Computational complexity of synthetic chemistry – Basic facts

Warren D. Smith*
wds@research.nj.nec.com

date = April 18, 1997

Abstract — **Three negative results:**

1. **Given a set of allowed starting materials and allowed chemical reactions, the problem of determining whether a chemical of specified structure is synthesizable is Turing undecidable.**
2. **The question of whether it is synthesizable in N steps is NP-complete. These 2 results apply in a general abstract setting and do not prove anything about how hard the problem is with any specific set of starting materials and allowed reactions. In particular we prove nothing about the case when the starting materials are all commercially available chemicals and the reactions are all known reactions!**
3. **The IUPAC naming scheme for assigning names to organic compounds will assign ambiguous names to most N-atom compounds and requires great chemical expertise. We show that both these problems are avoidable with alternative naming schemes. There are unquestionably *far* simpler and better methods to name chemical compounds than the IUPAC scheme.**

All 3 of these facts are almost trivial from the viewpoint of modern computer science, but I don't think they've been mentioned in the chemical literature.

Positive results: We discuss algorithms for finding out how to solve the synthesis problem (of determining whether something is synthesizable, and what the best synthetic process is). An interesting generalization of the famous "shortest path problem" in directed graphs arises in which the paths become synthetic trees, the graphs become hypergraphs, and the distances become synthetic costs. This problem is soluble either by generalizing Dijkstra's algorithm or the dynamic programming algorithm. Despite the fact that these algorithms are highly efficient as a function of the size of the hypergraph, in practice because the hypergraph arising from chemistry is enormous, that is inadequate. We discuss methods for ameliorating this.

Keywords — Hypergraph, graph isomorphism, chemical nomenclature, Turing undecidable, NP-complete, synthetic trees, generalized shortest paths.

*NECI, 4 Independence way, Princeton NJ 08544

1 Undecidability and NP-completeness of synthesizability problem

In this section we propose two mathematical models of synthetic chemistry. Inside these two models, it turns out that the questions of whether a given chemical is synthesizable from a given set of allowed starting materials with a given set of allowed reactions, are Turing undecidable and NP-complete respectively.

Note: neither of these two models addresses the *specific* problem of whether the chemicals currently commercially available, plus the reactions currently known to chemists (even assuming it were possible to say what these are!), lead to any undecidable problems. Instead we only prove that there exist undecidable (or NP-complete) instances of this *general, abstract* problem:

The problem poser gives us a set of starting materials, tells us a set of allowed "chemical" abstract transformations, and specifies a goal material. Is that goal reachable from the starting materials using the transformations?

Model #1 maps synthetic chemistry to the "word problem in semigroups." That is, suppose you have some set of "fundamental elements" a, b, c, ... of a semigroup, and some set of "relations" among those elements, for example $cabbbc = d$. Also, you have some set of "generators" each of which is a word made of fundamental elements, for example ab, bac. Then the "word problem" is: is some word (product of fundamental elements) an element of the semigroup generated by the generators? This is a well known undecidable problem (Apparently this was originally shown by Emil Post [see Davis 1965] and A.M.Turing 1950.)

The connection of this to chemistry is, our set of allowed chemical reactions happen to be multiplying a word by a generator on the left (modifying our current chemical) and then simplifying using the relations (rearrangements of the atoms in the chemical) then determining whether some word is in the semigroup (some chemical is synthesizable) is undecidable.

Model #2 maps synthetic chemistry to an NP-complete tiling problem. Given a set of T "tile types" (in fact, 1×1 squares with "matching conditions" on the edges will suffice) and an outer region (in fact, an

$N \times N$ square will suffice), the problem is to decide if the outer region may be tiled by the tiles. This is a well known NP-complete problem (Garey & Johnson 1979); L.Levin even has shown that a version of it is "average case NP-complete" (Levin 1986).

The connection of this to chemistry is, is our set of allowed chemical reactions happen to be: the tiles are chemicals (starting materials); the reactions are gluing two tiles at a compatible border (and that makes the border disappear - two sets of two tiles which each tile some region R create the same chemical, even though the starting materials were different). Finally, there are reactions for "deactivating" a border of any tile, which we use for the border of the outer region, assuming we ever successfully tile it.

Notice in this model, the only important allowed reactions are "addition" reactions which make a product which is a bigger molecule than the reactants. In other words, in this model, retro-reactions are always simplifying. This one-way quality forces the synthesizability problem to be decidable, in fact in NP, since any chemical with N tiles is going to be synthesizable (if it is) in $\leq N$ steps, and we could simply investigate every possible sequence of N steps to make the decision.

Incidentally, the question, given a finite set of tile types, of whether the entire plane (rather than an $N \times N$ square finite subset of it) may be tiles, is Turing undecidable. Translated to chemistry, that means that it is undecidable whether one can make an infinite "polymer sheet" (ala graphite) from a given set of starting materials and reaction types.

