ELSEVIER Discrete Applied Mathematics 67 (1996) 189–207

DISCRETE APPLIED MATHEMATICS

Pruning the search tree in the constructive enumeration of molecular graphs

Jiří Pospíchal, Vladimir Kvasnička*

Department of Mathematics, Faculty of Chemical Technology, Slovak Technical University, 812 37 Bratislava, Slovakia

Received 9 September 1993

Abstract

Exhaustive and nonredundant constructive enumeration of molecular graphs (vertex colored multigraphs) with possible further constraints such as prescribed valence states of atoms (degrees of each vertex for each edge multiplicity) is presented. The employed canonical labeling of vertices is based on maximum code formed from the rows of the upper-triangle part of the adjacency matrix concatenated in a sequence from the top to the bottom. The recurrent constructive scheme uses several techniques for rejection of subtrees of a search tree, such as checking conditions for sequences of valence states to be graphical and employing of the automorphism group. The description is accompanied by illustrative examples.

1. Introduction

Generally, molecular graph in the present paper means a colored multigraph, where colors of vertices substitute for kinds of atoms and multiplicities of edges correspond to multiplicities of chemical bonds. In this paper we shall deal only with integer bond orders, so the multiplicity of an edge is a positive integer. A basic approach to construction of graphs was described in [9,30]. An efficient generation of trees has been already successfully solved [18,29] ; the present paper deals with graphs containing cycles.

The need for a molecular graph generating algorithm in chemistry appears mostly in cases when we are given insufficient information about the molecule (chemical graph) to determine the molecular structure precisely, so we need to see the possible structures to decide which type of further check should be done to determine the structure properly. The available insufficient information comes mostly from chemical

*Corresponding author.

spectroscopic data, but it could also come from a range of artificial measures of structure similarity like topological indices. As the requirements differ from task to task, there is no point in designing an algorithm generating special types of graphs, and as the number of graphs with the required number of vertices could be astronomical, there is no point in generating full list of graphs once and forever. Therefore, an algorithm generating all possible graphs, which could then be adapted to take into account the prescribed restrictions, should be available.

There exist many algorithms, more or less efficient, solving the present task [16]. However, in most cases they can be divided into two groups. One group is made up of algorithms produced by chemists, adjusted to chemical needs, but sometimes lacking efficiency due to the method used for the construction of graphs [3, 23]. The other group is made by mathematicians or computer scientists, who are interested either in generating special types of graphs, or in a general problem of conditions for existence of generating algorithms for configurations. The latter algorithms are very sophisticated, but often poorly match specific chemical demands (with the exception of [12]) and their description is presented in high mathematical level, hardly readable by chemists. This statement can be illustrated by the fact that probably the most efficient algorithm for generating graphs, proposed by Faradzhev nearly 20 years ago [1, 10], yet has not been generally recognized by chemists. Though many additional sophisticated improvements can be added to this algorithm, their contribution to efficiency may be many times disputed. The present work presents some basic principles of the generation of molecular graphs in a more intuitive and easily readable way, combining Faradzhev's approach [10] with other methods [12, 25–27]. Most of these methods have already been exactly mathematically described, though sometimes in a different context, therefore, no proofs or exact definitions will be given here.

Generally, we are dealing with constructive enumeration of all graphs with given parameters. This should not present any basic problem. However, if the state space and search strategy are not chosen very carefully, exorbitant amounts of time may be required. The search strategy described further will be based on automorphism groups acting on graph and on backtracking. As we are trying to generate graphs efficiently, it can be done most advantageously in steps. Backtracking is a variant of the depth-first search in which a solution is sought by continually extending partial solutions. At each stage of the search, if an extension of the current solution is not possible, the algorithm backtracks to a smaller partial solution and tries again [5, 11]. In the first step, we would have only a set of nonadjacent vertices and then connect the first vertex with some other vertices. In the second step, we would connect the second vertex with some higher-labeled vertices, and continue in that way. At the end, we would exhaust the whole given number of edges and the graph would be finished. When we set out to generate the second graph, we need not begin from scratch, but it is sufficient to use as a basis one of the unfinished stages of construction from the previous graph. This approach can be described as a search tree, when the empty starting graph would be the root node and a finished graph would be a node down at the end of the branch. An edge in the search tree would amount to adding edges to an unfinished graph (node in search tree) thus creating another graph (node) down on a branch. We are generating each graph only once, so it is not necessary to check, whether the new graph has already been generated to weed out isomorphic copy of a previously determined graph. Instead, we would produce only labeled graphs satisfying some condition of such labeling, which would ensure that from a family of isomorphic graphs, only one has the right labeling. The rejection process employs the fact that graphs are generated by a backtrack algorithm in the decreasing lexicographic order. Unfortunately, checking the canonicity of the labeling corresponds in its complexity to the isomorphism checking. However, it is sometimes possible to recognize already in an early stage of construction, when only few edges were placed, that the graphs that would be constructed from this unfinished graph have been already constructed before. Therefore we would not continue in adding edges to this unfinished graph and go to another unfinished graph. In the search tree this corresponds to a cutting off of a branch before it reaches its end node (finished graph). A similar approach is used in checking of the canonicity of the generated graph, where a node of the search tree on kth level must be presented by a graph with k labeled vertices obtained by relabeling from checked generated graph containing n vertices, excluding edges between the rest $n-k$ unlabeled vertices.

We shall deal mainly with generation of simple graphs with given numbers of vertices and edges; its adaptation to some of the most common restrictions will be described afterwards.

To handle graphs with a computer, we need their proper description. The most popular one is the adjacency matrix, and probably the best known canonical labeling for construction of chemical graphs is based on the code produced by the maximum upper-triangle part of the adjacency matrix, where we concatenate the rows and handle the whole sequence as one number [1, 15]. Where not defined in another way, the term canonical will mean the above-mentioned type of canonicity.

Let us have a graph $G(V, E)$ with canonical labeling α: $V \to N$, where N represents natural numbers and vertices are labeled by the first $|V|$ natural numbers. This mapping also determines the mapping of unordered pairs of vertices α': $(v_i, v_j) \to N$ by the first $|V|*(|V|-1)/2$ natural numbers, $v_i, v_j \in V$. The mapping is done on the basis of lexicographic ordering of pairs according to the labeling α, i.e. for $|V| = n$

$$\begin{aligned} &\{1,2\} < \{1,3\} < \dots < \{1,n\} < \\ &\qquad\quad \{2,3\} < \dots < \{2,n\} < \\ &\qquad\qquad\quad \cdot \qquad\quad \cdot \\ &\qquad\qquad\quad \cdot \qquad\quad \cdot \\ &\qquad\qquad\qquad < \{n-1,n\} \end{aligned}$$

Labeling α' means practically labeling the entries of the upper-triangle part of adjacency matrix in the row-by-row fashion by natural numbers. Comparing the two labelings α and β up to kth level means comparing the entries of their upper-triangle parts of adjacency matrices A and B, starting from first entries $\{1,2\}$ and finishing either at $\{k, n\}$ entries or at the first different entries.

