modified to polynomially solve GI, so $GI \leq_m SGI$, but GI has not been shown to be NPC. The problems Ladner showed to be NPI under the assumption $P \neq NP$ are 'artificially constructed' diagonalized problems, but determination of graph isomorphism and integer primality are two naturally occurring candidates for membership in NPI. Furthermore, since there is no known efficient test for determining non-isomorphism of graphs, only failure of isomorphism tests, it is not known whether GI is in co-NPC. Thus the complexity of GI bears on the deep questions of containment among the complexity classes P, NP, co-NP, and the rest of the polynomial hierarchy.

Indications that GI may be in the class NPI are:

(1) The failure of current NPC proof techniques to show the membership of GI in NPC, as opposed to the thousands of NP problems that have been shown to be in NPC.

(2) The polynomial time equivalence of the existence and enumeration GI problems, which is at variance with some NPC problems [18] (although the polynomial equivalence of existence and enumeration variants of many NPC problems is currently unknown [23,44]). The enumeration variant of GI asks how many isomorphisms exist between a pair of graphs.

Graph isomorphism can be tested in polynomial time for restricted classes of graphs, such as trivalent or planar [18,25]. Luks showed in 1980 that isomorphism for graphs of bounded degree can also be tested in polynomial time [27]. An interesting question is whether bounding the degree of guest and/or host graphs makes SGI also solvable in polynomial time, and if it does, whether Luks's $O(n^6)$ algorithm could be made practical or improved for chemical graphs of the size current chemistry programs may encounter (about 50 vertices). Chemical graphs naturally have bounded degree, which is usually four (for elements such as carbon), and certainly less than eight.

Luks's technique exemplifies the group theoretic approach to GI, which seeks to determine a set of generating permutations for the automorphism group of a graph. The other main approach, exemplified by the work of Miller [28], is topological, and seeks to embed a graph onto a surface of minimal genus and then dissect the surface into planar components. The relationship between the two approaches is not known. There are a number of combinatorial and group-theoretic problems (some

concerning group generating sets) that are polynomial time equivalent to GI and we say that they constitute the class of *isomorphism complete* problems [25].

A polynomial time solution to GRAPHNAME would polynomially solve GI, but an interesting fact is that it is not currently known whether a polynomial time solution to GI could be extended to polynomially solve GRAPHNAME [25]. Thus we know that GI $\leq_m$ GRAPHNAME, but not whether GRAPHNAME $\leq_m$ GI.

5. Techniques to speed embedding

5.1. Modeling graphs with subroutines

Unrolling loops and making fewer array references and procedure calls can significantly shorten execution time of a program. Benstock has developed a technique to do this that models or *mirrors* each guest graph with a short subroutine in source code. A simple algorithm creates this code out of an adjacency representation for a guest graph. It apportions a 'for' loop in source code for each node of a guest graph, so that some guest graph information is implicit in the body of that loop. The technique is similar to that of transforming the labeled graphs of transition diagrams into code for lexical analyzers of compilers, as in [5]. There is a 1-to-1 map from a graph node to a 'for' loop that searches the node's neighbors to extend the embedding. The following operations are saved: procedure calls and array references for node, edge, and frond matching are either eliminated (in the frequent situation that any label will match, or that no fronds exist) or reduced to references to constants.

The pseudocode below for the mirroring technique shows the 'for' loop of one node with the 'for' loop of its successor (in DFS traversal) nested within it.

Pseudocode for Mirroring Technique

```
for each unmarked neighbor β of α do
  begin
    if (node labels of b and β match and
        fronds of b and β match and
        edge labels of edges (a, b) and (α, β) match) then
      begin
        mark β; image(b) := β; α := β;
        for ···
          ⋮
        unmark β; image(b) := 0; α := image(a);
      end; {if nodes, fronds, and edges match}
  end; {for each unmarked neighbor of α do}
```

The mirroring technique reduced execution time of MATCH by a factor of 174 when attempting to embed the perester pattern graph of Fig. 6 in each of the five host graphs of Fig. 5. It reduced execution time by a factor of 35 when embedding a benzene ring in itself. The mirroring technique is suitable for stable databases requiring fast pattern matching and with storage enough for a subroutine corresponding to each pattern in the database.