Thus,

Theorem: In model #1, the synthesizability problem is Turing undecidable. In model #2 (in which only syntheses in which the size of the chemical monotonically increases along every synthetic pathway are possible) the synthesizability problem is NP-complete.

2 Naming organic compounds, and a proposed replacement for the IUPAC scheme

Organic compounds are simply graphs of bounded valence with bounded length edge labels and vertex labels.

Although the graph isomorphism problem is suspected to be superpolynomially difficult in the worst case, this problem for V-vertex graphs of bounded valency is known to be polynomial (Luks 1982) in fact if all valencies are $\leq d$ the runtime is $V^{O(d/\log d)}$ (Babai, Kantor, Luks 1983) and with $d = 3$ the runtime is $O(V^3)$ (Galil et al 1987).

Unfortunately the algorithms that prove these *worst-case* bounds are complicated. But simple algorithms are available which can be proven to run quickly on *random* graphs. These will find a "canonical vertex ordering" of a graph (two graphs with canonically ordered vertices are isomorphic iff the isomorphism maps vertex i of one graph to vertex i of the other).

Given any connected graph with all valencies ≤ 4, one can create an organic molecule with the same graph in which the edges of the graph are replaced by ℓ-long alkane chains $(CH_2)_\ell$ (with a sufficiently large value of ℓ, it should be possible[1] to embed the molecule in 3-space without any atoms overlapping) and the k-valent vertices are replaced by carbon atoms with $4 - k$ hydrogens attached.

It is known that isomorphism of graphs with vertex and/or edge labels is polynomially reducible to the graph isomorphism problem for unlabeled graphs (replace the labels with little hanging-off subgraphs; edges may be dealt with by introducing artificial extra vertices in the middle of each edge, and labeling them appropriately).

One might think it possible to get use geometrical as well as graph information, for example taking advantage of the fact that two point clouds in 3-space may be judged to be the same under some isometry, or not, in only polynomial time. However, since c-long chains are highly flexible, not rigid (at least at non-cryogenic temperatures), and the stereochemistry of the "vertex" carbons seems not relevant to the graph isomorphism problem (since, due to this flexibility, the same graph could be made regardless). So, geometry will not help.

The preceding three paragraphs prove that the problem of determining whether two chemicals are the same, is computationally *polynomially equivalent* to the graph isomorphism question[2], and the problem of finding unambiguous names for chemicals is the same problem as the canonization problem – both for labeled graphs of bounded valency – and hence is soluble in polynomial time.

In fact the IUPAC naming system apparently can assign a superpolynomially large number of *different* names to the same molecule, although fortunately all names produced will be only polynomially long. The IUPAC naming scheme will produce one such name, more or less randomly sampled from the possibilities, in polynomial time. Hence to search the chemical literature might require you to perform a superpolynomially large number of different searches one under each IUPAC name.

Speaking as a non-chemist... if I may be allowed to make a value judgement at this point... I consider the IUPAC naming scheme to be an ill-conceived and utter disaster which fossilizes hundreds of historical mistakes. In chemistry nomenclature is of central importance (unlike most other scientific fields, where it is a trivial issue) because it is the essential point of access to the chemical literature. But, the IUPAC scheme is so complicated it requires months of training to understand, the rules are hundreds of pages, and is extremely hard for anybody who is not a chemist, to learn. Visiting the

[1] How large a value of ℓ is needed? Well, it is known (Rosenberg 1983) that any N-vertex graph of bounded valence may be embedded in a $O(N^{1/2}) \times O(N^{1/2}) \times O(N^{1/2})$ 3D grid (edges map to disjoint paths) with with maximum edge length (as a path) $O(N^{1/2})$. Hence $\ell = O(N^{1/2})$ suffices to make an organic chemical out of any N-vertex graph.

[2] Aside from some non-combinatorial chemical notions, such as knottedness of polymer chains. We shall disregard such things for the moment.

Princeton university chemistry library, I was told that without the aid of a computer search, it was virtually impossible for me to look for articles about a particular chemical structure – but such a computer search was not available to patrons of the library unless they had special financial donations from a Princeton chemistry professor. In short, most of the chemical literature is inaccessible to the average patron of the Princeton chemistry library. Another time at a SUNY chemistry library, I was unable to locate a fairly well known compound in *Beilstein* even after about 2 hours of consulting various "guide to Beilstein" books and with the aid of the librarian.

However, we now see that it *is* possible to devise an unambiguous name for any graph structure (including all the graph structures arising in chemistry) with V vertices and E edges, which has length only $O(V+E)$. (More precisely, $O((V+E)\log(V+E))$ bits, but only $O(V+E)$ numbers long if the name is a sequence of nonnegative integers $O(V+E)$ in size.)

