MAYGEN - an open-source chemical structure generator for constitutional isomers based on the orderly generation principle

Mehmet Aziz Yirik*; University Friedrich-Schiller, Lessing Strasse 8, 07743, Jena, Germany; yirik.mehmetaziz@uni-jena.de; ORCID: 0000-0001-7520-7215
Maria Sorokina; University Friedrich-Schiller, Lessing Strasse 8, 07743, Jena, Germany; maria.sorokina@uni-jena.de; ORCID: 0000-0001-9359-7149
Christoph Steinbeck*; University Friedrich-Schiller, Lessing Strasse 8, 07743, Jena, Germany; christoph.steinbeck@uni-jena.de; ORCID: 0000-0001-6966-0814

Corresponding authors emails: christoph.steinbeck@uni-jena.de, mehmetazizyirik@gmail.com

Abstract

The generation of constitutional isomer chemical spaces has been a subject of cheminformatics since the early 1960s, with applications in structure elucidation and elsewhere. In order to perform such a generation efficiently, exhaustively and isomorphism-free, the structure generator needs to ensure the building of canonical graphs already during the generation step and not by subsequent filtering.
Here we present MAYGEN, an open-source, pure-Java development of a constitutional isomer molecular generator. The principles of MAYGEN's architecture and algorithm are outlined and the software is benchmarked against the state-of-the-art, but closed-source solution MOLGEN, as well as against the best open-source solution OMG. MAYGEN outperforms OMG by an order of magnitude and gets close to and occasionally outperforms MOLGEN in performance.

Keywords: constitutional isomer generation, algorithmic group theory, algorithmic graph theory, chemical graph generation, open-source software, CDK

1. Introduction

The efficient generation of constitutional isomers of a given molecular formula has been a core area of cheminformatics research for decades [1]. Such molecular generation methods can be used as hypothesis generators in areas such as computer-assisted structure elucidation, but also to answer broader questions such as the exact size of a chemical space. Structure generators that produce constitutional isomers take a molecular formula as input, e.g., $C_{10}H_{16}O$, and enumerate or output all possible chemical structures that can be built with the given set of atoms in the molecular formula. The history of chemical graph generators reaches back to the 1960s and has recently been reviewed in detail [1].

Despite the long history of research on the theoretical and practical generation of chemical graphs, the number of publicly available algorithms and software for this purpose is still limited. For several decades, the closed-source, commercial structure generator MOLGEN, developed at the University of Bayreuth, marks the state of the art in terms of speed and completeness. Recognising the need for an open-source structure generator, Peironcely et al. [2] developed the Open Molecule Generator (OMG). OMG, however, is orders of magnitude slower than MOLGEN. The 452,458 isomers of $C_{10}H_{16}O$, for instance, are generated in only 5 seconds by MOLGEN, whereas OMG takes 22 minutes on the same machine (a 2020 Macbook Pro, 2,3 GHz 8-Core Intel Core i9). For more benchmarks, please see the results section of the present manuscript.

In this work, we present the development of an open-source structure generator MAYGEN, a pure-Java constitutional isomer generator based on the principle of orderly generation described by Grund et al [3]. We benchmark our method against the fastest available open-source solution OMG [2] as well as against the closed-source, *de facto* gold standard MOLGEN. MAYGEN outperforms OMG by an order of magnitude and above but is still outperformed by MOLGEN. In an old Arabic saying, "may" refers to a drop of water, and we hope that MAYGEN will be a good contribution to the field and trigger a surge in the development of improved and faster versions eventually rivalling the best closed-source solutions and thereby serving the scientific community. The complete MAYGEN code, as well as precompiled binaries, are available at https://github.com/MehmetAzizYirik/MAYGEN.

2. Methods

MAYGEN generates constitutional isomers of a given molecular formula with an orderly graph generation algorithm from the field of algorithmic group theory. The principles are described in detail in [3]. We summarize them in the following. A graph with *p* nodes, $\{1, 2, 3, \ldots, p\}$ has its symmetry group S_p. This symmetry group includes all the permutations of these nodes. However, for the case of coloured graphs, the nodes need to be partitioned (Equation 1), in other words, nodes are grouped based on their colours, degrees and edges.

$$\lambda := (\lambda_1, \lambda_2, \ldots) \; with \sum_i \lambda_i = n_i \quad (1)$$

A molecule can be represented as a coloured graph. For 4 isomers of $C_8O_2H_{16}$ (Figure 1), all atoms are coloured by their element types.

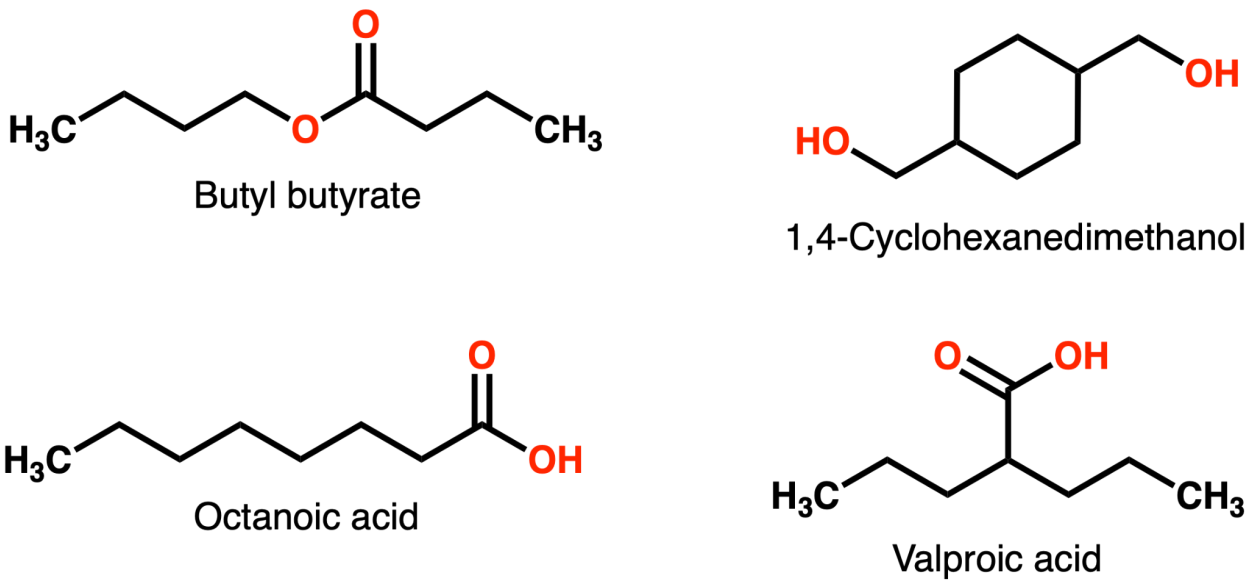

Figure 1. Four Isomers of $C_8O_2H_{16}$. Atoms are coloured by their type.