Let us have another labeling $\beta : V \to N$, which leads to mapping of unordered pairs of vertices $\beta' : \{v_i, v_j\} \to N$ and corresponding to the upper-triangle part of the adjacency matrix B. If labeling α is canonical, any labeling β must either be an automorphism and the upper-triangle parts of adjacency matrices are equal, or entries of upper-triangle parts are equal in lexicographic order up to m, and for $m + 1$ entry the $a_{ij} > b_{ij}$.

When for $m + 1$ entry $a_{ij} < b_{ij}$, the labeling α would not be canonical. Then, since our generating procedure should produce graphs in lexicographic order according to their codes, all solutions containing subgraphs described by the first $m + 1$ entries and isomorphic to subgraph described by the first $m + 1$ entries of the present code have been already generated. It is not necessary to expand this solution by filling some of the remaining $m + p$ entries ($p > 1$) of upper-triangle part of adjacency matrix and we would continue with the next matrix, containing some change in first $m + 1$ entries. There are several methods for pruning the search tree both during the generating process as well as in the canonicity checking process.

Efficient generation of colored multigraphs for chemistry should not be treated separately as a construction over algorithms generating graphs, though for simplicity we shall describe the principles of generation in this way. A method for generating simple graphs can be easily extended into a method generating graphs with colored vertices. Colors would be assigned by integers. The integer corresponding to a color of the vertex would be concatenated after each row of the upper-triangle part of adjacency matrix; then the maximality of code based on this expanded matrix will be compared. Moreover, there is the possibility to start with generation of simple graphs, to determine their automorphisms, and then to generate colored multigraphs by coloring vertices and "multiplying" edges of simple graph. The automorphisms of simple graphs would be used to delete isomorphic solutions, taking into account restrictions of mappings based on preservation of colors of vertices and multiplicities of edges. Such an approach has already been developed for Dendral, using double cosets [6]; the automorphism occurring in coloring or multiplying edges has also been studied [4]. However, this approach would probably be in most cases less efficient, as in chemistry most colors of vertices (kinds of atoms) cannot be exchanged – different kinds of atom differ in restrictions on valence conditions like maximum achievable degree (valence) and restrictions on multiplicity of bonds. Exceptionally efficient is construction studied by Laue [12].

2. Methods of pruning the search tree

2.1. Removing nongraphical valence sequences and, from the permutations of graphical sequences, removing ordered sequences that cannot lead to a maximum code or to a connected graph

First we start with the generating process. Given the numbers of vertices and edges, we can produce sets of graphical sequences of vertex valences. Necessary and sufficient condition for a sequence to be graphical (i.e. there exists a graph with degrees of vertices producing this sequence) was given by Havel [14] and later by Hakimi [13]. A non-increasing sequence of degrees (when the first number is smaller than the number of vertices) is graphical if and only if the sequence created by removing the first degree equal to k and by decreasing next k degrees by 1 is also graphical. This theorem may be used in a recurrent algorithm, based on repeating three operations: ordering, deleting the first degree and decreasing. The algorithm runs until we get a sequence of zeros, in which case the sequence was graphical, or until we obtain a negative degree for the nongraphical sequence.

Given a sequence of valences, we always start with the first vertex with maximum valence. Then we produce in all possible combinations reordering of the rest of given valence sequence. From these ordered sequences can be excluded those ones where vertices of valence one joined to the first vertex are labeled lexicographically lower than any vertex of higher valence joined to the first vertex. Also, if we are going to generate only connected graphs, any ordered sequence of valences of vertices $r_1, \dots, r_k$ must satisfy the inequality

$$\sum_{i=1}^{k} r_i > 2(k-1) \quad \text{for } k = 1, \dots, n-1.$$

2.2. *Generating "valence states" sequences for multigraphs and accepting only the graphical ones*

The condition to be satisfied for multigraphs with bounded valence of vertices was described in [8]. When the vector of valences $R = (r_1, r_2, \dots, r_n)$ is monotone with even sum and p is an upper bound for multiplicity of an edge, there exists a multigraph corresponding to this valence vector if and only if

$$\sum_{i=1}^{k} r_i \leqslant pk(k-1) + \sum_{i=k+1}^{n} \min\{r_i, pk\} \quad (k = 1, 2, \dots, n).$$

However, as the valence restrictions in chemistry tend to be strong, this condition would not be of much help for constructing molecular graphs.

Nevertheless, in chemistry additional boundary conditions often occur, mainly knowledge of the valence state of an atom, which could be described as a sequence of numbers of edges of certain multiplicities the vertex is incident with. In chemistry, the maximum multiplicity of an edge is usually three and the maximum degree for most atoms occurring in organic chemistry is four. Therefore it is useful to introduce another criterion for sequence of vertices to be graphical, when the sequence is given not only by simple degrees of vertices, but by separate degrees for each multiplicity of edges for each vertex. For the maximum multiplicity of edges equal to three and the maximum degree of vertices equal to four, necessary and sufficient conditions for sequence of "valence states" of vertices to be graphical have been described in [20]. A set of valence states is graphical if and only if the following degree sequences are all graphical: degrees for single edges, double edges, triple edges, degrees of single edges

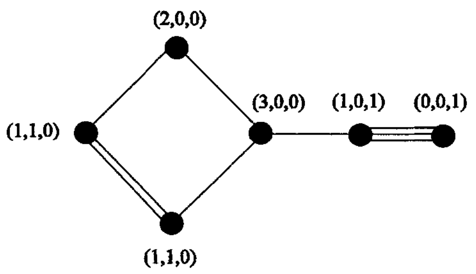


Fig. 1. Multigraph has at each vertex attached valence states describing number of single, double, and triple edges incident with the vertex. Separate nonincreasing sequences for single edges would be (3,2,1,1,1,0), for double edges (1,1,0,0,0,0), for triple edges (1,1,0,0,0,0), for single and double (3,2,2,2,1,0) and for single and triple (3,2,2,1,1,1). All these sequences are graphical. For example, for the last sequence it could be proved as follows: ($\underline{3}$,2 − 1,2 − 1,1 − 1,1,1) → (1,1,0,1,1) → ($\underline{1}$,1 − 1, 1,1,0) → (0,1,1,0) → ($\underline{1}$,1 − 1,0,0) → (0,0,0).

combined with double edges, when double edges are considered equivalent to single edges, and similarly degrees of single edges combined with triple edges (Fig. 1).

