Generation of Molecular Graphs for QSAR Studies: An Approach Based on Supergraphs

Andrey A. Melnikov, Vladimir A. Palyulin, and Nikolai S. Zefirov*

Department of Chemistry, Moscow State University, Moscow 119992, Russia

Received May 8, 2007

A new algorithm for generation of substituted derivatives of a given structure and its software implementation are described. The program has deterministic and stochastic generation modes and efficiently supports various types of structural constraints. The problem of exhaustive and irredundant generation is discussed, and a new algorithm of the complete rejection of isomorphic molecular graphs is proposed. The main application of the generator is QSAR studies; however, applications in combinatorial chemistry are also possible.

INTRODUCTION

Computer-aided generation of chemical structures plays an important role in rational drug design. Using a generator program one can form a large series of novel molecular structures in order to estimate their biological activity and to select the most promising structures for further studies.

There exist different types of structure generators for various applications. Over the last decades a rather large number of generators for the structure elucidation purposes have been developed.[1−10] These generators enumerate constructively structural isomers corresponding to a given molecular formula, taking into account a set of constraints such as bounds for ring sizes and the lists of fragments that must be present or absent. However, the generators for structural elucidation usually are not quite suitable for the design of biologically active compounds.

There are several types of generators that are used in drug design. If a biotarget (e.g., enzyme or receptor) and its 3D structure are identified, desired ligands can be obtained using a de novo generator.[11,12] Such a generator forms the structure of a ligand immediately at the binding site. Molecular fragments are docked into the site, scored, and linked together. Another way to search for biologically active compounds is in silico screening of virtual combinatorial libraries. Compounds from the library are docked into the binding site and scored. Such libraries can be generated by combining a number of building blocks. A central fragment or a scaffold has several substitution sites, to which building blocks are attached in the process of generation. Most combinatorial library generators consider the scaffold as an asymmetric molecule, because in many cases the scaffolds are indeed asymmetric. For example, the SMILIB[13] generator simply concatenates the scaffold molecules and building blocks with single bonds, and thus a combinatorial library is created by concatenating SMILES[14] strings. SMILIB can generate about 1 million structures within 30 s. However consideration of a scaffold

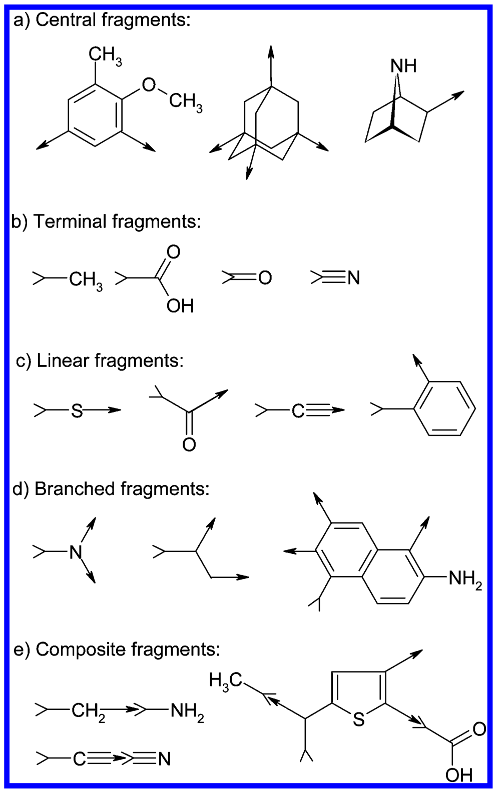


Figure 1. Examples of molecular fragments.

as an asymmetric molecule leads to the generation of duplicates if the scaffold is symmetric. Another generator, MOLGEN-COMB,[15] takes into account symmetry of the scaffold. Generators of combinatorial libraries are included in several commercial software products related to rational drug design.[11,16,17] Some generators of combinatorial libraries use the diversity- and similarity-based strategies to avoid generation of an enormous number of structures. Moreover there exist approaches to the automated structure generation based on genetic algorithms. Genetic algorithms can be used in de novo generators as well as in generators of combinatorial libraries.[18−21]

* Corresponding author e-mail: zefirov@org.chem.msu.ru.

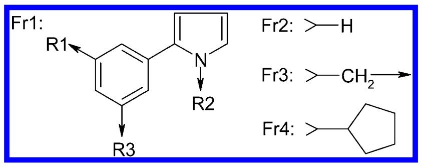


Figure 2. A simple generation task. Constraints: $H_{max}= 2$, $H_{min}= 1$, $M_{max}= 230$, $N_{max}(Fr3, R1) = 0$, $N_{max}(Fr3, R3) = 0$.

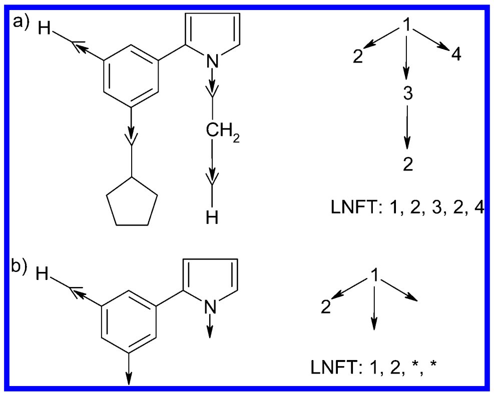


Figure 3. Examples of (a) a completely formed structure, its fragment tree, and LNFT and (b) a partially formed structure, its fragment tree, and LNFT.

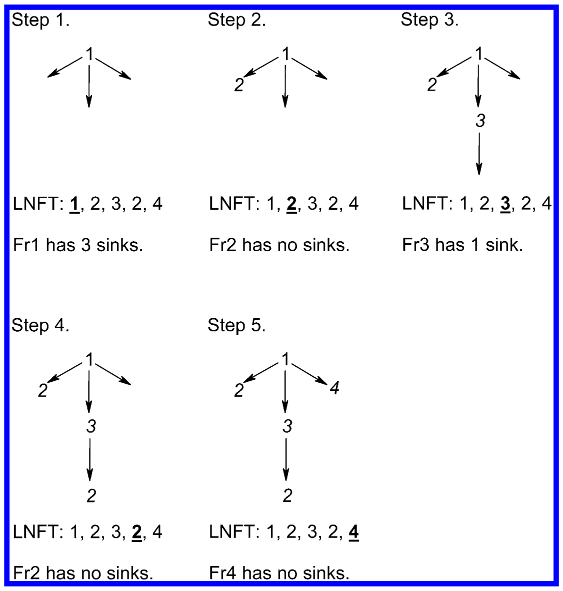


Figure 4. Reconstruction of the fragment tree by its LNFT.

Another field of application of generators in drug design is QSAR studies based on the experimental data on the biological activity of a series of compounds.[22] In the context of the QSAR approach promising compounds can be selected using a QSAR model. To form novel structures for a QSAR-based virtual screening it is preferable to use a generator designed specifically for QSAR studies. Such a generator takes into account the applicability range of QSAR model, so the biological activity of all the generated structures could be adequately evaluated. There are different ways of defining the applicability range of a QSAR model;[23] in particular, the model can be applicable for a single chemical class or for a group of compounds that meet structural rules. Also a QSAR oriented generator should let us avoid the generation of undesirable structures. A generation algorithm is usually developed for a single class of QSAR models since every class of such models defines its own rules for the determination of the applicability ranges. We have considered a rather commonly used class of QSAR models that describes some activity of substituted derivatives of a given structure. In this case the structure generation can be performed by a generator of the virtual combinatorial libraries as well as by a QSAR-oriented generator such as the previously developed GOLD.[24,25] The distinction of GOLD is that this generator not only attaches substituents to the scaffold but also forms substituents from the elementary molecular fragments. Moreover, GOLD supports various structural constraints, which can be specified by predicates. Due to these features the GOLD generator could be an efficient and easy-to-use tool for the QSAR studies. However, GOLD has several shortcomings caused by the limited power of computers at the time of its development. For example, a user of the GOLD generator should encode fragments in the Wiswesser line notation (WLN)[26] and analyze manually symmetry of the fragments.

The goal of this work is to develop an efficient method of completely automatic generation of substituted derivatives of a given structure for QSAR studies. In particular, we concentrated on the problem of efficient handling of constraints and at the problem of irredundant generation. A brief description of the prototype of our generator was published as a short communication.[27,28] In this paper we consider further development of the proposed approach.

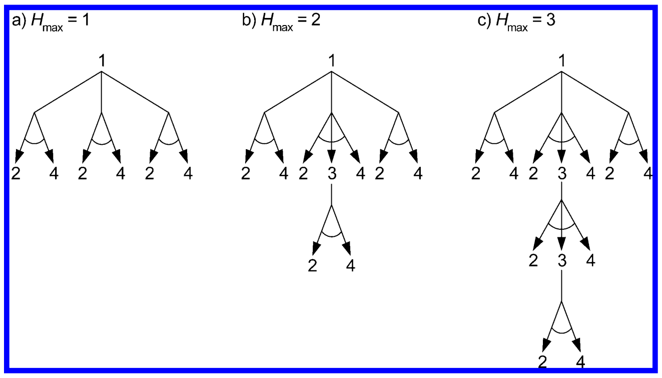


Figure 5. The fragment supergraphs for different values of H_{max}.