The atoms of $C_8O_2H_{16}$ can be partitioned in three groups as following: $\lambda = \{1, 2 \mid 3, 4, 5, 6, 7, 8, 9, 10 \mid 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26\}$. For the case of this node partition, the symmetry group of 26 nodes, S_{26}, cannot be used since the nodes are coloured. In this case, a special type of symmetry group is applied, consisting of Young subgroups, that are the symmetry groups built based on the initial node partition (Equation 2 and 3).

$$n = \bigcup_i n_i^{\lambda} \; where \; n_i^{\lambda} := \left\{ \sum_{j=1}^{i-1} \lambda_j + 1, \ldots, \sum_{j=1}^{i} \lambda j \right\} \quad (2)$$

$$S_{\lambda} := \left\{ \pi \in S_n \mid \forall i : \pi(n_i^{\lambda}) = n_i^{\lambda} \right\} \leq S_n \quad (3)$$

This symmetry group S_{λ} is the direct product of Young subgroups permuting each atom type within its partition. In the case of $C_8O_2H_{16}$, the symmetry group of λ is $S_{\{1, 2\}} * S_{\{3, 4, 5, 6, 7, 8, 9, 10\}} * S_{\{11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26\}}$. The permutations of

these symmetry groups only permute each element type within their groups, such as oxygens, carbons and hydrogens. The Young subgroups are then used for the construction of molecules' automorphism groups (Equation 4). These atom partitions and symmetry groups are the core part of the MAYGEN canonical test.

$$Aut(A) := \left\{\pi \in S_n \mid A\pi = A\right\} \leq S_p \quad (4)$$

MAYGEN's construction of candidate structures consists of three distinct recursive tasks. First, the hydrogens are distributed to the heavy (i.e. non-hydrogen) atoms of the molecular formula. Then, the structures are generated in a block-wise manner, and finally, the canonical test avoids the generation of duplicate structures in an efficient and dynamic manner.

2.1. Molecular formula check and hydrogen distribution

2.1.1. Graph Existence Check

Before calling the generator functions, there is a preliminary test for input molecular formulae. From graph theory, a degree list *d* can represent a graph with *p* nodes if the sum of all node degrees is equal or greater than $2 * (p - 1)$ (Equation 5) [3].

$$d = (d_1, d_2, \ldots, d_p) \quad \sum_{i=1}^{p} d_i \geq 2 * (p - 1) \qquad (5)$$

A graph with *p* nodes should consist of at least $(p - 1)$ edges. Since an edge is connected with two nodes in a graph, the sum of its node degrees should be equal to or greater than $2 * (p - 1)$.

2.1.2. Hydrogen Distribution

For a given molecular formula, MAYGEN processes the hydrogens first and distributes them to all the other atoms in all possible ways since a hydrogen atom has a valence of 1 and can always have only one neighbour. The hydrogen distribution function takes two inputs, the atom partition and the number of hydrogens. The hydrogens are distributed in ascending order within each partition in order to avoid duplicates.

After the hydrogen distribution, the initial degrees and the initial partition are updated for each hydrogen distribution. For example, the non-hydrogen atoms from the molecular formula C_6H_6 have the initial respective degrees as $[4, 4, 4, 4, 4, 4]$ and the initial partition $\{6\}$. There are 7 possible hydrogen distributions (Figure 2) to these carbon atoms. After the hydrogen distribution step, the new lists of node degrees and partitions are used for the structure generation process. With the pre-hydrogen distribution, MAYGEN deals with a $6\,x\,6$ matrix instead of a $12\,x\,12$ matrix. The matrix size also has an impact on the canonical test since this test depends directly

on the rows' permutations. The hydrogen distribution code is available in the hydrogenDistributor Java class.

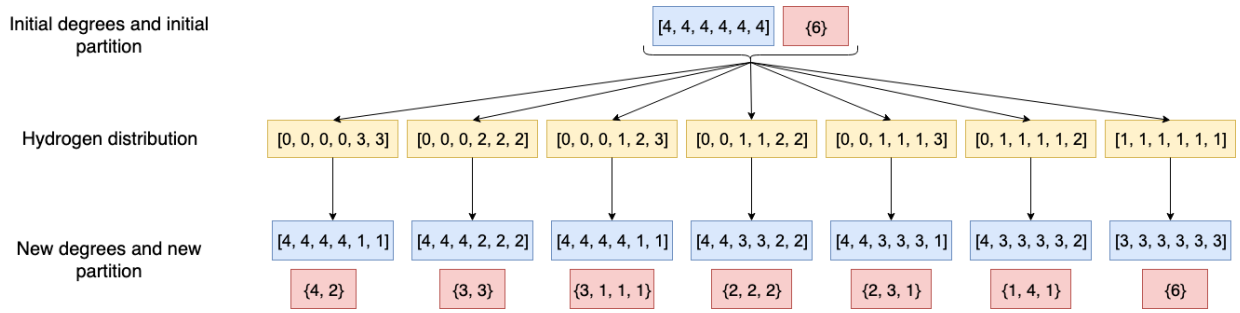

Figure 2. Illustration of the hydrogen distribution of C_6H_6 (in yellow) and its effect on the assigned atom valency (in blue) and on the atom partition (in red).

2.2. Construction of Candidate Structures

Once the molecular formula satisfies the graph existence criteria, the hydrogen distribution is performed to build a list of degrees. MAYGEN then starts the actual construction of candidate structures for each degree.

The structures are built in a block-wise manner. The algorithm is based on the node degrees that correspond to the atom valences. The initial partition of the atoms, based on their element symbols, defines the blocks of the matrix (Figure 3).