Actually, when we are generating labeled graphs satisfying ordered valence sequence, this condition comes prior to lexicographic ordering. It means that further produced graph with "smaller" ordered valence sequence can actually have a bigger code than the presently generated graph [19, 22]. However, this condition does not cause any algorithmic difficulty. To make the concept of producing graphs in lexicographically decreasing order formally clear, we could place the degrees on the main diagonal of adjacency matrix, and include these entries into upper-triangle part of adjacency matrix used for creating the code.

2.3. Semicanonical labeling

Graphs with n vertices can be described by (0, 1) $n \times n$ matrices with zeros on the diagonal [17]. The maximum code made up from an upper-triangle part of adjacency matrix can be achieved only if the corresponding graph is semicanonically labeled. Semicanonical labeling was introduced (intuitively and not under this name) by Faradzhev [10]. It is based on ordered partitions of vertices of graphs. This labeling enables both the canonicity check as well as the construction of graphs to be done by levels, i.e. in the kth level of construction of a graph we place edges between vertex labeled k and vertices corresponding to higher labels (i.e., unfinished rows) [21, 28].

The construction of graphs is done row by row. First, the sequence of valences of vertices is generated. Then this sequence is ordered in all possible manners, satisfying the conditions given in Section 2.1. After choosing the vertex valence for the current row i, we must deduct from it the sum of entries already contained in the ith column. The resulting number of edges is distributed throughout the ith row beginning with $i + 1$ entry. First we start with all edges (entries) cumulated at the left-hand side of the row. In further branches of the search tree we would shift edges to the right to get smaller code. This shift must satisfy conditions of semicanonical labeling, i.e. the

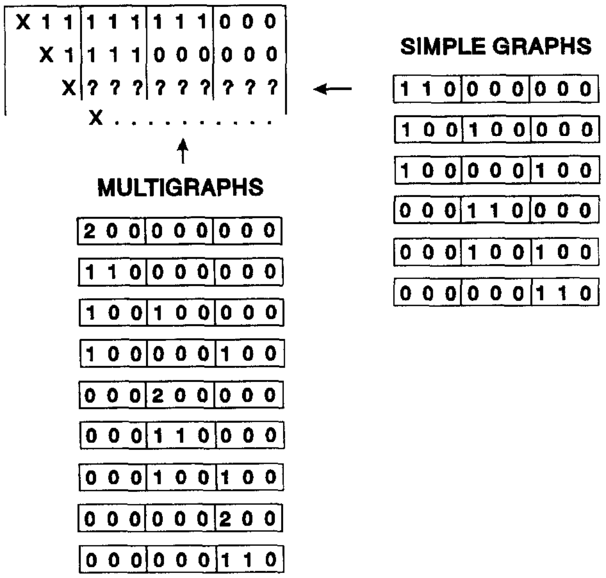


Fig. 2. Construction of a new row based on rules of semicanonical labeling. Two rows have been already filled and we should place two edges in the third row. Vectors over the third row can be divided into three parts: $\binom{1}{1}$, $\binom{1}{0}$, $\binom{0}{0}$ and the equivalent entries in the third row are divided accordingly. On the right hand side is a list of possible distribution of the two edges in the third row. Possible instances for multigraphs are in the lower part of the figure. However, in that case the first entry of the matrix should be of maximal multiplicity to get maximum code.

equivalence of edges based on the graph constructed in the previously generated level of matrix is taken into account. The row is divided into parts (sets) according to the columns upon each entry. When two consecutive entries of the row have equal vectors corresponding to the columns in which the row entries lie, one can say from the point of view of graph symmetry, that the edges corresponding to those entries are equal. Let us have m such sets. So when we fill k entries in the row, we distribute k in all possible ways within m sets. As the labeling should be maximal, the set which obtained p entries from k will place these entries at the leftmost positions. The entries in the lower, unfilled rows, are for the canonicity check supposed to be filled with zeros.

This approach allows us to check the canonicity of the produced graph after generation of each level (Fig. 2).

The generation of multigraphs proceeds substantially in the same way as generating of graphs, except that when we are starting to fill a row, placing k edges in graph, there are no first k entries filled by number one, but the first entry equals k (when it complies with degree restrictions) and the rest equal zero. Then we proceed similar to the construction of simple graphs. The row is divided into parts according to the columns above row entries, and the partitions are filled from the left, but now the first entry equals the prescribed sum of number of edges, other entries are equal to zero, and then edges are "spread" throughout following entries. For each subset (defined by equal upper columns) the entries must be in non-increasing order.

2.4. Checking of canonicity after filling each row of the upper-triangle part of adjacency matrix–backtrack canonicity check

Let us have a generated graph given by its generated code. To check whether the code is maximal, we should relabel its vertices in all possible ways and compare the outcoming codes with the initial one. However, semicanonical labeling helps to cope with this combinatorial explosion.

The first vertex must be one of maximum valence. After choosing the first vertex, the rest of the vertices are divided into two sets; first the vertices joined with the first vertex, then the rest of the vertices (if there are any left). Then we pick up randomly from the first set the second vertex and divide the remaining vertices in the set into two sets according to their adjacency to the second vertex. The second set is divided in the same way, vertices adjacent to the second vertex first, then the rest of the vertices. We can follow this approach recurrently, until the whole graph is labeled. However, each time a vertex is labeled, a row corresponding to the adjacency of the just-labeled vertex is defined. Therefore, each time we label a vertex, we can compare the code just produced with the initial code, comparing only current rows. If the new code is larger, then the initial labeling was not canonical and we can generate another graph. When the new code is smaller, then the newly proposed labeling is not canonical and we can try another labeling, until all possibilities are exhausted. When the new code is equal to the initial code, we can continue in further labeling, or, if the labeling is finished, we have found an automorphism.

When we have to label the next vertex and there are several vertices to choose from, we can choose that one which is connected with prospective higher-labeled vertices in such a way that the row would be the largest one. However, as this condition is just going to be checked, its implementation is not efficient. At least, we can divide the first set of unlabeled vertices on vertices connected with other unlabeled vertices and on the other set composed of vertices not adjacent to unlabeled vertices, Also, as we are going on in construction by levels, determined by graph distance from the first vertex, if we are at the beginning of the last level, the vertex has probably joined more "unfinished" vertices than the "finished" ones. As the structure of "unfinished" vertices is not very connected, the current vertex is not probably the hot candidate for the lexicographically low label. More probably, it is going to be also in the last level in the newly proposed labeling, so even the "probabilistic approach" could be applied in choosing low labeled vertices. Of course, this approach would have to be abandoned in the check of the finished graph.