Furthermore, it is possible to do so with an algorithm that requires no chemical knowledge whatever.

The only problem is that the known algorithms which are provably efficient in the worst case, are not very simple. However, most molecules arising in practice do not represent the worst case graphs. Therefore it is acceptable to use a much simpler algorithm which is fast on average, although it may be slow in the rare worst case.

The simplest "canonical vertex ordering" method is to permute the rows (and simultaneously columns correspondingly) of the adjacency matrix A (A_{ij} is the number of bonds between atoms number i and j) to produce the lexicographically minimal equivalent adjacency matrix (Proskurowski 1974; or equivalently a "canonical labeling," Babai & Kucera 1979, may be found). There are other possible canonical vertex orderings; this is merely the simplest one to describe. The name is then: any space-efficient description of the (sparse) adjacency matrix. So far we have only described how to name graph structures, but chemicals also incorporate geometrical information (such as the chirality at each tetrahedral carbon and trans and cis double bonds). This information is appendable by adding $O(V)$ extra bits to describe the relative orientation of all the bonds sticking out of each atom, in the natural ordering of the atoms that was decided upon, and $O(E)$ extra information to similarly describe the bonds. Also, we can describe what element lies at each vertex of the graph with $O(V)$ additional bits (and/or only perform permutations which preserve the fact that the vertices are ordered lexically by atomic composition).

Unfortunately, determining the lexicographically minimal adjacency matrix under all possible atom orderings, is thought to be computationally difficult in the worst case using simple algorithms which work by backtracking. But in the average case, quite naive canonization algorithms seem to be very satisfactory. Also, canonical orderings are easily found for graphs which are "trees." (For a rooted tree: canonically order all the subtrees of the root recursively, then sort the subtrees lexically by their canonizations. For unrooted trees, which vertex to call the "root" may be decided by brute force lexical minimization among the canonizations for all possible rooted forms.) So here is a specific suggestion, designed to be suitable for chemistry:

Canonical ordering algorithm suitable for chemistry.

1. Label every vertex of the graph with its atomic type (H, He, Li, etc.), ionization type (e.g. +1), and its valence. Label each bond with information such as single/double/triple.
2. If the graph is a tree, canonize it and we are done. Otherwise, prune off any trees which stick off the main 2-connected body of the graph, and replace them with additional vertex labeling information at their tree root, describing the canonization of that (rooted) tree.
3. Replace the label of each vertex by: its label, plus the lexically sorted set of labels of its neighbors.
4. Replace the label of each vertex by: its rank among all vertex labels, in lexical order. (For example, if among the V vertices, some vertex has the 7th smallest label in lexical order, then replace its label with "7.")
5. Back to step 3 until labelings cease to change.
6. Now append to the vertex labels any chirality information; also label each bond with any chirality information relevent to it (cis/trans), and go back to step 3 until labelings cease to change.
7. For each subset of vertices whose elements have not yet been ordered by the above procedure (i.e. tied subsets): break the ties by permuting their elements to minimize the adjacency matrix lexicographically.

The runtime of this algorithm up to and including step 6 is clearly bounded by polynomial(V). Note, at most V backward branches back to step 3 will ever be required, because each transit through the main loop (except for the last) breaks at least one tie among at least two vertices, so that after V transits the vertices are totally ordered (or we have stalled and all ties are unbreakable).

The final step 7 may be expensive in the worst case, but it will rarely be in practice.

This will now make it possible for somebody to search the chemical literature while only performing *one* search, with a sufficient precomputation (name computation by canonical vertex ordering), and this precomputation could be done by a standalone machine which itself had no chemical knowledge and was not part of some vast chemical search system brought to us by a great, overbearing, chemical information monopoly. Also, the naming of most small compounds should be within the reach of an unaided human, with all the rules expressed on 1 page, not 100s of pages.

So, I would recommend introducing a new standard chemical naming system based on these ideas to replace or augment the IUPAC system, at least for the purpose of indexing the literature.

Even the scheme we have described still will *not* suffice to describe topological information, such as is needed to describe catenanes (linked chains) and rotoxanes (knots). Even this can be accomplished, although at very heavy cost, because knot equivalence is decidable (Hemion 1992, Waldhausen 1978; W.P. Thurston has the best known method at present, based upon metrical equivalence of hyperbolic 3-manifolds, see Hoste et al. 1998), hence the lexically minimal shortest-named equivalent knot to any given knot is computable. However, in my opinion it simply is not worth it.

3 Efficient algorithms for finding the min-cost synthesis of a goal compound

The space of chemical compounds and the reactions among them are well described by a combinatorial entity I call a "directed, labeled hypergraph" (Berge 1989). Specifically, the vertices of the hypergraph correspond to chemicals. The hyperedges correspond to reactions.