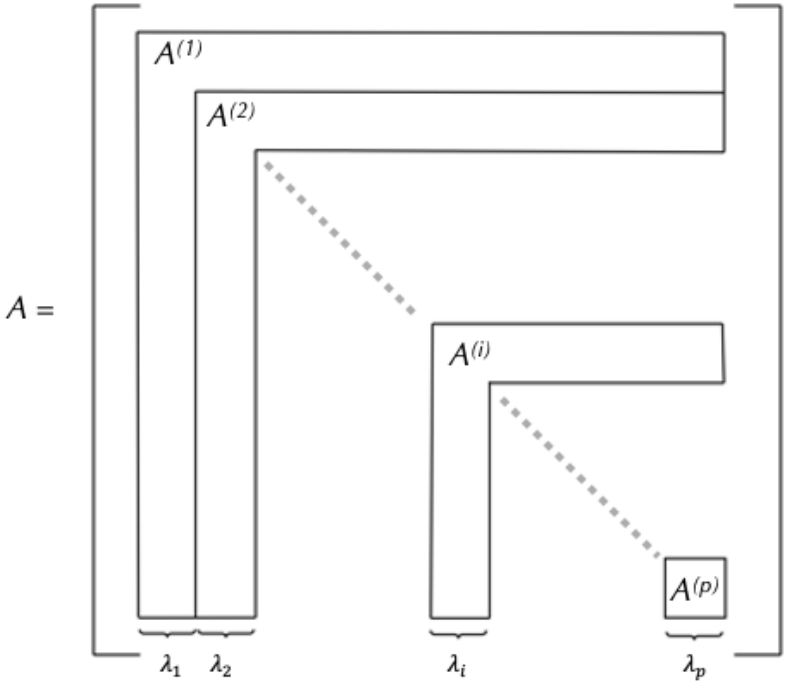

Figure 3. Block-wise representation of a matrix. Here, the matrix is split into parts based on the initial node partition λ with t entries. Image adapted from [3].

With *p* being the number of atoms in the molecular formula without the hydrogens, an empty $p\,x\,p$ matrix A is built. This matrix is filled in descending order starting with the maximal capacities (e.g. 3 for a carbon atom), decrementing its valence (e.g. 4 for the same carbon atom), and this is performed for each atom. Due to the diagonal symmetry of such matrices, only the upper triangular part needs to be filled. A canonical test, as described below, is performed once a block is filled. In a matrix, a block is defined as a number of rows and their transposes (i.e. columns). For example, a block between two indices 1 and 4 means the first 4 rows and the first 4 columns of the matrix. It needs to be noted that the canonical texts are performed without waiting for the whole matrix to be filled, which increases MAYGEN's efficiency. This is the early boundary condition of the blockwise generation and avoids the construction of duplicate molecular structures. When the whole matrix is filled, it is written into the output SDF file, if such an option is selected at the beginning of the process. The algorithm then modifies the same input matrix A until there are no more possible changes. This is called the *"build-and-forget method"* [3]. The overall algorithm structure is explained in Algorithm 1 [3].

Algorithm 1: MAYGEN algorithm

Input: Molecular formula with p non-hydrogen atoms
Output: SDF file with molecular structures

Step 1: Perform hydrogen distribution

Step 2: Set block index i

Step 3: First the block index i is set, $i = 1$; go to step 5.

Step 4: **if** $i = 0$ **then** the procedure stops **else** go to step 6

Step 5: **Maximum filling**
Fill the strip $A(i)$ in lexicographic order depending on the valences.
if no more fillings exist **then**

1. set $i = (i - 1)$
2. go to step 4

else go to step 7

Step 6: **Smaller filling**
Fill the strip $A(i)$ in a reverse lexicographic order depending on the valences.
if no more fillings exist **then**

1. set $i = (i - 1)$
2. go to step 4

else go to step 7

Step 7: **Canonical Test**
if $A(i) \geqslant A(i)\pi$ for all $\pi \in Aut(A)$ **then** $A(i)$ is canonical

1. **if** $i = p$ **then**
   (a) canonical matrix is complete
   (b) store in output SDF file
   (c) go to step 6
2. **else**
   (a) update $Aut(A)$
   (b) set $i = (i + 1)$
   (c) go to step 5

else go to step 6

Keeping the example of $C_6O_2H_6$, the initial valence vector is $v = [4, 4, 4, 4, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1]$, where the valences of each carbon atom are listed first, then the valences of each oxygen atom, and lastly the valences of all 6 hydrogen atoms. To optimize the process, the hydrogens are avoided in the further construction of the matrices, i.e

the hydrogen distribution step. Thus, the initial partition is $\lambda = \{6, 2\}$ and the corresponding matrix is a $8\ x\ 8$ matrix (built on 6 carbons and 2 oxygens).

2.2. Canonical Test

The canonical test is the crucial part of the MAYGEN algorithm. In blockwise orderly structure generation, the early canonical testing avoids the construction of many duplicates. Overall, the purpose of the canonical test is the detection of the maximal matrix with respect to the given initial node partition.

$$A \geq A\pi \ \forall \pi \in S_\lambda \quad (6)$$

In the naive version of the canonical, test matrix A is permuted for all the permutations of S_π and its maximality is checked (Equation 6). In the permuted matrices, $A\pi$, their rows and entries are permuted. The original matrix A is compared with all the permuted matrices. Two matrices are compared row by row in a lexicographical order (Equation 7).

$$A > A' :\Leftrightarrow (a_{1,1},\ldots, a_{1,p}, a_{2,1},\ldots, a_{2,p},\ldots, a_{p,1},\ldots, a_{p,p}) > (a'_{1,1},\ldots, a'_{1,p}, a'_{2,1},\ldots, a'_{2,p},\ldots, a'_{p,1},\ldots, a'_{p,p}) \quad (7)$$

In the blockwise orderly generation, only the rows within the blocks are compared.

2.2.1. Cycle Transpositions

In the canonical test, the size of the symmetry group affects the run time of the algorithm. The initial partition is updated for each row during the test. Starting with the initial partition, with each row, the partitions are refined. The refinement process (Equation 8) is explained below:

$$\lambda^{(i)} = \begin{cases} (\underbrace{1,\ldots,1}_{\text{i-1}}, 1, \lambda_i^{(i-1)} - 1, \lambda_{i+1}^{(i-1)}, \ldots) & \text{falls} \quad \lambda_i^{i-1} > 1 \\ (\underbrace{1,\ldots,1}_{\text{i-1}}, 1, \lambda_{i+1}^{(i-1)} & \text{falls} \quad \lambda_i^{(i-1)} = 1 \end{cases} \quad (8)$$

For $C_3O_2H_4$, the initial partition without hydrogens is $\{3, 2\}$. Thus the partition list for all the rows are:

$\lambda^{0} = \{3, 2\}$

$\lambda^{1} = \{1, 2, 2\}$

$\lambda^{2} = \{1, 1, 1, 2\}$

$\lambda^{3} = \{1, 1, 1, 2\}$

$\lambda^{4} = \{1, 1, 1, 1, 1\}$

These partition lists are used for the construction of the symmetry groups. By comparing the indices of two consecutive partitions, the cycle transpositions of symmetry groups are calculated. For partitions $\lambda^{(i-1)}$ and $\lambda^{(i)}$, the number of cycles is the i[th] entry in the former partition $\lambda^{(i-1)}_{i}$ (Equation 9).

$$S_{\lambda^{(i-1)}} = \bigcup_{j=i}^{\lambda^{(i-1)}_{i}} (i,j) S_{\lambda^{(i)}},\ i = 1,\dots,\ p-1. \quad (9)$$