When the canonicity is checked and confirmed, we have the available automorphisms, which can be used in a preliminary pruning by predictions of zeros in further levels of construction of graph (if there are still some edges left to place) (Fig. 3).

It would be unwise to construct graphs by filling the whole upper-triangle part of adjacency matrix first and then check the canoncity, when it is possible to check canonicity after filling each row from top to the bottom.

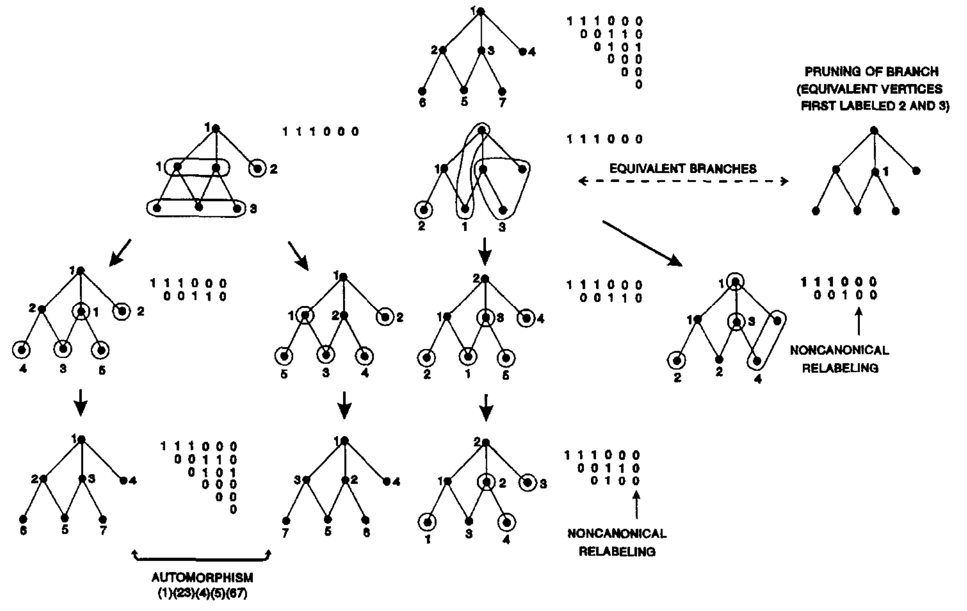


Fig. 3. Search tree for canonicity check. The graph on the top was generated, and its code must be searched, if other relabeling does not produce a bigger code. The first vertex to be labeled must be of the highest degree. Next to the graphs are given only parts of upper-triangle of adjacency matrices used for comparison with original upper-triangle matrix. Relabeling on the right hand side of the figure is pruned at the beginning, with use of automorphism found between first and second labelings. By encircled dots or areas are described unlabeled vertices or their sets to be further labeled. These sets are ordered, and their order number start from one, as well as labels of vertices. From the first set is chosen a vertex to be labeled in the next level. The partition uses not only rules for semicanonical labeling but also division of unlabeled vertices connected to the highest labeled vertex. In the first set there are vertices connected with other unlabeled vertices, the second set contains the rest of vertices connected to the highest labeled vertex.

When the first i rows of upper-triangle part of adjacency matrix are filled, canonicity of the corresponding unfinished graph will be checked before generating $i + 1$ row. The rows, which are not yet filled, are supposed to have all entries equal to zero. Then such canonicity check is not different from the canonicity check of a finished graph. The graph in Fig. 3 can be considered as unfinished graph with only 3 rows filled. This approach has the advantage that when the “unfinished” matrix is not canonical, we do not need to continue with filling the rest of the rows. As the matrix could be probably filled up in several ways, we would remove the generation and checking of several noncanonical matrices.

It is also possible to check canonicity after each filling of an entry by 1, starting from the leftmost position. This may also cut the branches, as in the case of noncanonical labeling we would not need to fill the rest of the subsets in the current row with any combination of remaining entries. However, when the valences of vertices are small, like in chemistry, this approach is not computationally efficient. Also, when we are using proof of noncanonicity to jump over further noncanonical solutions (see Section 2.7), this approach would become redundant.

2.5. Usage of found automorphisms to cut branches in further canonicity checks by removing some possibilities of labeling

Using automorphism group acting on the generated graph helps to avoid unnecessary further checking of canonicity and to introduce restrictions on further generated graphs, for positive as well as negative outcome of canonicity check. Orbits of this group play a fundamental role in the development of the algorithm. However for more deep search it would be necessary to save also exact automorphism mappings. The rejection process in canonicity check as well as in construction is essential, when the search level is not far from the root, because larger branches in the search tree can be pruned.

When all vertices are labeled in such a way that the produced code is the same as the initial one, we arrive at the automorphism mapping. This mapping can be used in a further check of canonicity of the initial labeling for pruning those labelings, where the labels up to the current level are mapped onto themselves and the current level label is mapped lexicographically onto the higher label (see last branch of Fig. 3).

Nevertheless, the complete isomorph rejection can often prove more expensive than the full search as it requires storing and further checking of automorphism mapping.

Orbit can be described as a subset of vertices (or edges) of the graph, where two vertices (edges) belong to the same orbit or cell of the partitioning if there exists an automorphism mapping one vertex (edge) onto the other.

When the vertices are colored or differentiated by their degree or otherwise, we can start determination of orbits from the already given partition, determined in turn by these additional conditions. Then it would be profitable to change the canonical labeling such that the first vertices come from the smallest cell of the given partition, followed by the vertices from the second smallest cell, and so on. By choosing the labeling restricted by the size of cells in increasing order, fewer nodes of the search tree may need to be examined. However, this approach would demand to rewrite the core of the algorithm. That could be profitable only for extensive generating of special types of graphs.

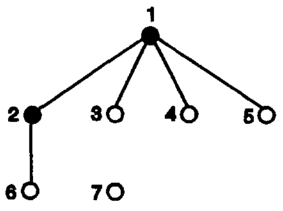
Let us suppose that during the checking of a labeling α we arrive at another labeling β, which produce the same upper-triangle part of adjacency matrix as α. From labelings α and β we can determine an automorphism that can be further used for cutting the search tree for canonicity check. The automorphism leads to the partitioning of the set of vertices into subsets of equivalent vertices. When we label vertices consequently, on each level we can usually choose from several vertices to be labeled next. This choice is restricted by the easily tested requirement of semicanonical labeling. Two vertices will be called equivalent if they belong to the same cell of the automorphism partitioning. Using the set of automorphisms found, we can further reduce the number of vertices available for the current level by deleting choice of vertices that have as available their equivalent counterparts with lower label. For this matching of available vertices we use also vertices that have been currently labeled at that level. However, this reduction can be done only for those equivalent vertices, whose equivalence is based on automorphism that maps all vertices in previous levels on themselves. Unfortunately, the efficiency of this approach largely depends on implementation.