If a reaction takes k reactants and generates a product, then there is a hyperedge containing those k reactants and the product vertices, and "directed" from the reactants toward the product. (If there are several products, then make several hyperedges.) Furthermore, each directed hyperedge with k reactants comes with k "retro yields," specifying how many grams of each reactant are needed to make 1 gram of product (these numbers incorporate both stoichiometry and yield). These numbers are always positive reals, and the sum of the retro yields on any one hyperedge is always ≥ 1.

Knowledge of the dollar "cost" of any starting material may be incorporated into this model by adding the additional chemical substance "money" and adding additional "reactions" which transform r grams of money into 1 gram of starting materials. (We assume the unit of money is scaled so that the retro yields in these reactions, too, are always ≥ 1. That is, the cheapest chemical per gram, costs exactly 1 dollar per gram.)

Knowledge of the dollar cost of disposing of waste products of each reaction unfortunately is not easy to incorporate into this view.

Now, within this weighted hypergraph model, there are "efficient" algorithms for finding the least cost synthesis of a given molecule, as I will soon show. Here by efficient, I mean when the running time is expressed in terms of the size of the hypergraph. Unfortunately, although this notion of efficiency is the one commonly used in computer science, it is not of great use to us because the hypergraph of all chemicals is enormous. (There are at least 10^{1000} chemicals.)

So what we really need is an algorithm that does *not* input this graph, but instead is provided this input in the form of a "reactions oracle" which can generate the hyperedges incident on a vertex (and vice versa) on request. This allows algorithms that run in finite time even on an infinite hypergraph.

3.1 *Dynamic programming algorithm – forward direction*

The first algorithm we discuss is based on "dynamic programming." It will find the least cost synthesis among all syntheses which require $\leq L$ "reaction-levels," where the 0th level is the starting materials. Indeed, it will find all chemicals synthesizable by ($\leq L$)-level syntheses, and the best (least cost) synthesis for each. See figure 1.

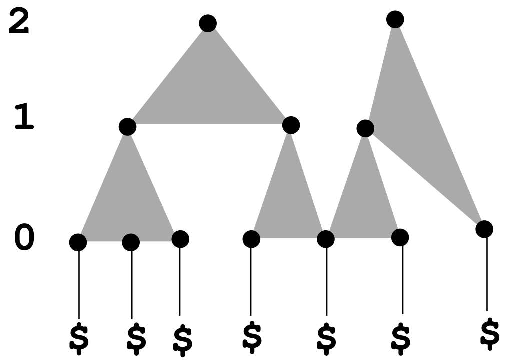

Figure 1: Hypergraph of chemicals and reactions. Hyperedges are denoted by grey polygons and by convention are directed upward, i.e. the product is the top vertex of each hyperedge and the reactants are the other vertices. The "starting materials" are the ones at "level 0." The hypergraph shown has 3 "level 1" chemicals and 2 "level 2" chemicals.

The data structure representing the hypergraph of chemicals and reactions is simple. Each *vertex* (representing a chemical) knows its synthetic cost per gram (initially all costs are ∞) and has pointers to its outgoing hyperedges. (It can also be convenient, but it is not necessary, for there to be pointers to its incoming hyperedges. Also, the "pointers" might not be explicitly stored but instead might only be accessible through the reactions oracle.) Each *hyperedge* (representing a reaction) knows its destination vertex (product of the reaction) and its other vertices (reactants). It has an integer "label" telling us the "number of unavailable reactants." For a k-reactant hyperedge, this integer is initially k, except for the special hyperedges with only one reactant "money" and only one product (a starting material), for which this integer is initially 0. (These special edges are shown at the bottom of figure 1.) Finally, each hyperedge knows a boolean "status" value saying whether this hyperedge "needs investigation;" all these boolean values are initially FALSE except for the special hyperedges (TRUE).

Hyperedges conceptually fall into "buckets" depending upon their label and their boolean status ("needs to be investigated" or not); we assume the maximum number of reactants in any one reaction is bounded by some constant, so there are a constant number of buckets.

The algorithm is a sequence of "stages." In each stage we examine every vertex and every incoming hyperedge to that vertex. Actually, though, we can get the same effect with less computing by only examining hyperedges in the (0, needs investigation) bucket, and only investigating the vertices which are their destinations.

At stage n, $n = 0, 1, 2, \ldots$, we find the least cost ($\leq n$)-level syntheses of all chemicals which may be synthesized in $\leq n$ reaction levels, and label the appropriate vertices of the hypergraph with their synthetic costs per gram. This subalgorithm may be implemented efficiently as follows. Every time we decrease the cost of a vertex (i.e., find a new and better synthesis) versus its value from the previous stage, we mark all outgoing hyperedges from that vertex as "needs investigation" and if the previous cost of that vertex was ∞ (i.e. we've just found the first synthesis) we decrement the count of "unavailable reactants" on every hyperedge leaving that node.