For example, the initial partition is $\{3, 2\}$ and the refined partition for the first row is $\{1, 2, 2\}$. Here the number of cycle transpositions is 3 since the first entry of the former partition is 3. The cycle transpositions are $(1,1)$, $(1,2)$ and $(1,3)$. These cycles are calculated row by row for all the partitions. The symmetry group of the molecule is calculated by the multiplication of all these cycles. The list of the partitions and their cycles are listed below:

$\lambda^{0} = \{3, 2\}$ $\quad \lambda^{1} = \{1, 2, 2\}$ $\quad Cycles$: $(1,1)$, $(1,2)$, $(1,3)$

$\lambda^{1} = \{1, 2, 2\}$ $\quad \lambda^{2} = \{1, 1, 1, 2\}$ $\quad Cycles$: $(2,2)$, $(2,3)$

$\lambda^{2} = \{1, 1, 1, 2\}$ $\quad \lambda^{3} = \{1, 1, 1, 2\}$ $\quad Cycles$: $(3,3)$

$\lambda^{3} = \{1, 1, 1, 2\}$ $\quad \lambda^{4} = \{1, 1, 1, 1, 1\}$ $\quad Cycles$: $(4,4)$, $(4,5)$

2.2.2. Calculation of Automorphisms

In the canonical test, for a candidate matrix, its automorphisms are calculated row by row. For the i[th] row of a matrix, the cycle transpositions $\varsigma_{(i,j)}$ are calculated based on the partitions $\lambda^{(i-1)}$ and $\lambda^{(i)}$. These cycle transpositions are used in the automorphisms search. All these cycles are multiplied in DFS manner with all the former automorphisms τ of the graph. This updated list of permutations are used in the canonical test of the matrix. For a graph with *p* nodes, its list of automorphisms until the i[th] row is:

$$F^{(i)} = \{\tau \in F^{(i-1)} \mid \tau * \varsigma_{(i,j)}\} \quad i < j < \lambda_i^{(i-1)}$$

(10)

After the multiplication with all its cycles (Equation 10), this updated list of automorphisms is used in the maximality check. If an automorphism is detected, that permutation is added to the automorphisms list, F^{i}. Thus, the automorphisms list is updated for each row until the row is in maximal form with respect to its partitions.

2.2.3. Maximality Check

For the maximality test of the i^{th} row of a matrix, the row is compared with each permutation action in the automorphisms list. For each permutation, the original matrix A is permuted. Then, the i^{th} rows of the original matrix and the permuted one are compared. These two rows are compared based on the i^{th} atom partition. For an initial matrix A, as shown in Figure 4a, with its partition $\lambda^{(0)} = \{5\}$ and the refined partition $\lambda^{(0)'} = \{1, 4\}$, there are 5 cycle transpositions. One of these cycles is $(1, 2)$. To perform the maximality test, its first and second rows are compared (Figure 4a).

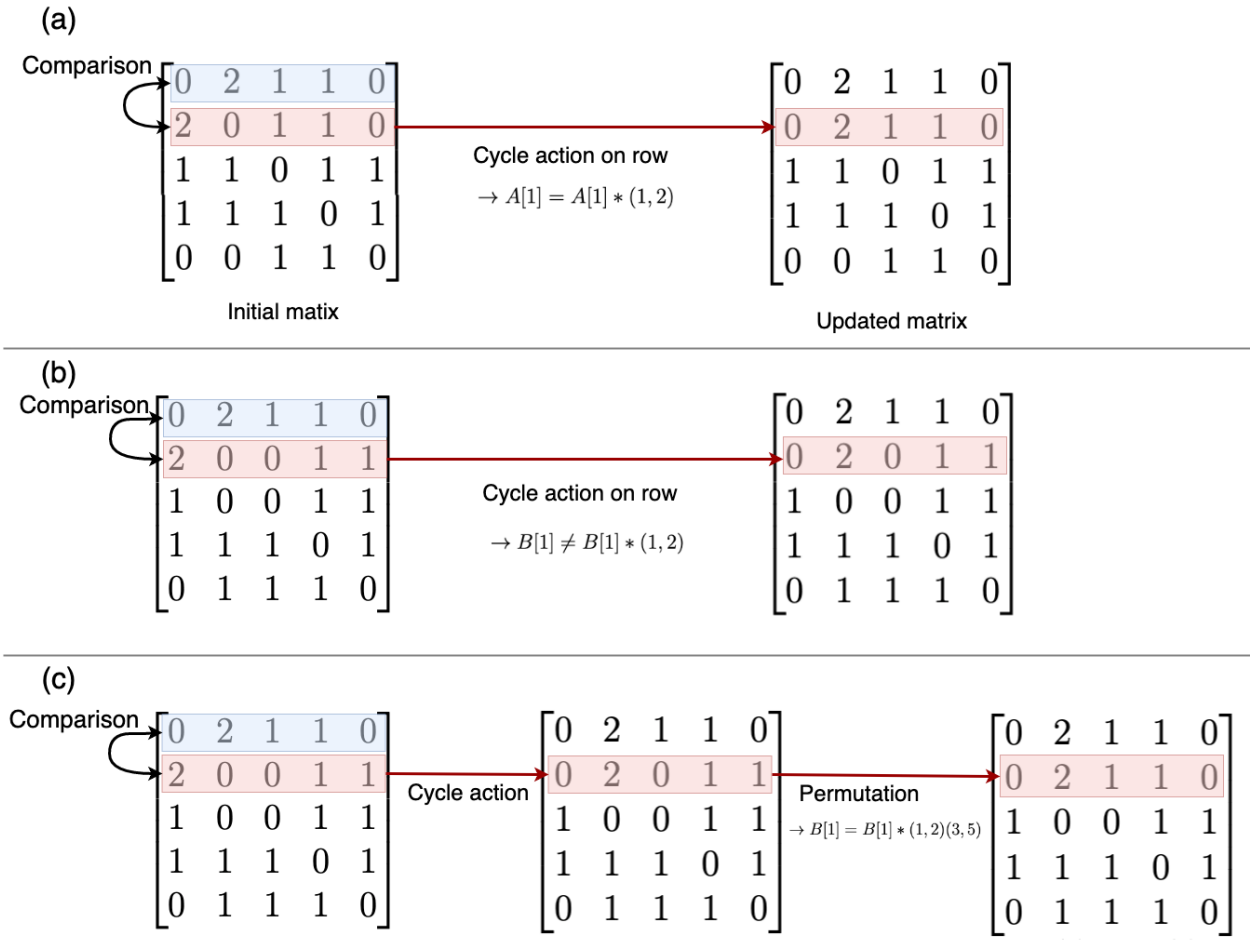

Figure 4. a) A matrix A is permuted with a cycle transposition. The first and the second rows are identical after the permutation action. b) A matrix B is permuted with a cycle transposition. The first and the second rows are not identical. c) The canonical permutation of matrix B is given.