Table 1. Generation Table

substitution position	parent fragment no.	parent fragment sink no.	child fragment nos.
R1	1	1	2, 4
R2	1	2	2, 3, 4
R2	3	1	2, 3, 4
R3	1	3	2, 4

BASIC PRINCIPLES AND TERMINOLOGY

The present generator forms substituents from elementary molecular fragments and attaches them to a central fragment (scaffold) that represents a basis compound. Substituents can consist of both single fragments and allowable combinations of several fragments. A substituent can contain several occurrences of a fragment. As far as substituents are created from elementary fragments, a rather large number of substituents can be formed. In many cases it helps to simplify

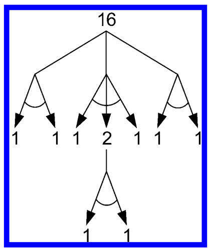


Figure 6. The fragment supergraph labeled by the weight values.

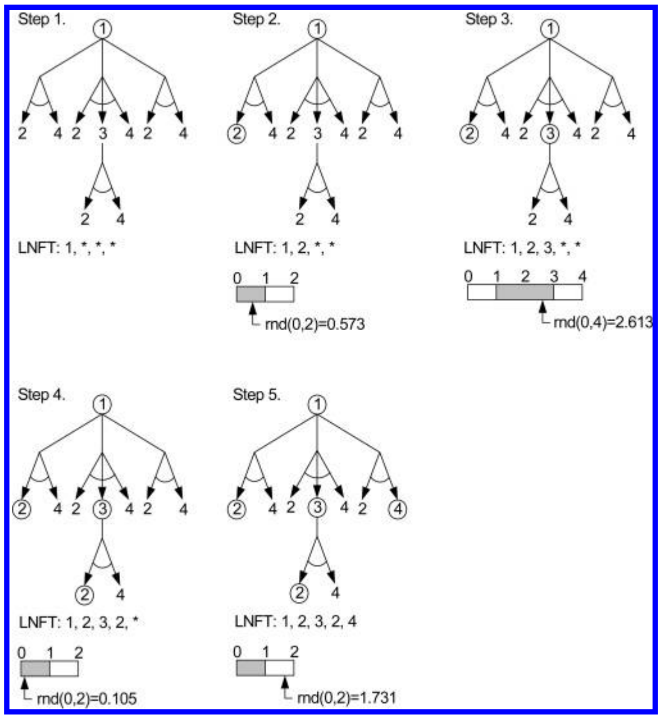


Figure 7. Generation of a structure in the stochastic mode. (Selected vertices are encircled. The schemes under the fragment supergraphs explain the vertex selection procedure using the pseudorandom-number generator.)

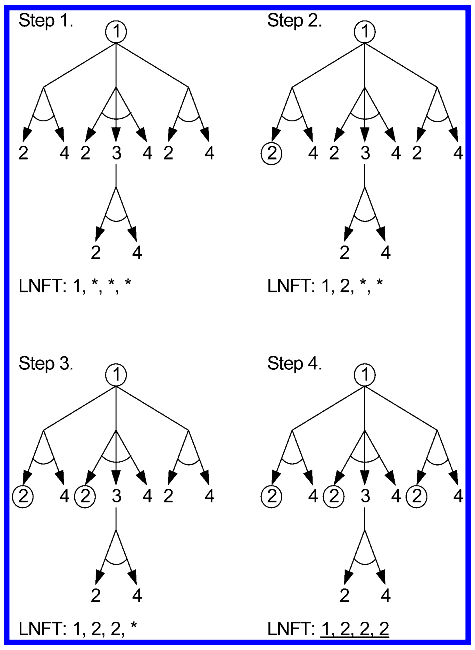


Figure 8. Generation of the first LNFT in the deterministic mode (selected vertices are encircled).

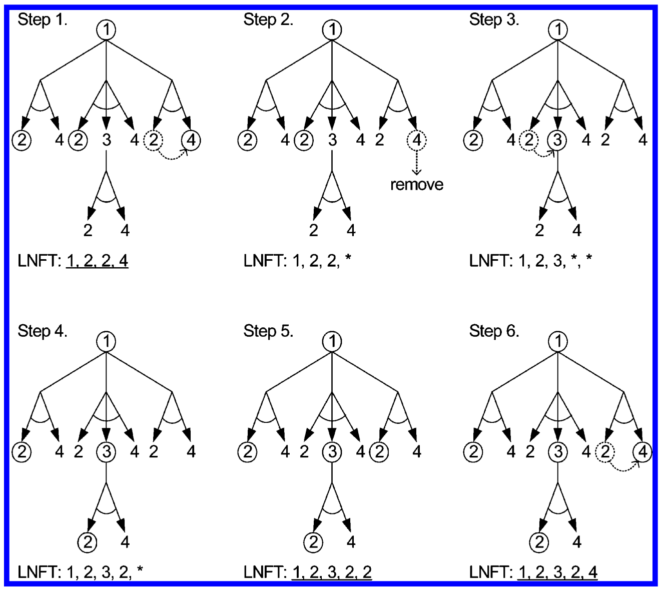


Figure 9. Modification procedure: generation of the second, third, and fourth LNFTs in the deterministic mode.

the input of generation tasks because we can enter just a small set of simple fragments instead of a long list of all necessary substituents. The user should specify some structural constraints to avoid the generation of useless structures or structures, which are out of the applicability range of QSAR model. The graphs of fragments are automatically analyzed at the preliminary generation step, and then the generator manipulates the fragments as integral objects and does not consider their internal structure. Thus the suggested approach makes it possible to achieve a rather high generation rate.

Any fragment can be represented as a molecular graph, which can, in addition to the vertices corresponding to atoms, contain some special vertices, namely sinks and sources that are used for connecting the fragments. Sinks and sources are free valences. A source is designated as a socket connector (—<). A sink is designated as a plug connector (→). By default all possible combinations of the fragments are allowed if the sink and the source of connected fragments have the same bond type. However, it is prohibited to form cyclic combinations of fragments. The formation of cyclic combinations could lead to a generation of structures, which differ noticeably from the structures used for the construction of a QSAR equation, and the prediction of the activity for these compounds by means of that QSAR model can lead to wrong results. Two types of free valences (sinks and sources) were actually defined in order to prevent the formation of new cycles. Moreover all fragments except the central fragment should have exactly one source. The central fragment does not have any sources (Figure 1a). Depending on the number of sinks the elementary fragments are classified as terminal, linear, and branched. Terminal fragments have no sinks (Figure 1b). Linear fragments have exactly one sink (Figure 1c). Branched fragments have more than one sink (Figure 1d). In a pair of connected fragments, the fragment connected by its sink is called a parent fragment. A fragment connected by its source is called the child fragment.

We will use a simple generation task (Figure 2) as an example to illustrate the algorithm of generation.

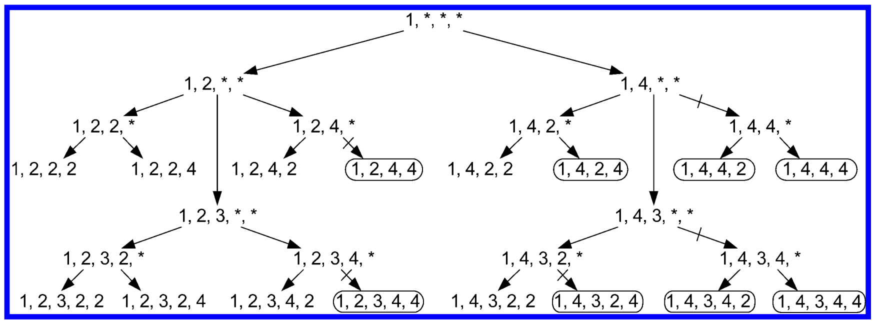


Figure 10. The generation tree (completely formed structures violating the maximal molecular mass constraint are outlined).

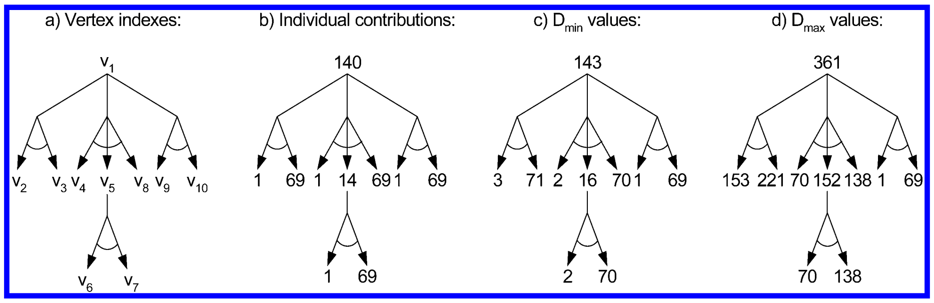


Figure 11. The fragment supergraph labeled by values of individual contributions, D_{min} and D_{max}.