A new stage may be accomplished by scanning through all hyperedges in bucket (0, needs to be re-investigated) and computing the new costs of their destination vertices as the min of their present cost values and their new cost values. Once a hyperedge is investigated in this way, its status flag is reset.

Of course, every time the status flag or unavailable-reactant count (label) of a hyperedge changes, we move it from one bucket to another.

Assuming each vertex can be involved in at most a constant number of reactions (since there are only a constant number of reaction types) and each reaction involves at most a constant number of reactants, then the total number of steps, and of memory locations consumed by, this algorithm is $O((V + E)L)$ where V is the number of vertices and E the number of hyperedges in the sub-hypergraph of chemicals which our algorithm actually explores before quitting, and L is the maximum permitted number of levels in the synthesis.

3.2 Dijkstra-like algorithm – reverse direction

In this algorithm, we start with the goal vertex, and work "backward" to reach the starting materials.

A complete synthetic process for some goal molecule is a "hypertree" formed by all the reactants and reactions leading from the starting materials toward that product (Smith & Warme 1999). Figure 1 is the union of two such hypertrees, one for each of the two level-2 chemicals. The leaves of this tree are the starting materials, the root is the goal molecule, and the (hyper)edges are the reactions. (Clearly, there will be no cycles, hence it is WLOG[3] a tree, because every directed cycle would have retro yield ≥ 1 hence WLOG could be discarded, and every time a chemical is synthesized in more than one way, WLOG we can ignore one of the ways.) The "synthetic cost" of such a tree T is given by

$$\text{cost} = \sum_{P \in T} \prod_{e \in P} r_e \tag{1}$$

where P are all the root-leaf directed paths in the tree T, e are all the directed edges in the path P, and r_e is the retro yield associated with edge (a 1-product, 1-reactant subset of a hyperedge) e. This assumes all our starting materials are of unit cost, or that we've used the "money" trick mentioned before to get that effect.

This cost is exactly the cost of all the starting materials needed to make 1 gram of product.

Our algorithm will proceed as follows. At any moment we have some already-explored portion E of the hypergraph. All vertices in the explored portion could have synthetic cost underestimates assigned to them, by making the "leaves" of the explored portion have their true costs (if known) or assigned an artificial cost of 1 dollar per gram (if unknown), and then propagating costs up the tree. Because all retro yields are ≥ 1 (and due to our previous scaling of "dollars") these cost underestimates are valid lower bounds.

We then add the "greedy vertex" on to E – the vertex η, adjacent to and a predecessor of some substance in E, whose addition increases the cost of any synthetic tree to make the goal molecule that includes η, by the least. We continue on until a complete synthetic tree has been found. (All vertices may be labeled as "synthesis known" or "synthesis unknown" and these labels may be propagated too; once the goal molecule is known, we have a complete synthetic tree.)

Theorem: This first tree found, will in fact be the least cost synthetic process.

Proof. During the algorithm we always have a valid lower bound on the synthetic cost of the goal molecule. The lower bound always increases. Once a synthesis is found, this lower bound is exact. Hence, we clearly find the least cost synthesis. □

Now we describe how this algorithm may be implemented efficiently.

Every time a vertex is explored or its cost updated, we update its predecessor hyperedges; they will be either added to a "heap[4]," or (if they are already there) they will have their incremental costs increased. The "incremental cost" of a hyperedge is the additive adjustment in the cost of the goal molecule that will be entailed by adding that hyperedge to the current best synthetic tree. It is the product of the retro yields on the root-to-its-product path (these products are assumed to be stored in each explored vertex), times a term depending only on the hyperedge itself.

Now to find the greedy hyperedge we simply find the minimal element in the heap.

We do not need to propagate costs until the end (once the best synthesis is found). To propagate costs: the unit cost of a node η is the min (over all hyperedges H containing η as their "product") of the sum (over all

[3]Without loss of generality.

[4]A "heap" is a data structure (see Tarjan 1983 or numerous other data structures/algorithms books) for organizing N items. Each item has a real "value." The virtue of the heap (as opposed to other data structures such as arrays), is that it permits the item with the minimum value cost to be retrieved in $O(\log N)$ steps and permits an item insertion or item value alteration in $O(\log N)$ steps. In our particular application here (to chemistry) we use the incremental costs as the "values."

reactants s in H) of $r_s c_s$ where c_s is the unit cost of the substance s.

The total time for running this algorithm will be $O(V + E \log V)$ where V and E are respectively the total number of vertices and edges explored by the algorithm before it succeeds (or fails – if no predecessors can be found). This is because each addition to E requires $O(1 + \log V)$ time, and the final cost propagation takes $O(E + V)$ time.