5.2. *Eliminating recursion and precalculating tables*

We have developed an embedding algorithm GREMBED (for Graph Embed) that turns the recursion of MATCH into iteration, and uses a precalculated edge table of the guest graph to control the search, instead of using direct DFS of the guest graph. It is efficient in the context of SYNCHEM2 to precalculate edge tables a single time rather than searching guest graphs anew each time (with marking, unmarking, querying, and backtracking operations) since there are only about 1200 reaction schemata, and edge tables of their guest graphs would be consulted repeatedly [1].

These modifications made GREMBED about two to three times faster than MATCH, in our tests which used the five host graphs shown in Fig. 5 and the five guest graphs shown in Fig. 6. We attempted embeddings, using MATCH and GREMBED, for each of the 25 possible pairs of host and guest graphs. The total execution times in milliseconds for finding all embeddings are shown below in Table 1, along with the MATCH/GREMBED ratios of those times. They are grouped by host and guest graph. The average ratio (speedup factor) is 2.6. The figures for finding just one embedding (existence) are shown in Table 2. The average ratio for this case is 3.3. Both algorithms were written in VAX/VMS PL/I and run on a MicroVAX-II. For simplicity, these tests did not consider stereochemistry and the heuristics for reducing the generation of redundant graphs.

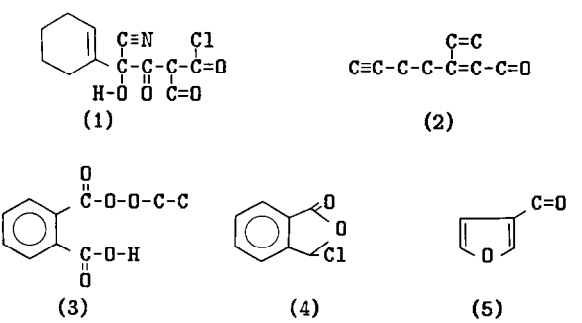

Fig. 5. Five host graphs used to compare MATCH and GREMBED. Unlabeled vertices represent carbon atoms by convention. Hexagonal rings with a circle in the center represent benzene rings where each of the six edges is labeled as a resonant bond. A carbon in a benzene ring has room outside the ring for only one single bond. (1) arbitrary molecule, (2) 3-vinyl-2-hepten-6-ynal, (3) OO-ethyl hydrogen monoperoxphthalate, (4) 3-chlorophthalide, (5) furoyl.

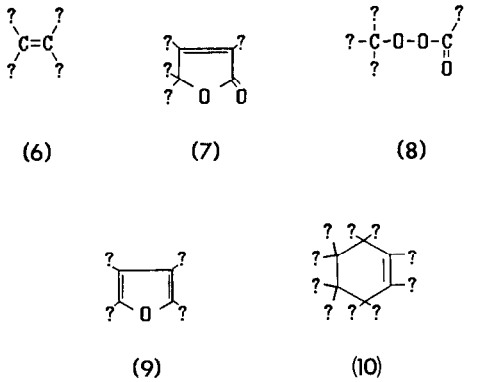

Fig. 6. Five guest graphs used to compare MATCH and GREMBED. (6) alkene, (7) lactone, (8) perester, (9) furan-3-carboxaldehyde, (10) cyclohexene.

There are 8 and 32 embeddings of alkene and cyclohexene respectively in the first host graph, 16 embeddings of alkene in the second host, 6 embeddings of perester in the third host, no embeddings in the fourth host, and 16 and 2 embeddings of alkene and furan respectively in the fifth host.

We also compared GREMBED and the canonical naming algorithm of SYNCHEM2 for testing graph isomorphism, and found GREMBED to be about 24 times faster on the average when testing each of the host graphs of Fig. 5 for isomorphism with itself. GREMBED was 70 times faster than canonical naming for the molecule below.

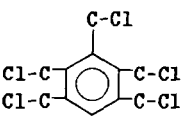

Table 1. Times (in milliseconds) using MATCH and GREMBED to find **all embeddings** of the 25 host-guest pairs. Summed and grouped by host and guest graph, and showing the MATCH/GREMBED ratios of the times.