A combination of two or more fragments can be considered as a composite fragment (Figure 1e). A composite fragment having neither sinks nor sources is a completely formed structure. A composite fragment having at least one sink and no source is a partially formed structure. A completely formed structure can be represented as a labeled oriented graph in which vertices correspond to the fragments and edges correspond to the bonds between the fragments (Figure 3a). This graph is a tree because only acyclic combinations of fragments are allowed; therefore, such a graph is hereafter referred to as a “fragment tree”. If the ordinal numbers of the fragments are used as the labels of the vertices of a fragment tree and the numbers of the fragments encountered in the course of the depth first search are written out, we obtain the so-called linear notation of the fragment tree (LNFT). An LNFT for a partially formed structure can also be obtained. In this case free sinks are represented in the LNFT as asterisks (Figure 3b).

From the available LNFT and the known number of the sinks of each fragment, the fragment tree can easily be reconstructed (Figure 4). The first element of the LNFT becomes the root of the fragment tree, and three directed edges are drawn from the root vertex due to the fact that Fr1 has three sinks in our generation task. Then for each succeeding element of the LNFT a new vertex is added to the fragment tree. The new vertex is incident to the first free arc achievable in the course of the depth-first search of the fragment tree under construction. If the corresponding fragment has one or more sinks, then the required number of the directed arcs is drawn from the vertex. Finally the initial fragment tree is obtained from the LNFT. The reconstructed fragment tree can be converted into a molecular graph using molecular graphs of the fragments. Thus an LNFT is a very compact representation of a derivative of the basis compound.

The generator produces LNFTs, from which molecular graphs can be derived. It should be noted that the generator does not form stereoisomers of generated structures. This problem can be solved using other specialized programs, e.g., StereoPlex.[11]

The generator has two generation modes. The deterministic generation mode is used to get all the structures conforming to a generation task. The resultant molecular graphs should be nonisomorphic. The stochastic generation mode is used to form a random representative subset of all possible structures.

The generation task consists of structural formulas of fragments, a list of forbidden fragment combinations, a list of forbidden bonds, some structural constraints, and generation parameters.

A forbidden combination is a pair of connected fragments. The user can either completely forbid the combination of the fragments or forbid connecting of the child fragment to the selected sinks of the parent fragment. A forbidden combination can be specified for all substituents or only for some of them. The list of forbidden bonds also gives an opportunity to control combinations of the fragments. The user can specify bonds that should not appear in the process of generation.

Structural constraints can be specified for a structure as a whole or individually for its substituents. The generator supports the following types of structural constraints: (1) Maximal/minimal height of the fragment tree (H_{max}/H_{min}). Maximal/minimal height is defined as the length of the longest/shortest route from the root to a leaf of the fragment tree. The length of the route is measured as the number of edges in the route. (2) Maximal/minimal branching degree of the fragment tree (B_{max}/B_{min}). Branching degree is defined as the total number of occurrences of terminal fragments. (3) Maximal/minimal total number of occurrences of elementary fragments (T_{max}/T_{min}). The central fragment is not

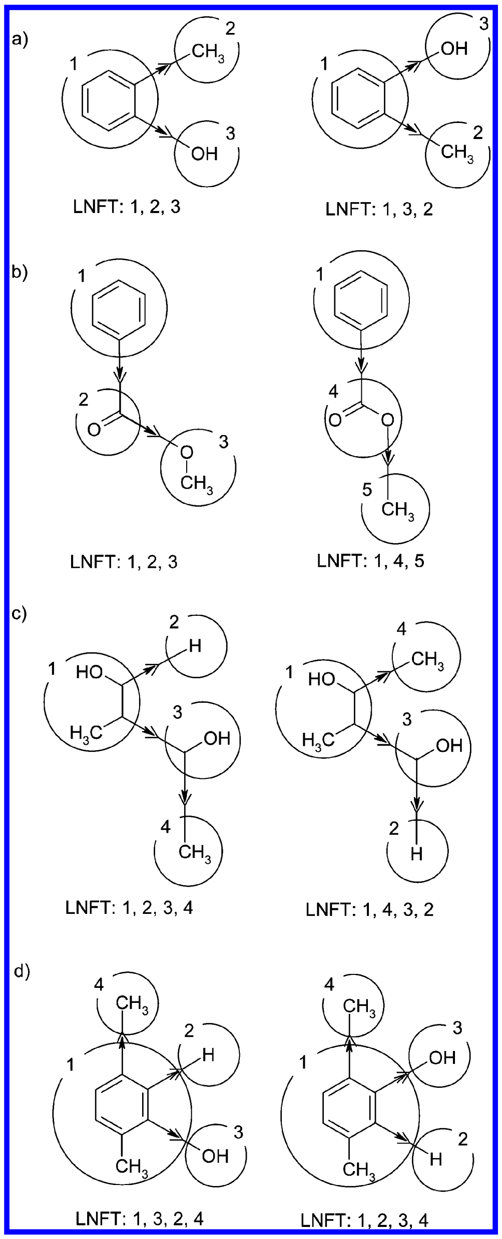


Figure 12. Examples of pairs of different LNFTs corresponding to isomorphic molecular graphs (each pair is related to a separate generation task).

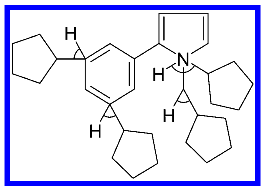


Figure 13. The molecular supergraph.

taken into account. (4) Maximal/minimal number of occurrences of the specified fragment (N_{max}/N_{min}). (5) Maximal/minimal number of substituted positions of the central fragment (S_{max}/S_{min}). (6) Maximal/minimal molecular mass of the structure (M_{max}/M_{min}).

Maximal height and minimal height are the mandatory constraints. Specification of other constraints is optional.

Generation parameters are used to specify a generation mode and some options. For stochastic generation mode a final number of structures to generate should be specified.

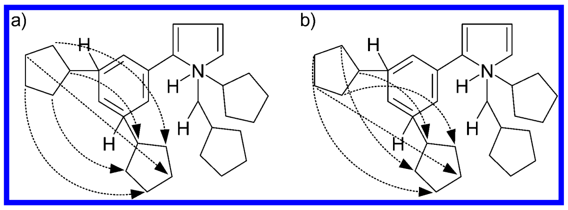


Figure 14. An example of equivalent mappings.

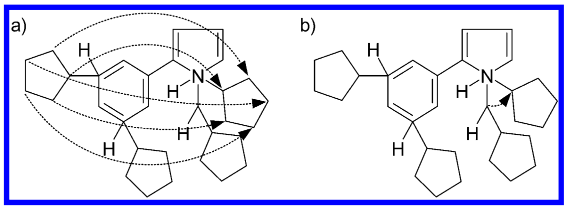


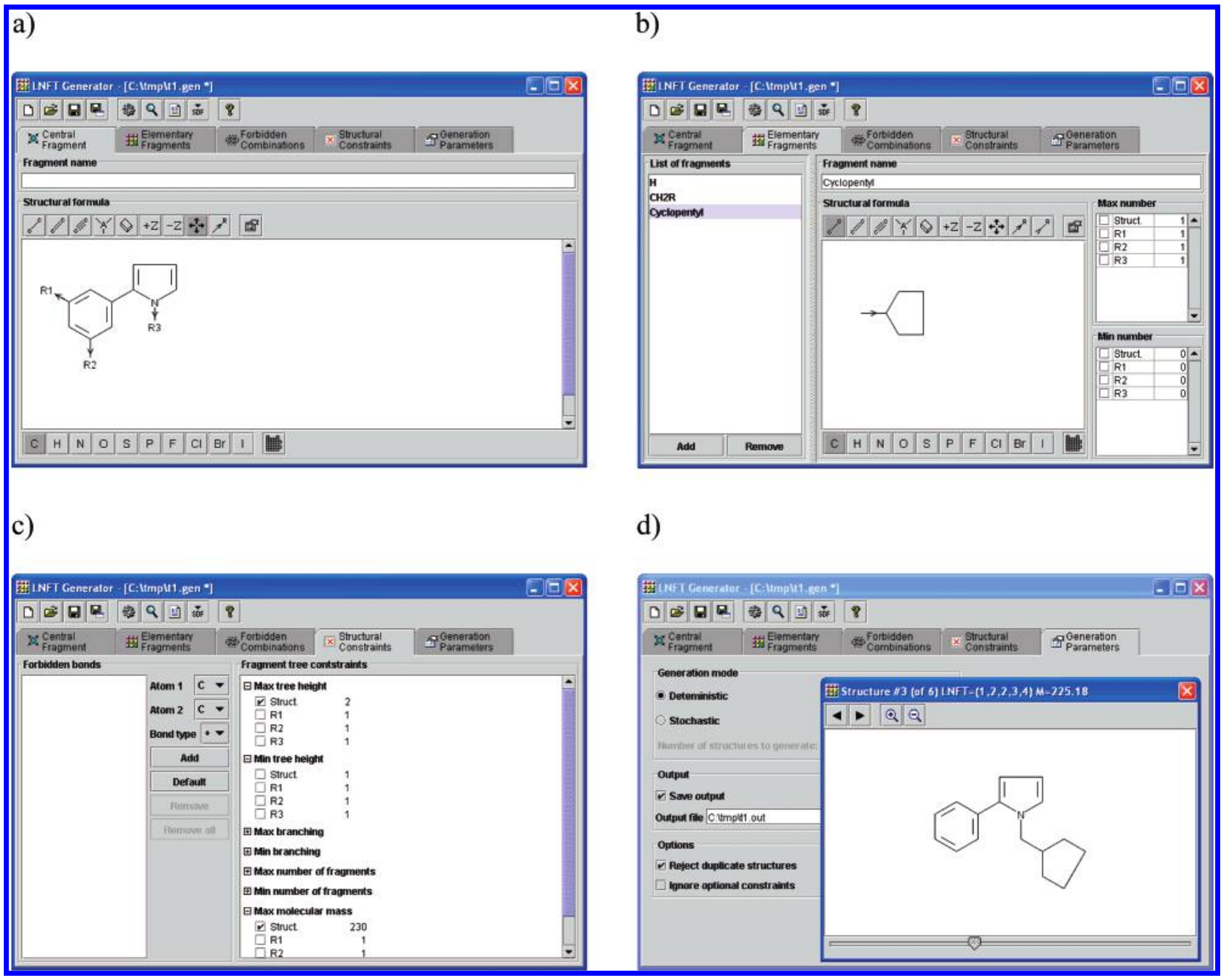
Figure 15. Examples of mappings that can be excluded.