In this example, the permutation $(1, 2)$ is an automorphism of the first row since it maps the row to itself in the adjacency matrix. Then this permutation is added to the automorphisms list. However, in the case where a mapping with a cycle does not map the row to itself, a canonical permutation is needed. Same as matrix A, for an initial matrix B (Figure 4b) with its initial partition $\lambda^{(0)} = \{5\}$, the refined partition $\lambda^{(0)'} = \{1, 4\}$, there are 5 cycle transpositions. One of them is $(1, 2)$. To perform the maximality test, its first and second rows are compared (Figure 4b).

Different from example A, in matrix B, its first and second row are not identical after the cycle transpositions. Therefore, a canonical permutation is needed. The canonical permutations are searched within the Young subgroups built with respect to the refined partition. In this example, the refined partition is $\lambda^{(0)'} = \{1, 4\}$. Thus, the symmetry group is $S_{\{1\}} * S_{\{2,3,4,5\}}$. For the canonical permutation search, only the permutations of the sets $\{1\}$and $\{2, 3, 4, 5\}$are required. For the rows of matrix B, the canonical permutation is then $(3, 5)$, as depicted in Figure 4c. Thus, $(1, 2)(3, 5)$ is the automorphism of the first row and added to the automorphisms list for further testings.

In general, there are three criteria for updating the automorphisms list and for the maximality check:

Procedure: Updating the automorphism list and maximality check

1. If the row i is maximal and equal to the permuted row, the permutation is added to the automorphism list;
2. If the row i is maximal but not equal to the permuted row, an automorphism is searched in its Young subgroup
   (a) If there is such an automorphism, the permutation is added to the automorphisms list;
   (b) Else, the automorphism is ignored and not added to the list.
3. If the original row i, is smaller than the permuted matrix, the tested candidate molecule is not canonical. The canonical test is then terminated.

In the canonical test, if the row is canonical after testing all the permutations, the partition $\lambda^{(i+1)}$ is built based on the ith row's entries. After filling the entries of the ith row, i.e., adding bonds to the ith atom, the atom neighbourhoods are changed. Therefore the partition $\lambda^{(i+1)}$ is defined

based on the partition $\lambda^{(i)}$ and the i[th] row entries. For matrix A and its refined partition $\lambda^{(0)'} = \{1, 4\}$, its partition first is updated with respect to the first row entries (Figure 5).

$$\text{Refined partition } \lambda^{(0)'} = \{1,4\} \rightarrow A[1] = [0|2,1,1,0] \rightarrow \text{Updated partition } \lambda^{(1)} = \{1,1,2,1\}$$

Figure 5. Updating partition after the canonical test with respect to the row entries.

The canonical test continues until the rows are in maximal form in lexicographic order. The automorphisms and partition lists are updated row by row.

2.2.4. Learning From Canonical Test

In case a molecule cannot pass the canonical test, there is still something to learn from the test. In the row by row comparison of the canonical test, when a row does not pass the test, the entry making it non-canonical is detected. As explained in Algorithm 1, if a block is not canonical, MAYGEN updates the matrix starting with its last entry in the block. However, with the help of the non-canonical matrix, the algorithm starts modifying the matrix from the entry making the matrix non-canonical. For a matrix C with its partition $\lambda^{(0)} = \{5\}$ and the refined partition $\lambda^{(0)'} = \{1, 4\}$, there are 5 cycle transpositions. One of these cycles is $(1, 3)$. To perform the maximality test, its first and third rows are compared as shown in Figure 6.

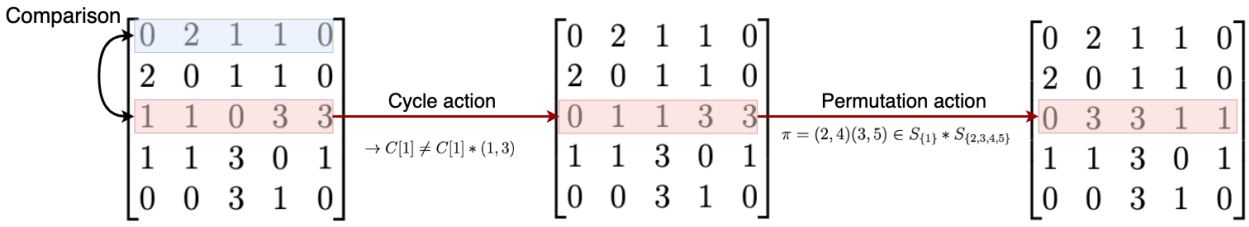

Figure 6. For a non-canonical matrix, detecting the entry indices makes it non-canonical.

The permutation $\pi = (2,4)(3,5) \in S_{\{1\}} * S_{\{2,3,4,5\}}$ makes the third row bigger than the first row. Here the first entry making the row non-canonical is $C[3,4]$ in the matrix. Then the matrix construction continues with the indices $[3,4]$. With the "learning from the canonical test", all the other non-canonical matrices are skipped.

2.3. Connectivity Test

The connectivity test of a graph is performed based on the neighbourhoods of all its nodes. The connectivity test starts with enumerating the nodes and setting this as the initial graph enumeration. The enumeration list is updated while checking the neighbour lists node by node. After detecting neighbours of a node, the labellings of the tested node and its neighbours from

the graph enumeration list are stored. The minimum value of this set is given as the smallest index of the neighbourhood. This smallest index value is used for updating the list of graph enumeration. The test is terminated once all the nodes have the same label or all the nodes are re-labelled. For example, the connectivity test is performed for an isomer of C_6H_6 represented by the adjacency matrix A (Figure 7a) with its initial node enumeration (labels) $\{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$,

(a) (b)

$$\begin{bmatrix} 0 & 3 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

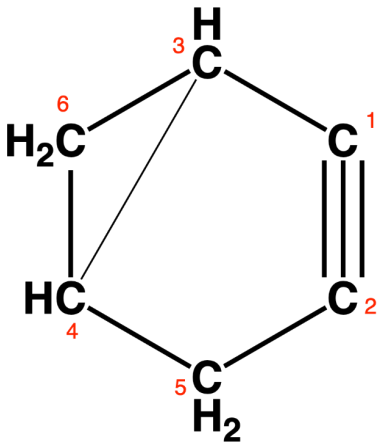

Figure 7: a) The adjacency matrix of an isomer of C_6H_6. b) A isomer of C_6H_6.

Table 1: The connectivity test for an isomer of C_6H_6 represented by matrix A (Figure 7a).