Note: in both of these algorithms, V and E are much less than the total number of vertices and edges in the whole chemistry graph (which in fact could be infinite for all we care) hence our running times are in some sense "sublinear" in the best case. Also, actually $E = O(V)$ because we are assuming at most some constant number of possible reaction and retroreaction kinds.

It is possible to reduce the time for our "Dijkstra" algorithm further if a priori lower bounds are available (from some oracle) on the synthetic cost of any chemical. (Of course, we were already using the bound "cost ≥ 1", but I am talking about better bounds than that.) These bounds would be incorporated into the "incremental costs" used by the Dijkstra algorithm to find the "greedy edge." In the ultimate limit in which these bounds were exact, this would in fact result in finding the best synthetic tree immediately without exploring in any wrong directions.

3.3 Connection to the classical "shortest path" problem in graphs

In the case when the *hypergraph* reduces to a *graph* (reactions with only 1 reactant) then the reader may readily verify that the problem of finding the least-cost synthetic *tree* reduces to the problem of finding a shortest *path* in a directed graph. In fact this is a single-source (money) single sink (target molecule) shortest path problem in whcih all edge lengths are positive (edge lengths correspond to log retro yields, and are positive since yields are < 1). Our algorithms reduce respectively to the dynamic programming (Bellman-Ford) and Dijkstra algorithms for the shortest path problem. These algorithms are described in most undergraduate algorithms textbooks, e.g. Cormen et al. 1990.

4 Available Computer programs for synthetic chemistry

Three of the most popular available programs are LHASA ("Logic and Heuristics Applied to Synthetic Analysis"), Syngen, and CAMEO ("Computer Assisted Mechanistic Evaluation of Organic reactions")

LHASA was initially written by E.J.Corey and W.T.Wipke at Harvard in 1967 and has now undergone 30 years of continuous development by a large number of contributors. One of LHASA's most important features is its chemist-friendly graphical user interface. By 1977, LHASA already consisted of 30K lines of FORTRAN plus a database of 600 reaction families described in their own language "CHMTRN," and presumably it is a lot larger now. LHASA also will supply literature citations to substantiate each synthetic step it proposes. LHASA usually must be employed in an interactive mode in which the human operator guides the search down pathways he considers promising. This can be very labor intensive on large searches.

LHASA is based on the ideas of E.J.Corey 1967 about how to find good syntheses. These ideas depend upon recognizing "retrons," which are patterns in the target molecule which may be transformed by an inverse chemical reaction (Corey calls this a "retroreaction," and his way of thinking "retrosynthetic") to get a simpler molecule or molecules. LHASA contains a large database of retrons and retroreactions.

There are also a large number of heuristics about which retroreactions look the most promising to try first (generally the ones which result in the "greatest simplification" because they cause a lot of "ring disconnection," are likely to allow a currently-blocked retro-transform, are "strategic bonds," remove stereocenters, etc.) and which "tactical combinations" of retroreactions are likely to result in dramatic simplifications. Recently several user-selectable "strategies" (long range, short range, topological, stereochemical, and starting material based) have been added to LHASA's heuristics.

Syngen appears to be based on "synchem," a program originally written by H.Gelernter at SUNY Stony Brook starting in the late 1960s. Given a goal molecule, it attempts to find the "best" synthesis of it from a large built-in set of $\approx$ 6000 commercially available starting materials (essentially, the Aldrich catalog) using a large built in database of reaction types. Only syntheses which involve at most 4 starting materials are allowed, and the goal molecules may not have more than 32 atoms or bonds. Within these constraints, Syngen tries to make an exhaustive search of all possible synthetic routes.

Presumably the algorithm used by Syngen is similar to the ones we've described, although it has not been formally described.

CAMEO (by W.L.Jorgensen et al.), given a description of starting materials and reaction conditions, attempts to predict the product(s) of that reaction.

How good are these programs compared to human chemists? How valuable do human chemists find them?

Apparently, very few chemists employ these programs at the present time, at least judging from the number of papers in chemistry journals which acknowledge their use (i.e. only a few per year; there seem to be substantially more papers about the creation of the programs than papers produced with their aid!).

As for how good they are, a small quiz was made by U.S. Environmental Protection Agency chemists (Anastas et al. 1994) in which Syngen, LHASA and CAMEO were asked to rediscover some well known syntheses of moderately small organic molecules.

On 5 quiz questions, CAMEO (with the optimal subset of modules activated – a different optimal subset for each quiz question – and being told the reactants, solvent, and conditions) scored 4 partially correct answers and 1 failure.

LHASA and Syngen both succeeded in producing plausible synthetic routes (sometimes more than one) to all compounds in the EPA test, although in some cases they did not find the standard or most cost effective routes.

The authors of LHASA have opined that it is currently not good enough to be an essential tool for every chemist, but claim that it is useful for suggesting alternative routes and generating ideas.