Best results have been obtained using only just-found automorphism for deletion of the choice of labeling of equivalent vertices in previous levels without saving the automorphism. This means that when we return back towards the root node and start to search another branch, we cannot use already found automorphisms to delete some search branches. Indeed, this is, in general, for reasonably large graphs, faster than saving and checking of automorphisms.

2.6. Prediction of zeros in further levels, which is based on equivalence of vertices

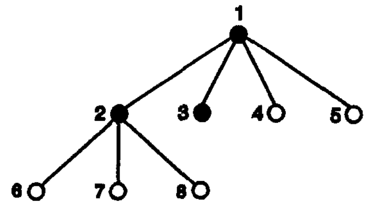
Automorphisms can be used to cut branches in further construction of a graph by predicting some zeros (or generally introducing some boundary conditions) in lower rows of the upper-triangle part of adjacency matrix than is the current construction level.

Let us have a graph constructed up to the kth row of upper-triangle part of adjacency matrix. If the column corresponding to the current level row is the same as the following columns, then the vectors corresponding to the following rows must be less than or equal to the vector corresponding to the current level row (Fig. 4).



$$\boxed{\begin{array}{ccccccc} X & 1 & 1 & 1 & 1 & 0 & 0 \\ & X & 0 & 0 & 0 & 1 & 0 \\ & & X & ? & ? & ? & ? \\ & & & X & ? & ? & ? \\ & & & & X & ? & ? \\ & & & & & X & ? \\ & & & & & & X \end{array}} \longrightarrow \boxed{\begin{array}{ccccccc} X & 1 & 1 & 1 & 1 & 0 & 0 \\ & X & 0 & 0 & 0 & 1 & 0 \\ & & X & 0 & 0 & ? & ? \\ & & & X & 0 & ? & ? \\ & & & & X & ? & ? \\ & & & & & X & ? \\ & & & & & & X \end{array}}$$

Fig. 4. Usage of equivalence of vertices in previous levels for prediction of zeros in unfilled rows. Heavy dots mark vertices corresponding to filled rows, vertices marked by empty circles correspond to unfilled rows. After filling the first row of upper-triangle part of adjacency matrix, the vertices 2–4 are equivalent. Therefore, after the second row is filled, the corresponding parts of the third and fourth rows should be smaller or equal to the second row. This means that the first zeros can be filled in forward, and the last sixth and seventh positions in the third and fourth rows should be smaller or equal to (1,0) which is in the second row. This means, that filling the third row by 0 0 1 1 or filling the fourth row by 0 1 1 is forbidden.



$$\boxed{\begin{array}{cccccccc}
X & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & X & 0 & 0 & 0 & 1 & 1 & 1 \\
 & & X & 0 & 0 & 0 & 0 & 0 \\
 & & & X & ? & ? & ? & ? \\
 & & & & X & ? & ? & ? \\
 & & & & & X & ? & ? \\
 & & & & & & X & ? \\
 & & & & & & & X
\end{array}}
\longrightarrow
\boxed{\begin{array}{cccccccc}
X & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & X & 0 & 0 & 0 & 1 & 1 & 1 \\
 & & X & 0 & 0 & 0 & 0 & 0 \\
 & & & X & 0 & 0 & 0 & 0 \\
 & & & & X & 0 & 0 & 0 \\
 & & & & & X & 0 & 0 \\
 & & & & & & X & 0 \\
 & & & & & & & X
\end{array}}$$

Fig. 5. Usage of current equivalence of vertex corresponding to filled row with vertices corresponding to unfilled rows. Let us have graph described by matrix filled up to the third row. As vertices 4–8 have automorphisms mapping them onto vertex 3, and the third row is already filled, the fourth to eight rows must be filled with zeros.

Let us also have an automorphism determining equivalent vertices. If the vertex corresponding to some of the filled row (including the kth one) is mapped by automorphism on the vertex corresponding to the lower (unfinished) row than is the current (finished) one, then fill the unfilled entries of the lower row by zeros (Fig. 5).

Generally, we could store automorphisms from the previous levels, and as in the previous level some unfinished vertices were mapped on each other, and in the current level one of those vertices becomes ‘finished’, it imposes boundary conditions also for the other “unfinished” vertex. Unfortunately, storing and retrieval of this information does not make this approach effective.

2.7. Using the proof of noncanonicity to skip other noncanonical solutions

Let us have noncanonical labeling of vertices β (with corresponding labeling of edges β') and canonical labeling of vertices α (for edges α'). A labeling is not canonical when there exists such a relabeling that the code corresponding to upper-triangle part of adjacency matrix is bigger for the new relabeling. It means that in the code the first m entries are equal, and the $m + 1$ entry is bigger for the new relabeling. This first different entry will be called “critical”. This entry was transferred to its current position by relabeling from some previous position. When its original position in the code was on the right to its current position, it would be necessary to change the original entry to zero to get a new graph with lexicographically smaller code, which may be canonically labeled (Fig. 6).