ALGORITHM OF GENERATION

At first we consider the algorithm of LNFT generation without taking into account optional constraints and the problem of duplicate structures generation. At the beginning all the fragments are numbered sequentially, and the so-called generation table is formed by the comparison of bond types of sinks and sources taking into account the list of forbidden combinations and N_{max} constraints if their values are equal to zero. In the generation table, for each sink the sequence numbers of the fragments whose sources can be attached to this sink are listed in the order of fragment numbers. The table has the "Substitution Position" column because some combination of fragments can be allowed for one substituent and forbidden for another. The generation table for our example is given in Table 1. The table has no rows with parent fragment number 3 and the substitution positions R1 and R3, because the generation task contains the constraints N_{max}(Fr3, R1) = 0 and N_{max}(Fr3, R3) = 0.

Using the generation table and the values of the mandatory constraints (H_{max} and H_{min}) the so-called fragment supergraph is formed. A fragment tree is a subgraph of the fragment supergraph if the fragment tree conforms to the mandatory constraints. The vertices of the fragment supergraph corresponding to fragments connected to the same sink of the parent fragment are called competitive vertices. The edges directed to the competitive vertices are marked by a circular arc. The fragment supergraph is an ordered tree because the order of vertices is important for correct generation. The competitive vertices are ordered by the fragment numbers, and the groups of the competitive vertices are ordered in the same way as are the sinks of the parent fragment. Examples of fragment supergraphs built by the generation table (Table 1) are shown in Figure 5. Our generation task corresponds to the second fragment supergraph (H_{max} = 2).

The fragment supergraph is formed by two recursive procedures. The first procedure creates a prototype of the fragment supergraph: at this step, ordered sets of the competitive vertices correspond to the respective rows of the generation table. The second procedure removes irrelevant vertices. A vertex is called irrelevant if it corresponds to a terminal fragment and the length of the route from the root of the fragment tree to the considered vertex is less than H_{min}. Moreover, a vertex is irrelevant if it has at least one free sink. If the supergraph obtained is empty, then it means that no structure meets the generation task requirements.



(b) List of fragments: H, CH2R, Cyclopentyl. Fragment name: Cyclopentyl

Max number	
☐ Struct.	1
☐ R1	1
☐ R2	1
☐ R3	1

Min number	
☐ Struct.	0
☐ R1	0
☐ R2	0
☐ R3	0

(c) Forbidden bonds — Atom 1: C; Atom 2: C; Bond type: •

Fragment tree contstraints

⊟ Max tree height
- ☑ Struct. 2
- ☐ R1 1
- ☐ R2 1
- ☐ R3 1

⊟ Min tree height
- ☐ Struct. 1
- ☐ R1 1
- ☐ R2 1
- ☐ R3 1

⊞ Max branching
⊞ Min branching
⊞ Max number of fragments
⊞ Min number of fragments

⊟ Max molecular mass
- ☑ Struct. 230
- ☐ R1 1
- ☐ R2 1

(d) Generation mode: [✓ Deteministic] Stochastic; Number of structures to generate:
Output: ☑ Save output; Output file: C:\tmp\t1.out
Options: ☑ Reject duplicate structures; ☐ Ignore optional constraints
Structure #3 (of 6) LNFT=(1,2,2,3,4) M=225.18

Figure 16. The graphical user interface of the generator: (a) the central fragment tab; (b) the elementary fragments tab; (c) the constraints tab; and (d) the structure viewer window and the parameters tab.

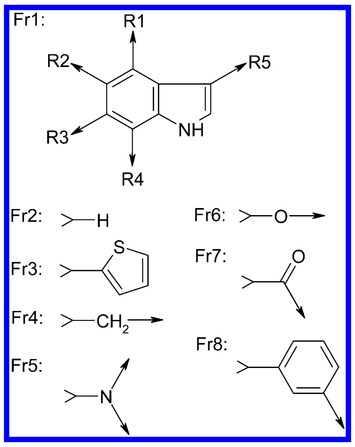


Figure 17. A typical generation task for a QSAR study. Constraints: $H_{max} = 3$, $H_{min} = 1$, $M_{max} = 350$, $T_{max} = 11$, $N_{max}(Fr3) = 1$, $N_{max}(Fr6, R5) = 0$, $N_{max}(Fr7) = 2$, $N_{max}(Fr8) = 1$. Forbidden combinations: Fr7 → Fr2, Fr7 → Fr7. Forbidden bonds: O−O, N−O, N−N.

To generate an LNFT in the stochastic mode a vertex is selected in each group of competitive vertices connected with the root of the fragment supergraph. If a selected vertex is not terminal, then the same procedure is performed for its child groups of competitive vertices. Thus the vertices are selected in the depth-first order. In the course of the vertex selection the LNFT is immediately formed from the left to

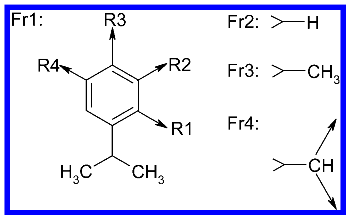


Figure 18. The generation task for the demonstration of the rejection of duplicates. Constraints: $H_{max} = 2$, $H_{min} = 1$.

the right. The labels of the selected vertices are added to the end of the LNFT.

To get a uniform distribution of LNFTs the weights of vertices are used so that the probability of the selection of a vertex is proportional to its weight. The vertex weight of a leaf of the fragment supergraph is equal to 1. For all other vertices the weight is calculated by the following formula: $w = \prod_i \sum_j w_{ij}$, where w_{ij} is the weight of the jth child vertex connected to the ith sink of the considered vertex. The fragment supergraph with the weight values is shown in Figure 6. Figure 7 illustrates the generation of a random LNFT for our example. For each selection we calculate the sum S of all weights in the group of competitive vertices and divide the interval (0, S) into segments. Each segment corresponds to a vertex, and the size of the segment is equal to the vertex weight. Then a random number from the interval (0, S) is generated, and the vertex whose segment spans the random number is selected. Due to the influence of several optional constraints the actual distribution of LNFTs can differ from the uniform distribution. A strictly uniform distribution of LNFTs could be obtained if the vertex weights are recalculated in the process of generation, but this considerably decreases the generation rate.

In the deterministic generation mode LNFTs are also formed from the left to the right. From the beginning the first LNFT is formed, and then each succeeding LNFT is obtained by a minor modification of the preceding one. The generation cycle is completed when any further modification of the last formed LNFT is impossible. To generate the first LNFT we select the first vertex in each group of the competitive vertices connected with the root of the fragment supergraph. If a selected vertex is not terminal, then the same procedure is executed for its child groups of competitive vertices, etc. Thus the vertices are selected in the depth-first order as in the case of the stochastic generation mode. The labels of the selected vertices are added to the end of the LNFT. Figure 8 illustrates the generation of the first structure. Let us consider the modification procedure (Figure 9). Let v_i be the last vertex listed in the current LNFT and v_i' be the next competitive vertex. To modify the current LNFT the label of v_i is replaced by the label of v_i' (step 1 in Figure 9). If v_i is the last vertex of its group of competitive vertices, then the label of v_i is removed from the LNFT (step 2), and the modification procedure is executed once again (step 3). If the result of the modification is a partially formed structure, then the LNFT is completed by choosing the first fragment in every competitive vertex group for all free sinks (steps 4 and 5). If it is required that the first element of the LNFT (the label of the central fragment) be removed, then this means that the further modification is impossible and the generation is complete. Thus for our example 16 LNFTs will be generated in the following order: (1, 2, 2, 2), (1, 2, 2, 4), (1, 2, 3, 2, 2), (1, 2, 3, 2, 4), (1, 2, 3, 4, 2), (1, 2, 3, 4, 4), (1, 2, 4, 2), (1, 2, 4, 4), (1, 4, 2, 2), (1, 4, 2, 4), (1, 4, 3, 2, 2), (1, 4, 3, 2, 4), (1, 4, 3, 4, 2), (1, 4, 3, 4, 4), (1, 4, 4, 2), (1, 4, 4, 4).