Node Index	Neighbors	Former Label	Minimum Label	Enumeration
1	{1,2,3}	{1,2,3}	1	{1,1,1,4,5,6}
2	{2,5}	{1,5}	1	{1,1,1,4,1,6}
3	{3,4,6}	{1,4,6}	1	{1,1,1,1,1,1}

The matrix A (Figure 7a) is connected since the smallest node label for each tested node is1 and its last node enumeration list includes only 1s. Thus there is only one component whose smallest index is 1 (Figure 7b). For a disconnected chemical graph represented by the adjacency matrix B (Figure 8a) with its initial node enumeration (labels) $\{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$.

(a)

$$\begin{bmatrix} 0 & 3 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

(b)

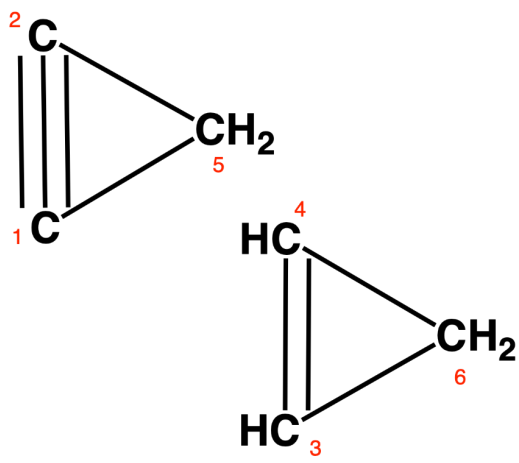

Figure 8. a) The adjacency matrix of an isomer of C_6H_6. b) A disconnected molecule with formula C_6H_6.

Table 2: The connectivity test for an isomer of C_6H_6 represented by matrix B (Figure 8a).

Node Index	Neighbors	Former Label	Minimum Label	Enumeration
1	{1,2,5}	{1,2,5}	1	{1,1,3,4,1,6}
2	{2,5}	{1}	1	{1,1,3,4,1,6}
3	{3,4,6}	{3,4,6}	3	{1,1,3,4,1,6}
4	{4,6}	{3}	3	{1,1,3,3,1,3}
5	{5}	{1}	1	{1,1,3,3,1,3}
6	{6}	{3}	3	{1,1,3,3,1,3}

The matrix B represents a disconnected isomer of C_6H_6. This molecule has two components (Figure 8b) with the indices $\varsigma_1 = \{1, 2, 5\}$ and $\varsigma_2 = \{3, 4, 6\}$. The first component ς_1 is the first component with respect to its atom labelling. Here, components are compared with respect to their maximum index.

2.2.4. Learning From Connectivity Test

Similar to Section 2.2.4, there is still something to learn from the connectivity test if a molecule is not connected. In MAYGEN, the connectivity test is performed when a canonical matrix is

complete. If a molecule is not connected, it is not stored in the output file and its first component needs to be detected. For example, the matrix B with Table 2, its first component is $\varsigma_1 = \{1, 2, 5\}$. The maximum index of the first component identifies where the graph gets disconnected.

In Algorithm 1, when a matrix is complete and stored in the output file, the generation process continues with the backward function. Here, the last index of the matrix is used as the input. However, with the “learning from connectivity test”, the algorithm continues with the last entry of the first component. For example, in matrix B, the first component is $\varsigma_1 = \{1, 2, 5\}$ and the maximum index is 5. Thus, the graph gets disconnected after the last entry of the fifth row, $B[5, 6]$ entry of the matrix B. All the other modifications on the matrix between its last entry [6,6] and [5,6] build only disconnected graphs. That is why the matrix modification process continues with the last entry of the first component. The learning from the connectivity test reduces the construction of disconnected graphs.

4. Results

MAYGEN is written purely in Java and hosted on GitHub (see section Availability). The full source code, as well as precompiled binaries, are available for download.
The code can be executed as follows:

```
> java -jar MAYGEN.jar

usage: java -jar MAYGEN.jar -f <arg> [-v] [-d <arg>]

Generates
molecular structures for a given molecular formula. The input is a
molecular formula string. For example 'C2OH4'.Besides this formula,
the directory is needed to be specified for the outputfile.

 -f,--formula <arg>   formula (required)
 -v,--verbose         print message
 -t,--tsvoutput       output formula, number of structures and
execution time in CSV format
 -o,--filename <arg>  store output in given file
```

In order to generate constitutional isomers, the user passes a molecular formula with the -f option:

```
> java -jar MAYGEN.jar -f C10H16
MAYGEN is generating isomers of C10H16...
The number of structures is: 24938
```

```
Time: 1.590 seconds
```

Alternatively, users who either want to contribute to the development or use the latest source code can clone the GitHub repository and build the MAYGEN binary using the Maven build environment.

For the purpose of this publication, MAYGEN was tested with randomly selected molecular formulae. The run times of MAYGEN and OMG are compared in Table 3, Table 4 and Table 5. Different from MOLGEN 5.0 [4], OMG generates structures for additional valences of sulfur (S), phosphorus (P) and nitrogen (N) and therefore more molecules than MOLGEN or MAYGEN [2]. MOLGEN 5.0 uses the default lowest valences for N(3), S(2), and P(3); unless a user defines the higher valences. For all the results given in these tables, MAYGEN generated the same number of structures as MOLGEN 3.5. The numbers reported in the tables are given for Molgen 5.0 since this could be run in batch mode on our Linux servers. Molgen 5.0 has an aromaticity filter that filters out resonance structures of substituted aromatic molecules. This filter is activated by default and therefore Molgen 5.0 generates fewer structures than Molgen 3.5 and MAYGEN. Since bromine (Br) atom type is not defined in OMG it does not generate structures with molecular formulae including Br.

Table 3: The number of structures and the run times are listed for MOLGEN 5.0, MAYGEN and OMG (molecules do not contain N, P or S). A more diverse set of molecular formulae is benchmarked in Table 5. Molgen 5.0 has an aromaticity filter that filters out resonance structures of substituted aromatic molecules. This filter is activated by default and therefore Molgen 5.0 generates fewer structures than Molgen 3.5 and MAYGEN.