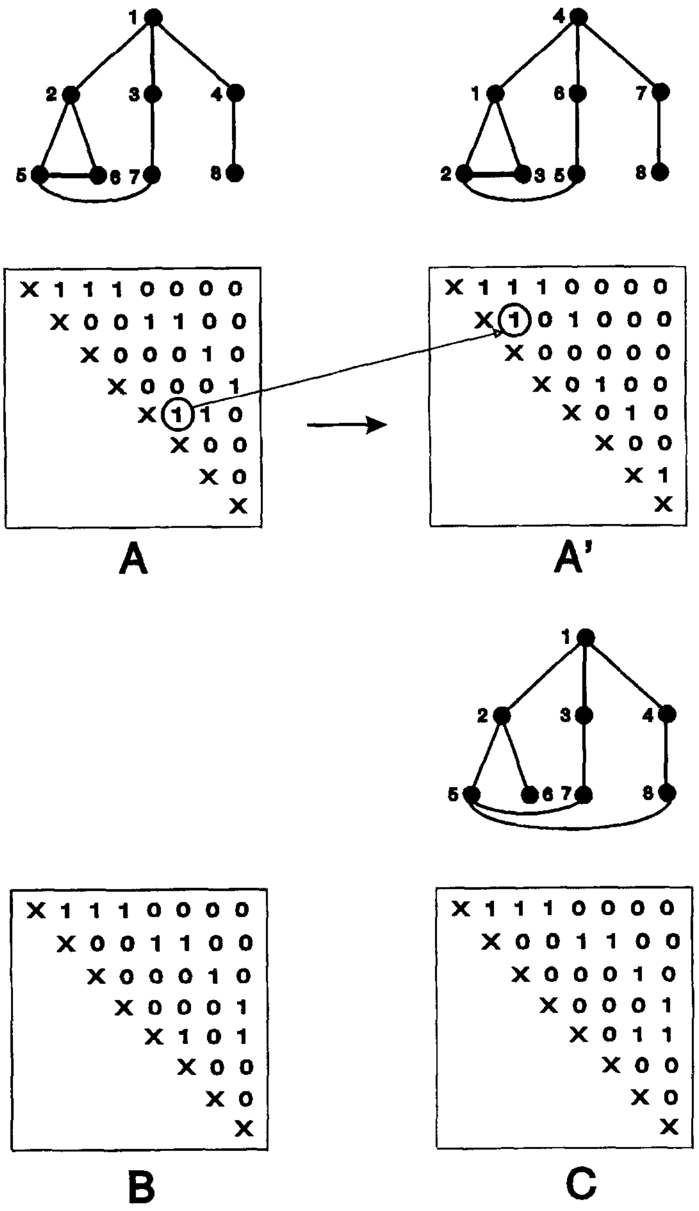


Fig. 6. The entry $\{5,6\}$ in noncanonical matrix A is in matrix A′ (corresponding to canonically relabeled graph) transferred to entry $\{2,3\}$, which is critical for proof of noncanonicity of A. Therefore, in the next generated matrix the entry $\{5,6\}$ should not be equal 1, as it is in matrix B, but this entry should be zeroed as in matrix C. The matrix B would not be generated.

However, this condition may be overridden by another possibility. Let the relabeling be done in such a way that some nonzero entry to the right of the "critical" position in the original labeling is mapped to the left of the "critical" position by the new labeling. Then zeroing this nonzero entry placed in the code to the right of critical

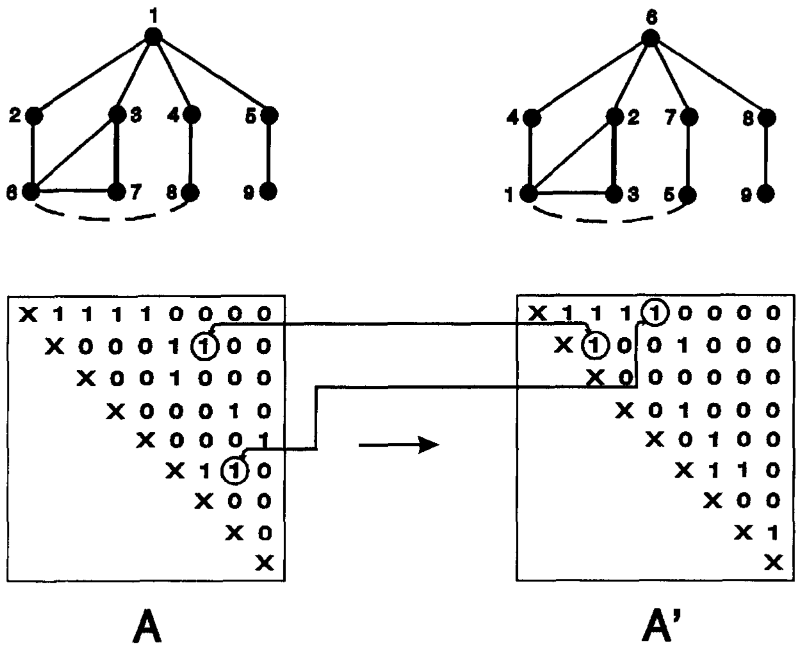


Fig. 7. Similar example as in Fig. 6, the entry causing noncanonicity is $\{2,7\}$ in noncanonical matrix A. This entry is transferred to critical entry $\{2,3\}$ in matrix A′. However, the entry $\{6,8\}$ in matrix A is transferred to $\{1,5\}$ in matrix A′. The corresponding edges in graphs are marked by interrupted lines. As the entry $\{1,5\}$ is previous to $\{2,3\}$, the next matrix should have the entry $\{6,8\}$ zeroed.

position will produce smaller code, which may be canonical. The "spoiled critical entry" would be in the new mapping "beaten" by changing some previous entry to zero. So to get the next (hopefully canonical) code, smaller than the current noncanonical one, but biggest of these smaller codes, we put to zero either the entry which after relabeling "spoiled" canonicity, or the rightmost nonzero entry, which by relabeling was placed before the critical position (Fig. 7).

A similar approach can be used for multigraphs. When there exists a relabeling producing bigger code than the checked labeling, noncanonicity of the original labeling is based on mapping of edge with higher multiplicity onto edge with smaller or zero multiplicity. The next generated graph must have the multiplicity of this mapped edge less than or equal to the value of the original critical entry. However, this condition can be overridden when there exists some higher relabeled edge mapped onto an edge with lower label than the critical edge for canonicity check. Then such an edge with the highest label should have lower multiplicity in the next generated graph.

2.8. Canceling unnecessary canonicity check on the basis of canonicity of previous graph

This approach can be used for unfinished graphs, when there exists in a previously generated branch a graph with equal upper-triangle part of adjacency matrix in

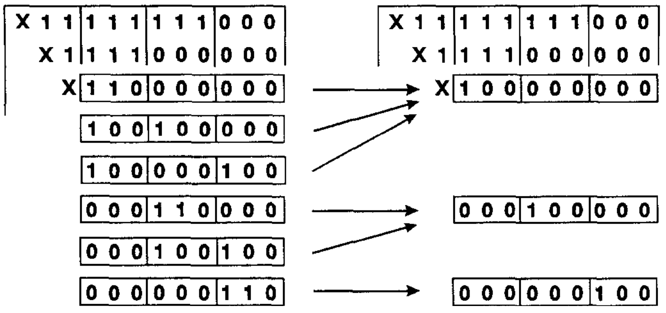

X 1 1 | 1 1 1 | 1 1 1 | 0 0 0
X 1 | 1 1 1 | 0 0 0 | 0 0 0
X | 1 1 0 | 0 0 0 | 0 0 0
1 0 0 | 1 0 0 | 0 0 0
1 0 0 | 0 0 0 | 1 0 0
0 0 0 | 1 1 0 | 0 0 0
0 0 0 | 1 0 0 | 1 0 0
0 0 0 | 0 0 0 | 1 1 0
X 1 1 | 1 1 1 | 1 1 1 | 0 0 0
X 1 | 1 1 1 | 0 0 0 | 0 0 0
X | 1 0 0 | 0 0 0 | 0 0 0
0 0 0 | 1 0 0 | 0 0 0
0 0 0 | 0 0 0 | 1 0 0

Fig. 8. Let us have graphs created in Fig. 2 checked for canonicity. (The rest of the matrix is always supposed to be filled with zeros.) If any graph corresponding to filled row on the left hand side of the arrow is canonically labeled, so it will be true for graph created by replacing the third row by the row on the right hand side of the arrow.

preceding rows and with higher degree corresponding to the current level row. Let us suppose that we have generated first $k + 1$ rows of the upper-triangle part of adjacency matrix, and the $(k + 1)$th row contains p nonempty entries. Let the corresponding labeling be canonical. Now, if we replace $(k + 1)$th row by a new row which differs from the previous one only in the last nonzero entry, zeroed in the new row, then the new graph is also canonically labeled. The same is true also for zeroing any number of last nonzero entries (Fig. 8).