ACCOUNT OF OPTIONAL CONSTRAINTS

The mandatory constraints considered above are used at the step of fragment supergraph building. The simplest way to take into account optional constraints is the filtration of the obtained structures. But this way is in fact inefficient, because in some cases most of the structures do not comply with optional constraints. Hence it is necessary to generate only desired structures.

Let us consider the so-called generation tree. The generation tree is a finite, labeled, directed tree, where the leaf nodes represent completely formed structures and the internal nodes correspond to partially formed structures. The labels of the generation tree are LNFTs. The process of the deterministic generation without taking into account any optional constraints could be considered as the depth first walk of the generation tree, and the stochastic generation could be represented as a walk from the root of the generation tree to a random leaf. We could improve the generation algorithm if we can estimate whether a branch of the generation tree contains a leaf that complies with the optional constraints or not. If the estimation shows the absence of such leaves, then the walk of this branch of the generation tree can be rejected (Figure 10).

Suppose that only one optional constraint is specified. Let us define for each vertex of the fragment supergraph an individual contribution to the constrained value. The individual contribution is defined by the type of the constraint: (1) Branching degree. The contribution is equal to 1 if the vertex corresponds to a terminal fragment, 0 otherwise. (2) The total number of occurrences of elementary fragments. The contribution is equal to 1 for any vertex except the root of the fragment supergraph. (3) The number of occurrences of the specified fragment. The contribution is equal to 1 if the vertex corresponds to the specified fragment, 0 otherwise. (4) The number of substituted positions of the central fragment. The contribution is equal to 1 if the vertex corresponds to a non-hydrogen fragment and its parent fragment is the central fragment, 0 otherwise. (5) Molecular mass. The contribution is equal to the molecular mass of the fragment.

If a constraint is specified for a substituent, then the individual contributions for vertices corresponding to all other substituents are equal to 0.

Then the so-called minimal and maximal possible increment of the constrained value ($D_{\min}$ and $D_{\max}$) is calculated for each vertex of the fragment supergraph. If we add a vertex label to the current LNFT and select other vertices to get an LNFT of a completely formed structure, then the constrained value increases. The increment depends on the selection of the following vertices, and, at the same time, by definition, the increment lies within the range of $D_{\min}$ to $D_{\max}$. $D_{\min}$ and $D_{\max}$ are calculated recursively: $D_{\min}(v_i) = C(v_i) + \min_j(D_{\min}(v_j'))$ and $D_{\max}(v_i) = C(v_i) + \max_j(D_{\max}(v_j'))$, where $C(v_i)$ is the individual contribution of the considered vertex v_i, and the set of vertices v_j' is the first group of competitive vertices that can be found after the vertex v_i in the course of the depth first search of the fragment supergraph. The results of the calculation of individual contributions, $D_{\min}$ and $D_{\max}$ values for the molecular mass constraint, are shown in Figure 11 (molecular masses are rounded to integer values). Let us consider in detail some of the calculations (vertex indexes are given in Figure 11a):

$$D_{\min}(v_1) = C(v_1) + \min(D_{\min}(v_2), D_{\min}(v_3)) = 140 + \min(3, 71) = 143$$

$$D_{\min}(v_2) = C(v_2) + \min(D_{\min}(v_4), D_{\min}(v_5), D_{\min}(v_8)) = 1 + \min(2, 16, 70) = 3$$

$$D_{\min}(v_5) = C(v_5) + \min(D_{\min}(v_6), D_{\min}(v_7)) = 14 + \min(2, 70) = 16$$

$$D_{\min}(v_6) = C(v_6) + \min(D_{\min}(v_9), D_{\min}(v_{10})) = 1 + \min(1, 69) = 2$$

$$D_{\min}(v_8) = C(v_8) + \min(D_{\min}(v_9), D_{\min}(v_{10})) = 69 + \min(1, 69) = 70$$

$$D_{\min}(v_9) = C(v_9) = 1$$

$$D_{\min}(v_{10}) = C(v_{10}) = 69$$

$$D_{\max}(v_1) = C(v_1) + \max(D_{\max}(v_2), D_{\max}(v_3)) = 140 + \max(153, 221) = 361$$

$$D_{\max}(v_2) = C(v_2) + \max(D_{\max}(v_4), D_{\max}(v_5), D_{\max}(v_8)) = 1 + \max(70, 152, 138) = 153$$

$$D_{\max}(v_5) = C(v_5) + \max(D_{\max}(v_6), D_{\max}(v_7)) = 14 + \max(70, 138) = 152$$

$$D_{\max}(v_6) = C(v_6) + \max(D_{\max}(v_9), D_{\max}(v_{10})) = 1 + \max(1, 69) = 70$$

$$D_{\max}(v_8) = C(v_8) + \max(D_{\max}(v_9), D_{\max}(v_{10})) = 69 + \max(1, 69) = 138$$

$$D_{\max}(v_9) = C(v_9) = 1$$

$$D_{\max}(v_{10}) = C(v_{10}) = 69$$

Let v_i be a chosen vertex of the fragment supergraph. Before we add the label of the vertex to the end of the current LNFT we estimate the constrained value. Let S_{LNFT} be the sum of individual contributions of the vertices listed in the current LNFT. The upper estimate of the constrained value is equal to $S_{\mathrm{LNFT}} + D_{\max}(v_i)$, and the lower estimate is equal to $S_{\mathrm{LNFT}} + D_{\min}(v_i)$. Let the constraint be defined as the range $[V_{\min}, V_{\max}]$. If the upper estimate is less than $V_{\min}$ or the lower estimate is greater than $V_{\max}$, then we can reject the selection of the considered vertex because the constraint is violated in any case. In the case of the stochastic generation the rejection of the vertex selection means that we try to select another competitive vertex using a random number generator. In the case of deterministic generation we try to select the next competitive vertex. For our example eight LNFTs will be generated taking into account the molecular mass constraint: (1, 2, 2, 2), (1, 2, 2, 4), (1, 2, 3, 2, 2), (1, 2, 3, 2, 4), (1, 2, 3, 4, 2), (1, 2, 4, 2), (1, 4, 2, 2), (1, 4, 3, 2, 2).

If more than one constraint is specified, then the individual contributions, $D_{\min}$ and $D_{\max}$, are calculated for each constrained value. The selection of a vertex is rejected if at least one estimate predicts a constraint violation.

The described method of the optional constraint handling is used for both stochastic and deterministic generation modes.

THE PROBLEM OF DUPLICATES

The above algorithm of deterministic generation produces all possible LNFTs. Each generated LNFT is unique. LNFTs are generated so that each succeeding LNFT is lexicographically larger than the preceding one. However, different LNFTs can correspond to the same molecular graph; therefore, the generation results can contain isomorphic molecular graphs.

We consider three factors leading to the generation of duplicates. The first factor is the symmetry of graphs of fragments having more than one sink (Figure 12a). The second factor is the presence of common subgraphs in molecular graphs. In this case combinations of various fragments can lead to the generation of identical composite fragments (Figure 12b). The third factor is apparently the most complex case. The attachment of a certain elementary or composite fragment to a parent fragment could lead to the appearance of new elements of symmetry (Figure 12c,d). For the example shown in Figure 12d the molecular graphs would be nonisomorphic if the methyl group is not attached to the R3 position of the central fragment. So we cannot completely solve the problem of duplicates using the approach which just takes into account automorphism groups of the branched fragments and the central fragment (such as proposed in refs 24 and 25). The problem could be solved by considering the internal structure of the fragments in the process of generation. However this way is very inefficient. In this work, we propose a new algorithm that recognizes the duplicate structures without their comparison with the previously generated structures.

By definition, the current structure is a duplicate of an earlier generated structure if the molecular graphs of both structures are isomorphic. As far as the generator forms LNFTs in the lexicographical order, the current structure is a duplicate if for its molecular graph an isomorphic graph exists and the isomorphic graph corresponds to an LNFT lexicographically smaller than the current LNFT. To determine whether the current structure is a duplicate or not, a backtracking procedure checks the existence of such an isomorphic molecular graph.

At the preliminary step the so-called molecular supergraph is formed (Figure 13). To form the molecular supergraph the vertices of the fragment supergraph are replaced by molecular graphs of the fragments. The vertices of the molecular supergraph correspond to non-hydrogen atoms. However if the generation task contains the fragment >—H, then the molecular supergraph is supplemented by hydrogen vertices which are implicitly present in the molecular fragments (for the sake of clarity these vertices are not shown in Figures 13−15). The final step of the molecular supergraph formation is the aromaticity detection based on the Hückel's 4n+2 π-electrons rule. Then the molecular supergraph is divided into parts, which correspond to the initial molecular fragments, and the mappings of each such part onto the molecular supergraph are found and analyzed.