Name	MATCH	GREMBED	Ratio
arbitrary molecule	1055	410	2.57
3-vinyl-2-hepten-6-ynal	665	210	3.17
OO-ethyl hydrogen monoperoxphthalate	825	315	2.62
3-chlorophthalide	520	200	2.60
furoyl	315	160	1.97
alkene	740	360	2.06
perester	540	160	3.38
lactone	465	205	2.27
furan	525	175	3.00
cyclohexene	1110	395	2.81

Table 2. Times (in milliseconds) using MATCH and GREMBED to find the first embedding (i.e., determine **existence** of embedding) of the 25 host-guest pairs. Summed and grouped by host and guest graph, and showing the MATCH/GREMBED ratios of the times.

Name	MATCH	GREMBED	Ratio
arbitrary molecule	885	255	3.47
3-vinyl-2-hepten-6-ynal	665	210	3.17
OO-ethyl hydrogen monoperoxphthalate	795	260	3.06
3-chlorophthalide	535	200	2.68
furoyl	325	105	3.10
alkene	575	200	2.88
perester	525	165	3.18
lactone	480	176	2.74
furan	515	165	3.12
cyclohexene	1130	275	4.11

This indicates that GI algorithms are to be preferred to GRAPHNAME algorithms in applications in which approximately one hundred graphs or fewer are to be pairwise compared and duplicates not named.

5.3. Search-ordering heuristics

Comparisons between search-ordering heuristics should analyze, among other factors, the following graph parameters: the expected number of embeddings (so we can seek to primarily accept or reject matches), the cycle structure and general graph topology, and the absolute and relative sizes of the guest and host graphs. For the sake of efficiency, attempts should be made to match distinguishing structural features, such as attributes and fronds, as soon as possible in an embedding search.

Time might be saved in SYNCHEM2 embedding searches if substructures in the guest graph were matched as units to their images in the host graph, using information already gathered by SUBSRCH. This *attribute matching* would also be suitable for applications with guest graphs that had many predefined substructures.

MATCH is passed matched initial root nodes by SUBSRCH, and we have found it to be highly sensitive to the choice of pattern graph root node. We have found that choosing a guest root node within a distinctive region of the guest (allowing rapid rejection of nonembeddings) can speed embedding by a factor of as much as four.

Breadth-first search (BFS) is the graph traversal method that visits the root, then all nodes at distance one from the root, then all nodes at distance two, etc., and never revisits nodes. Thus it crosses every edge exactly twice, counting backtracking upon finding an already visited node. Depth-first search (DFS) is the graph traversal method that visits the unvisited node of maximum depth at each step. BFS tends to waste storage space, since desired nodes within the graph lie too far from the root for most applications, and BFS also tends to manifest embedding failures more

slowly than DFS. Embedding using DFS is usually faster than BFS, perhaps because it covers cycles and other structural features sooner. BFS is faster than DFS is in some instances, however, and further investigation is needed to determine the conditions under which this obtains [1, 26]. Note that a computation to determine whether DFS or BFS is preferable in a given instance would add its own costs to the algorithm.

One possible heuristic to increase efficiency visits nodes in the order of *highest-degree-first* [1]. This, in effect, causes the algorithm to traverse chains and cycles of carbons early. We found that this particular traversal heuristic made no significant difference in embedding times.

A *fragment code* is a set of predefined symbols standing for a molecular substructure such as a ring or functional group. Time might be saved if SYNCHEM2 used a fragment code to represent substructures, as is the practice in many chemical databases. However, we feel this refinement would have only a small effect, which might even be negated by the added time spent expanding the fragment symbols, e.g., as must be done when rings are broken.

6. Summary and directions for further research

Graph embedding is a widely applied process that needs heuristic and theoretical improvements to push upward the threshold of combinatorial explosion. Our mirroring technique, which models graphs with procedures, makes embedding faster by a factor between one and two orders of magnitude. The mirroring technique is suitable for stable databases needing fast pattern matching. We have eliminated recursion and used a precalculated edge table in the general embedding algorithm of SYNCHEM2, which made embedding faster by a factor of about three in 25 test cases. It will be useful to determine the relative contributions of recursion elimination and the precalculated table, and the reasons why BFS is occasionally faster than DFS.