5 Appendix on computer science terminology

For the benefit of chemists unfamiliar with computer science, I define some common computer science and mathematics words and concepts, and refer you to some elementary textbooks.

An *algorithm* is a computer program guaranteed to terminate, no matter what input it is supplied with. If the maximum number of steps the algorithm will run before terminating, on an N-character long input, is $F(N)$, then the function $F(N)$ is called the "time complexity" of that algorithm. If all the algorithm's memory accesses lie entirely inside a stretch of memory $G(N)$ words long (maximized over all N-bit inputs), then $G(N)$ similarly is the "space complexity" of that algorithm.

A class of computational problems (often, for short, just called a "problem")[5] is "solved" by an algorithm if the algorithm, when supplied with input describing the specific problem instance from that class, will always output the solution and then stop.

The set of problem-classes soluble by algorithms are the *decidable* ones. The set of problem-classes soluble by algorithms with time complexity bounded by polynomial functions of N, is called "P." For example, multiplication of two integers is obviously in P, since multiplication algorithms exist with runtime bounded by a polynomial function of the number of digits in the integers. P is often thought of informally as the "feasible" or "easy to solve" problems. The set of yes-no problems whose "yes" solutions, once known, have proofs which may be verified by an algorithm in P, is called "NP." For example, the question of whether an integer is composite (has a factor besides itself and 1) is in NP because you can verify a factorization by multiplication. P$\subseteq$NP$\subset${all decidable problems}. A problem class X is *NP-complete* if any instance Y of any problem in NP may be transformed into a problem in X, such that the solution of X may be transformed to the solution of Y, and both these transformation tasks are in P.

Thus, NP-complete problems are at least as hard (up to polynomials) as any problem in NP, and hence are thought to be very hard. The standard text on NP-completeness is (Garey & Johnson 1979).

[5] Meaning, a map from "problem instances" to "solutions," both being strings of characters.

A *semigroup* is a set S together with a binary "product" operation mapping $S \times S$ into S, such that this product is "associative," i.e. $a(bc) = (ab)c$. The "word problem in semigroups" (described in the main paper) is an example of an *undecidable* class of problems. An introductory book on decidability is (Minsky 1967).

A *graph* (Harary 1972) is a set of "vertices" and "edges," where an edge is just a pair of vertices. A "directed graph" (sometimes called a "digraph" for short) is the same except that the edges are now *ordered* pairs of vertices, one being a source and the other a destination. (Directed graphs are typically drawn with arrows on the edges.) For example, the triangle-cycle directed graph has vertices {1,2,3} and directed edges {(1,2), (2,3), (3,1)}.

A *hypergraph* (Berge 1989) is just like a graph, except that it has *hyperedges* instead of edges, where a hyperedge is a set of vertices whose cardinality is now allowed to be any number ≥ 2. The present paper also admits the notion of a "directed hypergraph" in which each hyperedge has one distinguished "destination" vertex.

An introductory textbook on algorithms is (Cormen et al 1990).

One example of a graphical computational problem is the "shortest path problem." In this problem, the input is a description of a graph where each edge comes with an integer "length." The task is to find the shortest path from one specified vertex to another. One algorithm with polynomial time complexity for solving this problem (which works if all the lengths are non-negative) is "Dijkstra's algorithm." Many graph algorithms, including this one, are discussed in (Tarjan 1983).

Another example is the "graph isomorphism problem" (GI) in which the input is the description of two graphs, and the output is a permutation of the first graph's vertices, which causes its edges to become the same as the second graph's edges. Such a permutation exists if and only if the two graphs are "isomorphic." (If they aren't, the algorithm instead should say so.) Clearly the graph isomorphism existence problem is in NP. Although it is not known whether GI is in P, it is nevertheless normally thought of as an easy problem because superpolynomially hard worst case instances seem to be very rare (if they exist).

6 References

P.T.Anastas, J.D.Nies, S.C.DeVito: Computer assisted alternative synthetic design for pollution prevention at the U.S. Environmental protection agency.

L.Babai, P.Erdös, S.M.Selkow: Random graph isomorphism, SIAM J. Comput. 9 (1980) 628-635.

L.Babai, W.M.Kantor, E.M.Luks: Computational complexity and the classification of finite simple groups, SFOCS 24 (1983) 162-171.

L.Babai & L.Kucera: Canonical labeling of graphs in linear average time, Proc. 20th IEEE Symp. Foundations of Computer Sci. (1979) 39-46.

Claude Berge: Hypergraphs: combinatorics of finite sets, North-Holland 1989.

E.J. Corey: General Methods for the Construction of Complex Molecules, Pure & Applied Chemistry 14 (1967) 19-37.

E.J.Corey & X-M.Cheng: The Logic of chemical synthesis, Wiley 1995.

E.J.Corey & T.W.Wipke: Science 166 (1969) 178-192.