The duplicate recognition procedure searches for a set of mappings, which constitute a bijection from the current structure onto a structure with a lexicographically smaller LNFT. If such a bijection exists, then the current structure is a duplicate, and its LNFT is not written to the output file. The procedure uses information about mappings only, i.e., the molecular supergraph is not used at this step, and fragments are considered as integral objects. Returning to our example let us note that the duplicate recognition procedure rejects LNFTs (1, 4, 2, 2) and (1, 4, 3, 2, 2), and so the resultant set of structures consists of six LNFTs. For example, the LNFT (1, 4, 2, 2) is a duplicate because there exists a set of four one-to-one mappings of fragments, and these mappings constitute a bijection onto the molecular graph of the lexicographically smaller LNFT (1, 2, 2, 4). The duplicate recognition algorithm is discussed in the Appendix.

Note that even for simple generation tasks an enormous number of mappings could be found. Efficiency of the procedure decreases as the number of mappings grows, since we use a backtracking procedure for the duplicate recognition. Fortunately, most of the mappings can be excluded.

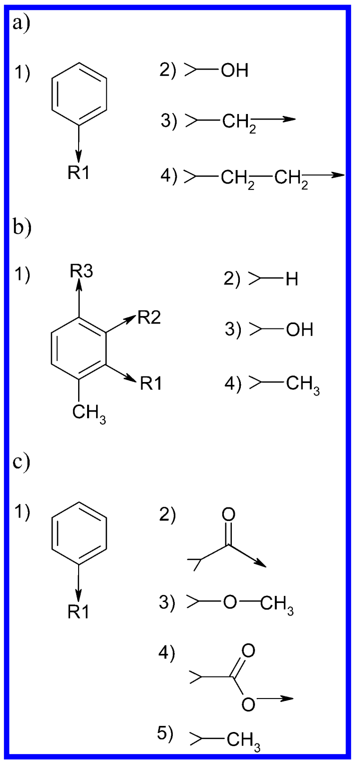


Figure 19. The simple generation tasks and their results (the demonstration of the duplicate rejection). (a) Constraints: $H_{\max} = 3$, $H_{\min} = 2$. Saved LNFTs: (1, 3, 2), (1, 3, 3, 2), (1, 3, 4, 2), (1, 4, 4, 2). Rejected LNFTs: (1, 4, 2), (1, 4, 3, 2). (b) Constraints: $H_{\max} = 1$, $H_{\min} = 1$, $N_{\max}(\mathrm{Fr2}) = 1$, $N_{\max}(\mathrm{Fr3}) = 1$, $N_{\max}(\mathrm{Fr4}) = 1$. Saved LNFTs: (1, 2, 3, 4), (1, 2, 4, 3), (1, 3, 4, 2), (1, 4, 2, 3), (1, 4, 3, 2). Rejected LNFT: (1, 3, 2, 4). (c) Constraints: $H_{\max} = 2$, $H_{\min} = 2$. Forbidden bonds: O−O. Saved LNFTs: (1, 2, 3), (1, 2, 5). Rejected LNFT: (1, 4, 5).

For example, the cyclopentyl fragment in R1 position of the central fragment can be mapped onto the same fragment in R3 position in two ways (Figure 14). The table of mappings contains only one such mapping because these mappings are related to the same permutation of the fragment labels in an LNFT. Also we can exclude the mappings inconsistent with the mappings of the parent fragment. For example, the cyclopentyl fragment in R1 position can be mapped onto the same fragment in the R2 position (Figure 15a), but the central fragment cannot be mapped onto itself so that R1 substitution position replaces R2. At last we can exclude the mappings that have no inverse mappings because such mappings cannot constitute bijections. An example of such a mapping is shown in Figure 15b, and the corresponding inverse mapping does not exist because the cyclopentyl fragment cannot be mapped onto a linear fragment having only one carbon atom.

The proposed approach does not prevent the generation of duplicate structures, but it makes possible recognizing them without the comparison with other structures. In spite of using the brute force approach, in most cases this method is significantly more efficient than the filtration of duplicate structures using a graph isomorphism test and is more reliable than the filtration based on comparison of structures by their descriptors. The efficiency of the proposed algorithm is achieved by the reduction of the search space. The method guarantees the total absence of duplicates in the generation results.

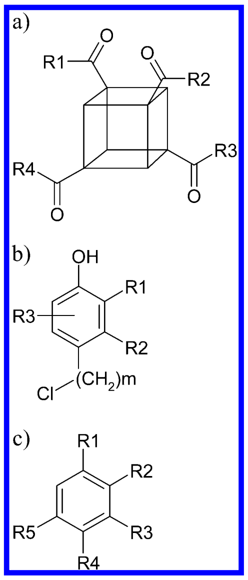


Figure 20. The generation tasks described in refs 15 and 30. (a) Substituents for R1−R4: 20 amino acids. (b) R1: methyl or ethyl (substituent variation), R2: alkyl (homology variation) - 1−6 carbon atoms, R3: amino (position variation), m: 1−3 (frequency variation). (c) R1: methyl, ethyl, OH, R2: alkyl (1−6 carbon atoms), R3: OH, OCH_3, OC_2H_5, methyl, ethyl, R4: OH, CH_2Cl, amino, R5: H, methyl, ethyl, amino.

Table 2. Comparison of Generation Results

generation task	MOLGEN-COMB	LNFT generator
Figure 20a	8855 structures, 601.40 s[a]	8855 structures, 3.51 s
Figure 20b	396 structures, 4.9 s[b]	396 structures, 0.22 s
Figure 20c	5939 structures, 44 s[b]	5939 structures, 0.25 s

[a] The configuration of the computer is not mentioned. [b] PC with a Pentium 4 (2.5 GHz).

SOFTWARE IMPLEMENTATION AND RESULTS

The above-described method of the generation of molecular graphs was implemented as a program written in Java. The generator can be run on a wide range of platforms, including Windows, Mac OS, Linux, IRIX, etc. The program has an easy-to-use interface (Figure 16). The structural formulas of generated structures can be depicted (Figure 16d). The program includes an LNFT-to-SDF file converter, which exports generation results to the SDF format.[29] Therefore the generator is compatible with most of the programs for chemical studies.

Let us consider several examples of the generation tasks and their results. To process these tasks we use a PC (Intel Pentium 4, 2.4 GHz). The first example (Figure 17) is a typical generation task for QSAR studies. In the deterministic generation mode the generator produces 211 317 unique structures (another 43 999 structures are rejected) within 3.68 s, and the generation of 10 000 structures in the stochastic mode takes 0.21 s. The second example (Figure 18) illustrates the influence of all the above-mentioned factors leading to the generation of identical structures. For the second task our generator stores 196 structures and rejects 1100 structures within 0.56 s. Moreover we ran several simple generation tasks (Figure 19), which demonstrated the duplicate recognition for the typical cases.

The generator can be used for the constructive enumeration of structures corresponding to a given Markush formula and

Table 3. Results for the Variations of the Generation Task in Figure 17

test no.	substitution positions	additional terminal fragment	H_{max}	reject duplicates	no. of saved structures (N_G)	no. of rejected duplicate structures (N_D)	total duration of the task processing (T), s	duration of the generation step (T_G), s	efficient generation rate (N_G/T), [structures per s]	real generation rate ($[N_G + N_D]/T_G$), [structures per s]
1	R1−R5		2	Y	10 759	1355	0.17	0.12	63 288	100 950
2	R1−R5		3	Y	211 317	43 999	3.68	3.07	57 423	83 164
3	R1−R5		4	Y	454 752[a]	97 221	30.36	9.20	14 978	59 997
4	R1−R5	C_2H_5	3	Y	1 464 921	2 688 890	55.35	53.86	26 466	77 122
5	R1−R4		3	Y	135 675	45 578	2.25	1.88	60 300	96 411
6	R1−R3		3	Y	24 673	13 293	0.53	0.28	46 552	135 592
7	R1−R4		4	Y	488 979[a]	162 274	25.41	9.64	19 243	67 557
8	R1−R3		4	Y	196 096	111 003	11.69	3.7	16 774	82 999
9	R1−R5		2	N	12 114	0	0.13	0.11	93 184	110 127
10	R1−R5		3	N	255 316	0	2.91	2.89	87 737	88 344
11	R1−R5		4	N	551 973	0	8.55	8.52	64 558	64 785

[a] The number of structures generated in test 7 is greater than the number of structures generated in test 3 due to the influence of the T_{max} constraint.

for the generation of virtual combinatorial libraries. We reproduced the examples from refs 15 and 30 (Figure 20) and compared the results (Table 2).

To investigate the influence of complexity of the generation task (the number of substitution sites, the number of elementary fragments, H_{max}, and the use of duplicate recognition procedure) on the generation rate we ran several generation tasks based on the task shown in Figure 17. Some results of these tests are given in Table 3.