This concept can be used also for multigraphs not only for zeroing but also for reducing multiplicity. This approach both for graphs and multigraphs requires storing information. Therefore, its effective implementation depends to a large extent on the technical manipulation with this information (saving in main memory or on disc) and therefore cannot be generally considered as an effective one. This is true also for other methods demanding storage and checks of information.

2.9. Omitting the canonicity check in the level of construction of the graph, where there is a simple, nonforked (nondivided) branch of the search tree

When we have already filled some entries, and we have given sequence of valences of vertices to be satisfied, the possibilities of filling the rest of entries are restricted. In some cases it is even possible to fill up on the basis of these restrictions some of the unfilled rows, or even the whole rest of matrix. This approach is advantageous, because when we would go "step by step", we would check the canonicity after filling each row, i.e. at every node of a branch of search tree. When we choose to fill the whole matrix at once, we shall check the canonicity only at the end of the nonforked branch. Generally, it is efficient to check canonicity only at those nodes of the search tree where the search tree is forked or at the end of the branches.

The actual prediction of how to fill unfilled entries is done by comparing the minimum degree vector, which would be achieved by filling all remaining unfilled entries by zeros and then counting degrees of vertices, with maximum degree vector,

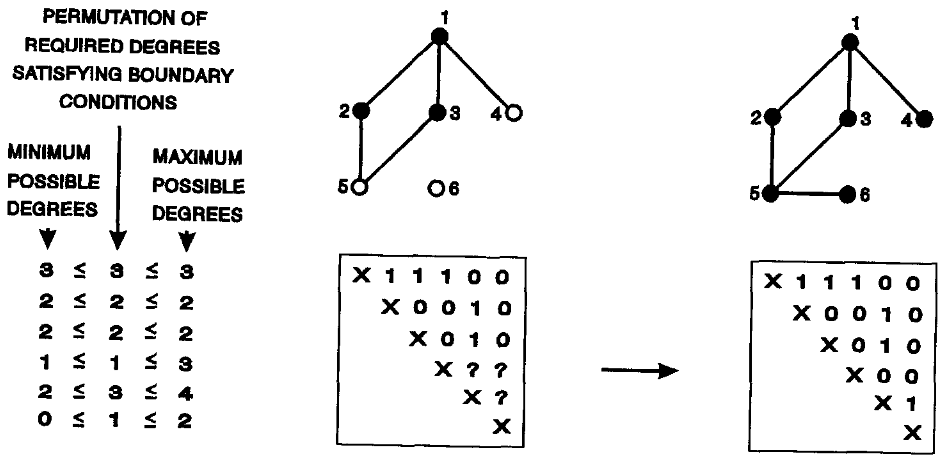


Fig. 9. Filling up the matrix on the basis of the prescribed degree sequence. Let us have the required unordered degree sequences (3,3,2,2,1,1). The matrix has already three rows filled, which leads to maximum and minimum possible degrees. In the only permutation of given degrees, which matches these boundary conditions, degree 1 of the fourth vertex has already been achieved, so the unfilled entries of this row can be filled by zeros. Then there is the only possibility to fill the last entry by 1 to fulfill boundary conditions. By filling two rows at once one canonicity check has been saved.

which would be achieved by filling all the remaining entries by ones. We can try to put a permutation of the prescribed degree vector between these limiting vectors. If for some entry by no permutation of prescribed degrees can be achieved other value than equal to entry in maximum (minimum) degree vector, then we can fill up this row by ones (zeros). Then the maximum and minimum vectors are also changed, and we can continue with the same procedure, until no filling occurs. The situation for multigraphs is in principle the same, only technically complicated by restrictions for each multiplicity (Fig. 9).

3. Conclusions

The algorithmic approach presented stands on a code based on maximum upper-triangle part of adjacency matrix. This code is simple and clearly describable, and by allowing a deeper view into the structure of the graph supports utilization of symmetry in cutting branches both in generating of code, as well as in checking the correctness of the code. Unfortunately, employing all of the principles for pruning the search tree may not bring as high an effect as expected, because these principles often compete between themselves for cutting a branch. In some cases, particularly for graphs with smaller number of vertices, up to about 15, a reduced set of pruning methods can give higher saving of computational time compared with full set of pruning methods. This is especially true for saving automorphisms and using them in further branches of canonicity check.

However, for checking of isomorphism of existing graphs there are more powerful coding approaches. The presented approach is, when picking for example the first vertex, able to divide the rest of the vertices in the structure into only two sets – connected or nonconnected. Further ordered partitioning depends on the choice of the second vertex, and its right choice would need deeper analysis of its neighborhood.

Actually, when we choose the first vertex, we produce ordered partitioning according to the graph distance from the first vertex. This partitioning must be held for the canonical code, which means that first come vertices with graph distance equal to one from the first vertex, then the vertices with graph distance two from the first vertex, etc. However, this rule cannot be applied for further labeled vertices. As the partitioning according to the graph distance comes naturally with semicanonical labeling, it would be redundant to introduce these restrictions previously.

By choosing the first vertex we are given a great amount of easily accessible information about possible further partitions of the graph – for example partitions of vertices according to the distance from the first vertex, equidistant partitions, etc. Choosing the first (second, third, etc.) vertex we can easily derive quite an amount of information.

Unfortunately, using the maximum upper-triangle code, this information can be used only in comparing whether the proposed labeling gives the same code as the original one. If not, it is not clear whether this labeling would give a better or a worse code. In fact, this information does not help in further processing at all.

We could use this information, if we would use a code based also on other principles, such as equidistant partition. Using such a code would help us to cope with automorphisms of graphs and checking the "maximality" – or whatever the demanded property (properties) of the canonical code would be, but the principle of the code would probably disqualify using symmetry in the process of the construction of graphs, e.g. in predicting zeros in the future build up of the graph, or skipping other noncanonical graphs when the current labeling was discovered to be noncanonical. This trade-off would be unprofitable.