In addition to graph-theoretical heuristics, a chemical context allows chemical heuristics to speed embedding searches and a wide range of possible suitable heuristics is open for investigation, including the suggested 'attribute matching'. A statistical profile of SYNCHEM2 execution is needed to determine the distribution and structure of chemical graphs most frequently input to MATCH. A preliminary count indicates that embeddings determined by SUBSRCH are rejected by MATCH 60% of the time in typical SYNCHEM2 runs.

Further investigation is needed into other possible embedding heuristics and their costs, as well as the costs of determining when they are favored. Towards this end, we have developed an embedding program and a driver program above it that can accept arbitrary traversal sequences, and can output traversal and embedding information at variable levels of detail. A particularly useful investigative tool would be a graphics program that displayed traversals through graphs at variable speeds using

highlighting or color, allowing for concrete experience of, and experimentation with, different embedding strategies on different classes of graphs.

Acknowledgements

We wish to thank Prof. H. Gelernter and Dr. G. Miller for suggestions and helpful discussions, and Noëlle Benstock for assistance with the diagrams.

References

[1] K.K. Agarwal, Graph transformation and canonization algorithms, Ph.D. Thesis, State Univ. of NY at Stony Brook (1976).
[2] K.K. Agarwal, D.P. Agrawal and M.A. Lassner, Subgraph identification using associative techniques with applications to information science and chemical structure investigation, NSF Report, Wayne State Univ., College of Engineering, Detroit, MI (1982).
[3] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Algorithms (Addison-Wesley, Reading, MA, 1974).
[4] A.V. Aho, J.E. Hopcroft and J.D. Ullman, Data Structures and Algorithms (Addison-Wesley, Reading, MA, 1983).
[5] A.V. Aho, R. Sethi and J.D. Ullman, Compilers: Principles, Techniques, and Tools (Addison-Wesley, Reading, MA, 1986).
[6] R.B. Banerji, Artificial Intelligence: A Theoretical Approach (North-Holland, Amsterdam, 1980).
[7] A. Barr, P.R. Cohen and E.A. Feigenbaum, eds., The Handbook of Artificial Intelligence (William Kaufmann, Los Altos, CA, 1982).
[8] H.G. Barrow, A.P. Ambler and R.M. Burstall, Some techniques for recognizing structures in pictures, in: S. Watanabe, ed., Frontiers of Pattern Recognition (Academic Press, New York, 1972) 1–30.
[9] L. Bic, Processing of semantic nets on dataflow architectures, Artificial Intelligence 27 (1985) 219–227.
[10] R.H. Boivie, Heuristic search guidance in SYNCHEM2, Ph.D. Thesis, State Univ. of NY at Stony Brook (1977).
[11] J.M. Brayer and K.S. Fu, Some multidimensional grammar inference methods, in: C.H. Chen, ed., Pattern Recognition and Artificial Intelligence (Academic Press, New York, 1976) 29–60.
[12] J.G. Carbonell and S. Minton, Metaphor and common sense reasoning, in: J.R. Hobbs and R.C. Moore, eds., Formal Theories of the Common Sense World (Ablex, Norwood, NJ, 1985) 405–426.
[13] H.W. Davis, Computer representation of the stereochemistry of organic molecules, in: Interdisciplinary Research Series 23 (Birkhauser, Basel, 1976).
[14] H. Ehrig, M. Nagl and G. Rozenberg, eds., Graph-Grammars and Their Application to Computer Science, 2nd International Workshop, 1982, Haus Ohrbeck, Germany, Lecture Notes in Comp. Sci. Vol. 153 (Springer, Berlin, 1983).
[15] N.V. Findler, ed., Associative Networks: The Representation and Use of Knowledge by Computers (Academic Press, New York, 1979).
[16] E.C. Freuder, Structural isomorphism of picture graphs, in: C.H. Chen, ed., Pattern Recognition and Artificial Intelligence (Academic Press, New York, 1976) 248–256.
[17] K.S. Fu, Syntactic Methods in Pattern Recognition (Academic Press, New York, 1974).
[18] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).