As evident from Table 3 the duration of the task processing noticeably increases as H_{max} grows if the duplicate recognition is switched on. Mainly this is caused by an increase in the time required for the analysis of the molecular supergraph at the preliminary step. When H_{max} is increased by one unit, the molecular supergraph becomes significantly bigger. Work on the optimization of the algorithm of molecular supergraph analysis is in progress. To evaluate efficiency of the structure generation algorithm and the duplicate recognition algorithm we measured the duration of the generation step and the real generation rate. A decrease in the real generation rate with the growth of H_{max} is not so noticeable. If the duplicate recognition is switched off, then writing the structures into a file becomes the main time-consuming operation. If one does not save structures on the disk and the duplicate recognition is switched off, then the generation rate exceeds 1 million structures per second. In practice H_{max} is usually equal to 2 or 3, and most of the really useful generation tasks are processed in a few seconds.

Our generator software has no explicitly defined limitation on the number of substitution positions, the number of elementary fragments, etc. But the user is advised to apply structural constraints to restrict the number of generated structures and to prevent the generation of structures that are out of the applicability range of the QSAR model under study.

CONCLUSION

In this paper we proposed a new approach for the generation of substituted derivatives of a given structure. The described generator efficiently handles various types of constraints and supports the stochastic generation mode as well as the deterministic one. The use of the fragment supergraph allowed us to implement the algorithm of the exhaustive generation, which supports various structural constraints, while the use of the molecular supergraph allows us to analyze conditions for the generation of duplicate structures and to use the results of such analysis to cut off all the duplicates. The important distinction of the proposed approach is that the generator completely rejects isomorphic molecular graphs without the comparison with the previously generated structures, while the internal structure of the molecular fragments is considered only at the preliminary step of the generation. In most cases it gives a rather high generation rate and provides at the same time the exhaustive and irredundant generation that can be considered as a serious advantage of the proposed method. Finally all the above-mentioned features of the generator make it a very useful tool for the design of novel biologically active compounds.

ACKNOWLEDGMENT

We thank Dr. Eugene V. Radchenko and Dr. Alexander A. Oliferenko for fruitful discussions.

APPENDIX: THE DUPLICATE RECOGNITION ALGORITHM

Let $L_a = (a_1, a_2, ... a_n)$ be an LNFT of the current structure and $L_b = (b_1, b_2, ... b_m)$ be an LNFT of an earlier generated structure such that these structures have isomorphic molecular graphs and L_b is lexicographically smaller than L_a. Then there exists an integer i such that $a_i > b_i$ and $a_j = b_j$ if $j < i$.

Every element of an LNFT corresponds to a vertex of the fragment supergraph, and every vertex f_k of the fragment supergraph corresponds to a subgraph F_k of the molecular supergraph. Let G_{k1}, G_{k2}, ... G_{km} be subgraphs of the molecular supergraph, which are isomorphic to F_k, and $\{v_{k1}, v_{k2}, ..., v_{kp}\}$ be a set of vertices of F_k. A bijection of F_k onto G_{ki} is called a mapping $p_i = (\{v_{k1}, v_{k2}, ..., v_{kp}\} \rightarrow \{v_{t1}, v_{t2}, ..., v_{tp}\})$. Elements of $\{v_{k1}, v_{k2}, ..., v_{kp}\}$ are called source vertices of the mapping, and elements of $\{v_{t1}, v_{t2}, ..., v_{tp}\}$ are called target vertices. Target vertices of a mapping can be related to different vertices of the fragment supergraph. These vertices of the fragment supergraph are listed in array *fragments* = $\{f_{r1}, f_{r2}, ... f_{rm}\}$ that will be denoted below as *p.fragments*, where p is the mapping under consideration. If the sets of target vertices of two mappings p_1 and p_2 are intersecting, then we say that these mappings are conflicting.

Let f_k be a vertex of the fragment supergraph and f_p be the parent of f_k. The mapping $pp = (\{v_{p1}, v_{p2}, ..., v_{pm}\} \rightarrow \{v_{t1}, v_{t2}, ..., v_{tm}\})$ related to f_p is called the parent mapping of the mapping $p = (\{v_{k1}, v_{k2}, ..., v_{kn}\} \rightarrow \{v_{s1}, v_{s2}, ..., v_{sn}\})$ related to f_k if the following conditions are satisfied: (1) v_{ki} is adjacent to v_{pj}; (2) pp maps v_{pj} into v_{tj}; (3) v_{si} is adjacent to v_{tj}; and (4) $\{v_{s1}, v_{s2}, ..., v_{sn}\} \cap \{v_{t1}, v_{t2}, ..., v_{tm}\} = \varnothing$.

Evidently, a mapping can have several parent mappings. However in this case several copies of the mapping are formed, and each mapping is linked to one parent mapping. The parent mapping of the mapping p is denoted as *p.parent*.

A mapping p can have one or more inverse mappings. Each inverse mapping corresponds to a vertex listed in *p.fragments*; therefore, the number of the inverse mappings is equal to the number of elements in *p.fragments*. The sum of the inverse mappings "reverses" the action of p. Some mappings have no inverse mappings, but such mappings are ignored in the context of the duplicate recognition algorithm. The array of the inverse mappings of the mapping p is denoted as *p.inverseMapping*.

The duplicate recognition algorithm is implemented in six procedures. Each of these procedures returns *true* or *false* values. If the procedure returns *false*, then the current LNFT is a duplicate. If Procedure1 returns *true*, then the LNFT is not a duplicate. If any other procedure returns *true*, then it means that one continues the analysis. Let us consider these procedures in details and present their Java-like pseudocode.

The first procedure tries to find the numerical position i ($1 \leq i < n$, i is a zero-based index) from which the elements of LNFTs L_a and L_b are not equal. In this procedure the cycle is started from 1 because the zero element of every LNFT corresponds to the central fragment which is the common fragment for all generated structures.

```
Procedure1(){
    n = number of elements in La;
    for(i = 1; i < n; i++){
        if(!Procedure2(i))
            return false;
    }
    return true;
}
```

For given i the second procedure tries to find b_i such that $a_i > b_i$ where a_i is the ith element of L_a and b_i is the ith element of L_b. Let f_a be a vertex of the fragment supergraph corresponding to a_i. This vertex can be determined by the generation stack. *Procedure2* consequently tests the vertices of the fragment supergraph that precede f_a in its group competitive vertices. We test these vertices because the competitive vertices are ordered by the fragment numbers in the ascending order.

```
Procedure2(i){
    $f_a$ = fragment supergraph vertex corresponding to i-th element of $L_a$;
    $C_a$ = group of competitive vertices so that $f_a$ is a member of the group;
    k = zero-based index of $f_a$ in group $C_a$;
    for(j = 0; j < k; j++){
        $f_b$ = j-th item of $C_a$;
        if(!Procedure3(i, $f_b$))
            return false;
    }
    return true;
}
```

Procedure3 tests the mappings related to f_b. There is no reason to consider the identity mappings because such mappings have no target vertices belonging to the current structure.

```
Procedure3(i, $f_b$){
    n = number of mappings of $f_b$;
    for(j = 0; j < n; j++){
        p = j-th mapping of $f_b$;
        if(p is an identity mapping)
            continue;
        if(!Procedure4(i, p, $f_b$))
            return false;
    }
    return true;
}
```

Let us consider the auxiliary procedure *TestMapping*. It returns *false* if all the target vertices of the given mapping belong to the current structure. Otherwise the procedure returns *true*.

```
TestMapping(p){
    n = number of elements in p.fragments;
    for(j = 0; j < n; j++){
        $f_m$ = p.fragments[j];
        if($f_m$ is not a member of structure defined by $L_a$)
            return true;
    }
    return false;
}
```

Procedure4 tests the mapping p using the procedure *TestMapping*. If *TestMapping* returns *false*, then p is added to the list *BtoA*, the inverse mappings of p are added to the list *AtoB*, and f_b is added to the list *bFragments*. Let f_m be the fragment supergraph's vertex belonging to the current structure. If f_m is equal to *p.fragments[j]*, then we put the jth inverse mapping of p into the mth element of the array *aMappings*. Then *Procedure4* calls *Procedure5* if p is a mapping of the central fragment otherwise *Procedure4* calls itself a test *p.parent*. We restore the initial state of *aMappings*, *AtoB*, *BtoA*, and *bFragments* before *Procedure4* returns a result value. This is necessary for the correct execution of the duplicate recognition procedures, because these data structures are common in all the procedures and they are initialized at the preliminary step of generation.

```
Procedure4(i, p, $f_b$){
    if(TestMapping(p))
        return true;
    n = number of elements in p.fragments;
    BtoA.add(p);
    bFragments.add($f_b$);
    for(int j = 0; j < n; j++){
        ip = p.inverseMapping[j];
        AtoB.add(ip);
        $f_m$ = p.fragments[j];
        aMappings[index of $f_m$ in $L_a$] = ip;
    }
    if(p.parent == null)
        result = Procedure5(i, 1);
    else
        result = Procedure4(i, p.parent, $f_b$.parent);
    BtoA.remove(p);
    bFragments.remove($f_b$);
    for(int j = 0; j < n; j++){
        ip = p.inverseMapping[j];
        AtoB.remove(ip);
        $f_m$ = p.fragments[j];
        aMappings[index of $f_m$ in $L_a$] = null;
    }
    return result;
}
```