E.J.Corey, A.K.Long, S.D.Rubenstein: Computer assisted organic synthesis, Science 228 (1985) 408-418.

E.J. Corey: Computer Assisted Analysis of Complex Synthetic Problems, Quart. Rev. Chem. Soc. 25 (1971) 455-482.

E.J. Corey: Pure & Applied Chemistry 2 (1971) 45-68.

T.H.Cormen, C.E.Leiserson, R.L.Rivest: Introduction to algorithms, MIT press 1990.

M. Davis (ed.): The undecidable; basic papers on undecidable propositions, unsolvable problems and computable functions. Raven Press, Hewlett NY 1965

N.Deo, J.M.Davis, R.E.Lord: A new algorithm for graph isomorphism, BIT 17 (1977) 16-30.

Z.Galil, C.M.Hoffmann, E.M.Luks, C.P.Schnorr, A.Weber: An $O(n^3 \log n)$ deterministic and $O(n^3)$ probabilistic isomorphism test for trivalent graphs, J.ACM 34 (1987) 513-531.

M.Garey & D.S.Johnson: Computers and intractibility, a guide to the theory of NP-completeness, Freeman 1979.

H.Gelernter, A.F.Sanders, D.L.LArsen, K.K.Agarwal, R.H.Boivie, G.A.Spritzer, J.E.Searleman: Empirical explorations of SYNCHEM, Science 197 (1977) 1041-1049.

International union of pure and applied chemistry: Nomenclature of organic chemistry A,B,C,D,E,F,G,H, Pergamon 1979.

William L. Jorgensen, Ellen R. Laird, Alan J. Gushurst, Jan M. Fleischer, Scott A. Gothe, Harold E. Helson, Genevieve D. Paderes, and Shenna Sinclair: CAMEO: A Program For The Logical Prediction Of The Products Of Organic Reactions, Pure & Appl. Chem. 62 (1990) 1921-1932.

W.L.Jorgensen: Free Energy Calculations: A Breakthrough for Modeling Organic Chemistry in Solution, Acc. Chem. Res. 22 (l989) 184-

L.Kucera: Canonical labeling of regular graphs in linear average time, Proc. 28th IEEE Symp. Found. Of Comput. Sci. (1987) 271-279.

H. E. Helson and W. L. Jorgensen: Computer Assisted Mechanistic Evaluation of Organic Reactions. 24. Carbene Chemistry. J. Org. Chem. 59(1994) 3841-3856.

G.Hemion: The classification of knots and 3-dimensional spaces Oxford University Press, 1992.

J.Hoste, M.Thistlewaite, J.Weeks: The first 1701936 knots, Mathematical Intelligencer 20 (Fall 1998) 33-??. `http://www.math.utk.edu/ morwen/`

F.T. Leighton: Complexity issues in VLSI, MIT Press 1983.

F.T.Leighton & A.L.Rosenberg: Three-dimensional circuit layouts, SIAM.J.Comput. 15,3 (1986) 793-813.

L.Levin: Average case complete problems, SIAM J. Comput 15,1 (1986) 285-286.

E.M.Luks: Isomorphism of graphs of bounded valence can be tested in polynomial time, J.Comput.Syst.Sci. 25 (1982) 42-65.

M.L.Minsky: Computation: finite and infinite machines, Prentice-Hall 1967.

R.Panico, W.H.Powell, J.C.Richer: A guide to IUPAC nomenclature of organic compounds, Blackwell Scientific Publishing, Oxford 1993.

A.Proskurowski: Search for a unique incidence matrix of a graph, BIT 14 (1974) 209-226.

R.C.Read & D.G.Corneil: The graph isomorphism disease, J.Graph Theory 1 (1977) 239-263.

A.L.Rosenberg: Three-dimensional VLSI: a case study, J.ACM 30,3 (1983) 397-416.

G.Rozenberg & A.Salomaa: Cornerstones of undecidability, Prentice Hall, 1994.

W.D.Smith & D.Warme: Hypertrees, manuscript in progress, 1999.

R.E.Tarjan: Data structures and Graph algorithms, SIAM 1983.

A.M.Turing: The word problem in semigroups with cancellation, Ann.Math. (Princeton) 52 (1950) 491-505.

J.D.Ullman: Computational aspects of VLSI, Computer Science Press 1984.

F.Waldhausen: Recent results on sufficiently large 3-manifolds, Proc.Symp.Pure Math. 32 (1978) 21-37.

W.T.Wipke & W.J.House (eds.): Computer assisted organic synthesis, ACS 1977

R.B. Woodward: Synthesis, in A.R. Todd (ed) "Perspectives in Organic Chemistry" (1956)

V.M.Zemalychenko, N.M.Kornienko, R.I.Tyshkevich: Graph isomorphism problem, J. Soviet Math. 29 (1985) 1426-1481.