Formula	# Molecules	MOLGEN Run Time (s)	# Molecules	MAYGEN Run Time (s)	OMG Run Time (s)
$C_3Cl_2H_4$	7	0.006	7	0.070	0.219
$C_3O_3H_4$	152	0.026	152	0.110	0.389
C_6H_6	217	0.049	217	0.159	0.454
$Cl_2C_5H_4$	217	0.028	217	0.116	0.659
C_5H_9ClO	334	0.009	334	0.147	0.745
$C_6OF_2H_{12}$	536	0.032	536	0.210	6.219
C_7H_{10}	575	0.078	575	0.148	0.877

$C_6O_2H_{12}$	1,313	0.082	1,313	0.294	3.503
C_6OH_6	2,237	0.092	2,237	0.292	2.706
$C_6H_7F_2I$	3,523	0.053	3,523	0.293	12.233
C_8H_{10}	4,678	0.250	4,679	0.354	6.533
$C_5F_2O_2H_2$	7,094	0.125	7,094	0.359	52.146
C_7OH_{10}	7,166	0.192	7,166	0.422	9.209
$C_4ClHF_2O_3$	7,346	0.076	7,346	0.542	159.133
$C_4O_5H_6$	8,070	0.110	8,070	0.634	27.074
$C_5ClHF_2O_2$	12,400	0.146	12,400	0.484	309.788
C_9H_{12}	19,980	0.783	19,983	1	36.798
$C_6H_{10}O_2Br_2$	24,201	0.197	24,201	1	N/A
$C_{10}H_{16}$	24,938	1	24,938	1	86.477
C_6H_6ClOI	30,728	0.226	30,728	0.834	70.666
$C_7O_2H_{10}$	54,641	0.411	54,641	2	98.448
C_8OH_{10}	69,659	0.811	69,669	2	119.510

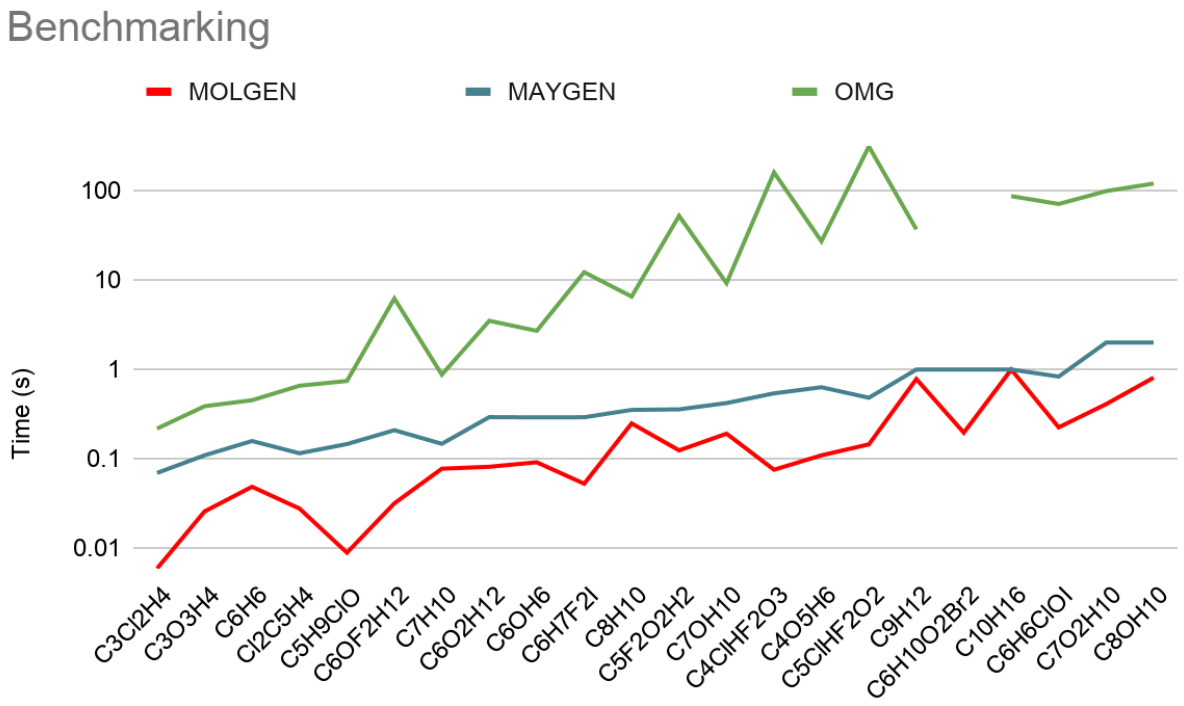

Figure 9: Timing of structure generation runs with MOLGEN 5.0, MAYGEN and OMG for molecular formulae containing carbon, hydrogen, oxygen and halogens, but not containing nitrogen, phosphorus or sulfur. The latter were omitted to keep fair conditions because OMG generates additional structures with higher oxidation states for those omitted elements.

Table 4: The number of structures and the run times are listed for MOLGEN 5.0, MAYGEN and OMG (molecules do not contain N, P or S). A more diverse set of molecular formulae is benchmarked in Table 5. Molgen 5.0 has an aromaticity filter that filters out resonance structures of substituted aromatic molecules. This filter is activated by default and therefore Molgen 5.0 generates fewer structures than Molgen 3.5 and MAYGEN.

Formula	# Structures	MOLGEN (s)	# Structures	MAYGEN (s)	# Structures	OMG (s)
C_9	811	0.333	832	0.739	832	225
C_9H_6	56,106	5	56,437	1	56,437	166
C_9H_8	57,615	2	57,771	2	57,771	122
C_9H_8O	1,011,969	13	1,013,745	13	1,013,745	2,790
$C_9H_8O_2$	9,979,619	80	9,990,575	117	9,990,575	38,260
C_{10}	4,197	1	4,330	4	4,330	2,481

$C_{10}H_6$	436,444	44	439,373	7	439,373	2,066
$C_{10}H_8$	486,354	33	488,125	8	488,125	1,598
$C_{10}H_6O$	8,633,799	242	8,671,508	103	8,671,508	53,640
$C_{10}H_6O_2$	93,695,925	1,572	93,964,875	1,204	N/A	>24 h
C_{11}	24,542	7	25,227	58	25,227	27,854
$C_{11}H_8$	4,423,944	355	4,442,438	64	4,442,438	19,048
$C_{11}H_{10}$	3,606,031	238	3,614,427	53	3,614,427	12,910
$C_{11}H_{10}O$	79,695,287	1,363	79,818,477	1,057	N/A	>24 h
$C_{11}H_{10}O_2$	954,738,367	8,913	955,729,849	13,191	N/A	>24 h
C_{12}	167,791	53	171,886	1,994	N/A	>24 h
$C_{12}H_{14}$	11,443,070	796	11,451,841	210	11451841	45930,809
$C_{12}H_{16}$	4,263,152	223	4,264,429	93	4,264,429	18,317
$C_{12}H_{16}O$	99,529,810	1,243	99,549,462	1,874	N/A	>24 h
$C_{12}H_{16}O_2$	1,263,998,275	8,148	1,264,165,511	23,294	N/A	>24 h