Procedure5 tests elements of L_b having a numerical position less than i. If the position is related to a vertex from the list *bFragments*, then we can skip testing this position because the same test has already been performed in *Procedure 4*.

```
procedure5(i, j){
    if(j == i){
        //the index of the central fragment is 0
        //thus procedure6 is started from i=1
        return Procedure6(1);
    }
    //if j<i then the j-th element of $L_a$ is the j-th element of $L_b$
    $f_b$ = fragment supergraph vertex corresponding to j-th element of $L_a$;
    if($f_b$ is in bFragments)
        return Procedure5(i, j+1);
    n = number of mappings of $f_b$;
    for(k = 0; k < n; k++){
        p = k-th mapping of $f_b$;
        if(p.parent is not in BtoA)
            continue;
        if(TestMapping(p))
            continue;
        if(p conflicts with a mapping from BtoA)
            continue;
        BtoA.add(p);
        m = number of elements in p.fragments;
        for(r = 0; r < m; r++){
            ip = p.inverseMapping[r];
            AtoB.add(ip);
            $f_m$ = p.fragments[r];
            aMappings[index of $f_m$ in $L_a$] = ip;
        }
        result = Procedure5(i, j+1);
        BtoA.remove(p);
        for(int r = 0; r < n; r++){
            ip = p.inverseMapping[r];
            AtoB.remove(ip);
            $f_m$ = p.fragments[r];
            aMappings[index of $f_m$ in $L_a$] = null;
        }
        if(!result)
            return false;
    }
    return true;
}
```

At the moment of the first call of *Procedure6* the first *i* elements of L_b are already known. Moreover mappings for some elements of L_a are stored in the array *aMappings*, and this procedure tries to find mappings for all the rest. If the procedure finds the required mapping for each element of L_a, then the current structure is a duplicate.

```
Procedure6(int i){
    if(i == number of elements in $L_a$)
        return false; //$L_a$ is a duplicate
    if(aMappings[i] != null)
        return Procedure6(i+1);
    $f_a$ = fragment supergraph vertex corresponding to i-th element of $L_a$;
    $f_p$ = parent of $f_a$;
    pp = aMappings[index of $f_p$ in $L_a$];
    n = number of mappings of $f_a$;
L1: for(j = 0; j < n; j++){
        p = j-th mapping of $f_a$;
        if(p.parent != pp)
            continue;
        m = number of elements in p.inverseMapping;
        for(k = 0; k < m; k++){
            ip = p.inverseMapping[k];
            if(TestMapping(ip))
                continue L1;
        }
        if(p conflicts with mappings from list AtoB)
            continue;
        AtoB.add(p);
        result = Procedure6(i+1);
        AtoB.remove(p);
        if(!result)
            return false;
    }
    return true;
}
```

REFERENCES AND NOTES

(1) Lindsay, R. K.; Buchanan, B. G.; Feigenbaum, E. A.; Lederberg, J. *Applications of artificial intelligence for organic chemistry: The Dendral Project;* McGraw-Hill: New York, 1980.

(2) Kudo, Y.; Sasaki, S. I. Principle for Exhaustive Enumeration of Unique Structures Consistent with Structural Information. *J. Chem. Inf. Comput. Sci.* **1976**, *16*, 43−49.

(3) Shelley, C. A.; Hays, T. R.; Munk, M. E.; Roman, R. V. An Approach to Automated Partial Structure Expansion. *Anal. Chim. Acta* **1978**, *103*, 121−132.

(4) Benecke, C.; Grund, R.; Hohberger, R.; Kerber, A.; Laue, R.; Wieland, T. MOLGEN+, a generator of connectivity isomers and stereoisomers for molecular structure elucidation. *Anal. Chim. Acta* **1995**, *314*, 141−147.

(5) Bangov, I. P. Computer-assisted structure generation from a gross formula. 4. Fighting against graph-isomorphism disease. *MATCH Commun. Math. Chem.* **1992**, *27*, 3−30.

(6) Bohanec, S.; Zupan, J. Structure Generator GEN. *MATCH Commun. Math. Chem.* **1992**, *27*, 49−85.

(7) Faulon, J. L. Stochastic Generator of Chemical Structure. 1. Application to the Structure Elucidation of Large Molecules. *J. Chem. Inf. Comput. Sci.* **1994**, *34* (5), 1204−1218.

(8) Molchanova, M. S.; Shcherbukhin, V. V.; Zefirov, N. S. Computer Generation of Molecular Structures by the SMOG Program. *J. Chem. Inf. Comput. Sci.* **1996**, *36* (4), 888−899.

(9) Molodtsov, S. G. The generation of molecular graphs with obligatory, forbidden and desirable fragments. *MATCH Commun. Math. Comput. Chem.* **1998**, *37*, 157−162.

(10) Luinge, H. J. AEGIS, a Structure Generation Program in Prolog. *MATCH Commun. Math. Chem.* **1992**, *27*, 175−189.

(11) *SYBYL*, *version 7.0*; Tripos Inc.: St. Louis, MO, 2006.

(12) Anderson, A. C. The Process of Structure-Based Drug Design. *Chem. Biol.* **2003**, *10*, 787−797.

(13) Schüller, A.; Schneider, G.; Byvatov, E. SMILIB: Rapid Assembly of Combinatorial Libraries in SMILES Notation. *QSAR Comb. Sci.* **2003**, *22*, 719−721.

(14) Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31−36.

(15) Gugisch, R.; Kerber, A.; Laue, R.; Meringer, M.; Weidinger, J. MOLGEN-COMB, a software package for combinatorial chemistry. *MATCH Commun. Math. Comput. Chem.* **2000**, *41*, 189−203.

(16) *Cerius^2*, *version 4.10*; Accelrys, Inc.: San Diego, CA, 2005.

(17) *MOE*, *version 2006.08*; Chemical Computing Group Inc.: Montreal, Canada, 2006.

(18) Douguet, D.; Thoreau, E.; Grassy, G. A genetic algorithm for the automated generation of small organic molecules: Drug design using an evolutionary algorithm. *J. Comput.-Aided Mol. Des.* **2000**, *14*, 449−466.

(19) Pegg, S.; Haresco, J.; Kuntz, I. A genetic algorithm for structure-based de novo design. *J. Comput.-Aided Mol. Des.* **2001**, *15*, 911−933.

(20) Jamois, E.; Lin, C.; Waldman, M. Design of focused and restrained subsets from extremely large virtual libraries. *J. Mol. Graphics Modell.* **2003**, *22*, 141−149.

(21) Gillet, V.; Willett, P.; Fleming, P.; Green, D. Designing focused libraries using MoSELECT. *J. Mol. Graphics Modell.* **2002**, *20*, 491−498.

(22) Kubinyi, H. *QSAR: Hansch Analysis and Related Approaches;* VCH: Weinheim, 1993.

(23) Netzeva, T. I.; Worth, A. P.; Aldenberg, T.; Benigni, R.; Cronin, M. T. D.; Gramatica, P.; Jaworska, J. S.; Kahn, S.; Klopman, G.; Marchant, C. A.; Myatt, G.; Nikolova-Jeliazkova, N.; Patlewicz, G. Y.; Perkins, R.; Roberts, D. W.; Schultz, T. W.; Stanton, D. T.; van de Sandt, J. J. M.; Tong, W.; Veith, G.; Yang, C. ECVAM WORKSHOP REPORT Current status of methods for defining the applicability domain of (quantitative) structure-activity relationships. *ATLA* **2005**, *33*, 155−173.

(24) Lomova, O. A.; Sukhachev, D. V.; Kumskov, M. I.; Palyulin, V. A.; Tratch, S. S.; Zefirov, N. S. The Generation of Molecular Graphs for QSAR Studies by the Acyclic Fragment Combining. *MATCH Commun. Math. Chem.* **1992**, *27*, 153−174.

(25) Tratch, S. S.; Lomova, O. A.; Sukhachev, D. V.; Palyulin, V. A.; Zefirov, N. S. Generation of Molecular Graphs for QSAR Studies: An Approach Based on Acyclic Fragment Combinations. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 130−139.

(26) Smith, E. G. *The Wiswesser Line-Formula Chemical Notation*; McGraw-Hill: New York, 1968.

(27) Mel'nikov, A. A.; Palyulin, V. A.; Zefirov, N. S. Generation of Molecular Graphs for QSAR Studies. *Dokl. Chem.* **2005**, *402*, 81−85.

(28) Palyulin V. A.; Melnikov, A. A.; Zefirov, N. S. An Approach to the Generation of Molecular Graphs for QSAR Studies Based on Supergraphs. *Book of Abstracts of 2 German conferences on chemoinformatics*, November 12−14, 2006, Goslar, Germany, p 55.

(29) *MDL CTfile Formats*. http://mdli.com/solutions/white_papers/ctfile-_formats.jsp (accessed May 5, 2007).

(30) Kerber, A.; Laue, R.; Meringer, M. An Application of the Structure Generator MOLGEN to Patents in Chemistry. *MATCH Commun. Math. Comput. Chem.* **2003**, *47*, 169−172.

CI700156F