We see the possible future development of generation of molecular graphs in a suitably chosen code allowing not only an easy check of canonicity of graph labeling, but also using symmetry of graph in the generating algorithm. The properties of the code must hold during the construction of the graph (based for example on adding edges) so that we can check the required property of an "unfinished" graph (with a lower number of edges than required). When the required property is not satisfied, we would be able to cut those construction branches rooting from the "unfinished" graph, which lead only to noncanonical "finished" graphs.

Another perspective would be a better algorithm for checking isomorphism. There already exist exponential algorithms for checking isomorphism for graphs with degrees bounded by three [2, 7, 24], or for some special types of graphs, like planar graphs. When these methods would be extended to a maximum degree equal to four (valence of carbon, basic element in organic chemistry) it could be quite a breakthrough also for generating of molecular graphs.

Acknowledgements

One of the authors (J.P.) thanks the EC program "GO WEST" for financial support for stay at Faculty of Mathematical Studies in University of Southampton. The author wishes to express his thanks to Dr. E.K. Lloyd for stimulating conversations and for his hospitality while visiting his Faculty.

References

[1] V.L. Arlazarov, I.I. Zuev, A.V. Uskov and I.A. Faradzhev, Algorithm for transformation of finite nondirected graphs to canonical form, Zh. Vychisl. Mat. Mat. Fiz. 14 (1974) 737–742.
[2] L. Balai and E.M. Luks, Canonical labeling of graphs, in: proceedings 15th ACM Symposium on theory of computing (1983) 171–183.
[3] J. Bauer, Igor2: A PC-program for generating new reactions and molecular structures, Tetrahedron Comp. Method. 2 (1989) 269–280.
[4] U. Baumann, Automorphisms of graphs with edge coloring and regular groups of color preserving automorphisms, J. Inform. Process. Cybernet. 23 (1987) 615–618.
[5] E.F. Beckenbach, ed., Applied Combinatorial Mathematics (Wiley, New York, 1964) 5–31.
[6] H. Brown, L. Hjelmeland and L. Masinter, Constructive graph labeling using double cosets, Discrete Math. 7 (1974) 1–30.
[7] J. -Y. Cai, M. Fürer and N. Immerman, An optimal lower bound on the number of variables for graph identification, Combinatorica 12 (1992) 389–410.
[8] V. Chungphaisan, Conditions for sequences to be r-graphical, Discrete Math. 7 (1974) 31–39.
[9] C.J. Colbourn and R. C. Read, Orderly algorithms for generating restricted classes of graphs, J. Graph Theory 3 (1979) 187–195.
[10] I.A. Faradzhev, Generation of nonisomorphic graphs with given sequence of vertices (in Russian) in: I.A. Faradzhev, ed., Algorithmic Investigations in Combinatorics (Nauka, Moscow, 1978) 11–19.
[11] S.W. Golomb and L. D. Baumert, Backtrack programming, J. Assoc. Comput. Mach. 12 (1965) 516–524.
[12] R. Grund, A. Kerber and R. Laue, MOLGEN, ein Computeralgebra-System für die Konstruktion molekularer Graphen, in: A Kerber, ed., Generatoren für molekulare Graphen, Nomenklaturfragen, Match 27 (1992) 87–131. A. Kerber, Algebraic combinatorics via finite group actions (Wissenschafts-verlag, Mannheim, 1991). R. Laue, Construction of Combinatorial Objects – A Tutorial, Bayreuther Math. Schr. 43 (1993) 53–96.
[13] S.L. Hakimi, On realizability of a set of integers as degrees of vertices of a linear graph, J. Soc. Indust. Appl. Math. 10 (1962) 496–502; 11 (1963) 135–147.
[14] V. Havel, A remark on existence of finite graphs, Časopis Pěst. Mat. 80 (1955) 477–480.
[15] J.B. Hendricson and A. G. Toczko, Unique numbering and cataloguing of molecular structures, J. Chem. Inf. Comput. Sci. 23 (1983) 171–177.
[16] A. Kerber, ed., Generatoren für molekulare Graphen, Nomenklaturfragen, Match 27 (1992) 1–216.
[17] V. Kvasnicka and J. Pospichal, Canonical indexing and constructive enumeration of molecular graphs, J. Chem. Inf. Comput. Sci. 30 (1990) 99–105.
[18] V. Kvasnicka and J. Pospichal, Constructive enumeration of acyclic molecules, Coll. Czech. Chem. Comm. 56 (1991) 1777–1802.
[19] V. Kvasnicka and J. Pospichal, Constructive enumeration of molecular graphs with prescribed valence states, Chemometrics Intell. Lab. Syst. 11 (1991) 137–147.
[20] V. Kvasnicka and J. Pospichal, An existence theorem for molecular graphs determined by a sequence of valence states, J. Math. Chem. 11 (1992) 353–364.
[21] V. Kvasnicka and J. Pospichal, An improved method of constructive enumeration of graphs, J. Math. Chem. 9 (1992) 181–196.
[22] V. Kvasnicka and J. Pospichal, An improved version of the constructive enumeration of molecular graphs with prescribed sequence of valence states, Chemometrics Intell. Lab. Syst. 18 (1993) 171–181.

[23] R.K. Lindsay, B.G. Buchanan, E.A. Feigenbaum and J. Lederberg, Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project (McGraw-Hill, New York, 1980).

[24] E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, J. Comput. System Sci. 25 (1982) 42–65.

[25] B.D. McKay, Computing automorphisms and canonical labelings of graphs in: D.A. Holton and J. Seberry, eds., Combinatorial Mathematics. Proceedings of the International Conference on Combinatorial Theory, Canberra (1977), Lecture Notes in Mathematics, Vol. 686 (Springer, Berlin, 1978) 223–232.

[26] B.D. McKay, Backtrack programming and isomorph rejection on ordered subsets, Ars Combin. 5 (1978) 65–99.

[27] B.D. McKay, Practical graph isomorphism, Congr. Numer. 30 (1981) 45–87.

[28] J. Pospichal and V. Kvasnicka, An alternative approach for constructive enumeration of graphs, Coll. Czech. Chem. Comm. 58 (1993) 754–774.

[29] R.C. Read, The enumeration of acyclic chemical compounds, in: A.T. Balaban, ed., Chemical Applications of Graph Theory (Academic Press, London, 1976) 25–61.

[30] R.C. Read, Everyone is winner or how to avoid isomorphism search when calculating combinatorial configurations, Ann. Discrete Math. 2 (1978) 107–120.