[19] H.L. Gelernter, N.S. Sridharan, A.J. Hart, S.C. Yen, F.W. Fowler and H.-J. Shue, The discovery of organic synthesis routes by computer, in: Topics in Current Chemistry 41(1) (Springer, Berlin, 1973) 114–130.

[20] H.L. Gelernter, A.F. Sanders, D.L. Larsen, K.K. Agarwal, R.H. Boivie, G.A. Spritzer and J.E. Searleman, Empirical explorations of SYNCHEM, Science 197 (1977) 1041–1049.

[21] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation (Addison-Wesley, Reading, MA, 1979).

[22] E.B. Hunt, Artificial Intelligence (Academic Press, New York, 1975).

[23] D.B. Johnson and S. Kashdan, Lower bounds for selection in $X+Y$ and other multisets, J. ACM 25(4) (1978) 556–570.

[24] M.A. Jones, The organization and design of SYNCHEM2, Tech. Report No. 82/038, State Univ. of NY at Stony Brook (1982).

[25] C.M. Hoffmann, Group-Theoretic Algorithms and Graph Isomorphism, Lecture Notes in Comp. Sci. 136 (Springer, Berlin, 1982).

[26] M.A. Lassner, Graph embedding algorithms and their applications, Ph.D. Thesis, Wayne State University, College of Engineering (1981).

[27] E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, in: Proceedings 21st IEEE Symp. on Foundations of Comp. Sci., Rochester (1980) 42–49.

[28] G.L. Miller, Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus, Information and Control 56(1/2) (1983) 1–33.

[29] H.L. Morgan, The generation of a unique machine description for chemical structures – a technique developed at Chemical Abstracts Service, J. Chemical Documentation 5(2) (1965) 107–113.

[30] R.R. Muntz, The WELL system: a multi-user database system based on binary relationships and graph-pattern-matching, Information Systems 3 (1978) 99–115.

[31] N.J. Nilsson, Problem-solving Methods in Artificial Intelligence (McGraw-Hill, New York, 1971).

[32] D.A. Norman and D.E. Rumelhart, Explorations in Cognition (Freeman, San Francisco, CA, 1975).

[33] L.J. O'Korn, Algorithms in the computer handling of chemical information, in: R.E. Christoffersen, ed., Algorithms for Chemical Computation (American Chemical Society Symposium Series Vol. 46, Washington, DC, 1977) 122–148.

[34] T. Pavlidis, Linear context-free graph grammars, J. ACM 19(1) (1972) 11–22.

[35] J.F. Pfaltz and A. Rosenfeld, Web grammars, in: 1st Proc. IJCAI, Washington, DC (May 1969) 609–619.

[36] R.C. Read and D.G. Corneil, The graph isomorphism disease, J. Graph Theory 1 (1977) 339–363.

[37] A. Rosenfeld, Picture automata and grammars: An annotated bibliography, in: Proc. Symp. Comput. Image Process. Recognition 2 (Univ. of Missouri at Columbia, Aug. 1972).

[38] A. Rosenfeld, Picture processing: 1977, in: Computer Graphics and Image Processing 7 (1978) 211–242.

[39] A.F. Sanders, Some applications of graph theory, Ph.D. Thesis, State Univ. of NY at Stony Brook (1976).

[40] A.C. Shaw, Picture graphs, grammars and parsing, in: S. Watanabe, ed., Frontiers of Pattern Recognition (Academic Press, New York, 1972) 491–510.

[41] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1(2) (1972) 146–160.

[42] R.E. Tarjan, Graph algorithms in chemical computation, in: R.E. Christoffersen, ed., Algorithms for Chemical Computation (American Chemical Society Symposium Series Vol. 46, Washington, DC, 1977) 1–20.

[43] A. Tucker, Applied Combinatorics (Wiley, New York, 1980).

[44] L.G. Valiant, The complexity of computing the permanent, Theoret. Comp. Sci. 8 (1979) 189–202.