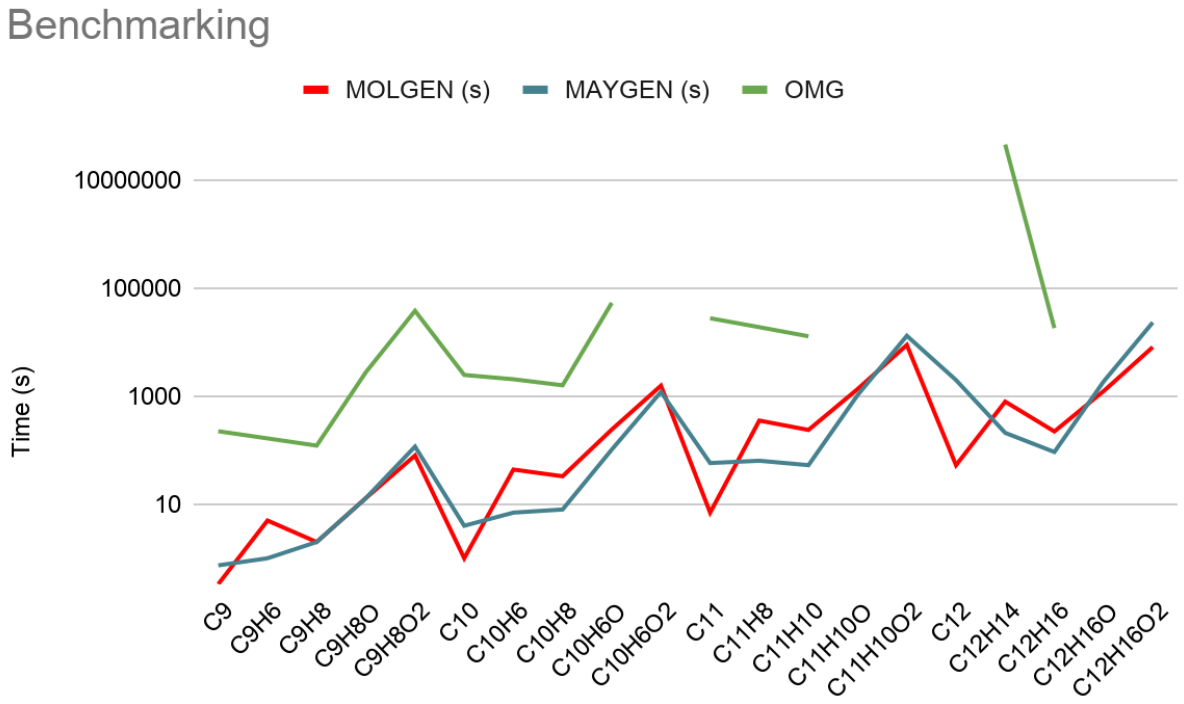

Figure 10. Times for structure generation runs with MOLGEN 5.0, MAYGEN and OMG for molecular formulae containing only carbon, hydrogen, oxygen. For this common elemental composition, MAYGEN achieves approximately the same performance as Molgen.

Table 5: The number of structures and the run times are listed for MOLGEN, MAYGEN and OMG with a diverse set of molecular formulae. MAYGEN always generates the same number of structures as MOLGEN. Molgen 5.0 has an aromaticity filter that filters out resonance structures of substituted aromatic molecules. This filter is activated by default and therefore Molgen 5.0 generates fewer structures than Molgen 3.5 and MAYGEN. OMG generates more structures in some cases due to different valences of S, P and N, which is why the per molecule run time is also given in milliseconds (ms).

Formula	# Molecules	MOLGEN Run Time (s)	Per molecule (ms)	# Molecules	MAYGEN Run Time (s)	Per molecule (ms)	# Molecules	OMG Run Time (s)	Per molecule (ms)
$C_2NO_2H_5$	84	0.007	0.083	84	0.085	1	97	0.359	4
$P_3O_3NCl_2$	665	0.021	0.031	665	0.170	0.256	3,862	379	98
$F_2P_3BrNO_2H$	1,958	0.024	0.012	1,958	0.233	0.118	N/A	N/A	N/A

C_5H_6BrN	2,325	0.041	0.017	2,325	0.232	0.099	N/A	N/A	N/A
$C_5H_5SI_5$	2,619	0.081	0.030	2,619	0.654	0.249	21,648	464	21
$C_3O_3NH_5$	2,644	0.064	0.024	2,644	0.254	0.096	3,733	6	2
C_5H_9ClOS	3,763	0.059	0.015	3,763	0.252	0.066	15,721	149	9
$C_3NO_2SH_7$	3,838	0.064	0.016	3,838	0.291	0.075	15,978	100	6
$C_4H_8Cl_3O_2P$	9,313	0.102	0.010	9,313	1	0.107	15,776	814	52
$C_5H_2F_2SO$	13,446	0.153	0.011	13,446	0.568	0.042	67,720	645	10
$C_5H_{10}BrF_2OP$	15,009	0.090	0.005	15,009	2	0.133	N/A	N/A	N/A
$C_7H_{11}ClS$	15,093	0.220	0.014	15,093	0.957	0.063	73,183	1,455	20
$C_4NO_3H_7$	18,469	0.212	0.011	18,469	0.767	0.041	26,530	94	4
$C_4H_5O_2F_2P$	41,067	0.237	0.005	41,067	2	0.048	59,035	317	5
$C_4H_5O_2Br_2N$	41,067	0.225	0.005	41,067	2	0.048	N/A	N/A	N/A
$C_3N_3O_2H_7$	45,626	0.291	0.006	45,626	1	0.021	124,808	1,378	11
$C_5N_3H_9$	46,125	0.469	0.010	46,125	1	0.021	134,278	2,635	20
$C_3O_6PH_5$	51,323	0.388	0.007	51,323	2	0.038	83,977	564	7
$C_4H_{10}NOSP$	52,151	0.280	0.005	52,151	2	0.038	303,760	51,917	171
$C_5H_5POBr_2$	62,886	0.405	0.006	62,886	1	0.015	N/A	N/A	N/A
$C_4H_8OI_2NP$	65,980	0.343	0.005	65,980	2	0.030	127,328	10,277	81
$C_4H_5NO_2S$	73,045	0.460	0.006	73,045	2	0.027	486,829	6,353	13
$C_4H_6N_2O_2$	75,211	0.532	0.007	75,211	2	0.026	174,687	2,518	14
$C_5H_4N_2O$	86,900	0.813	0.009	87,055	1	0.011	280,491	3,891	14
$C_4HN_3O_2$	94,247	0.824	0.008	94,422	2	0.021	1,175,05	73,605	63

							6		
C_5H_5ClNOF	111,921	0.520	0.004	111,921	2	0.017	186,322	2,778	15
$C_4H_2N_2O_3$	131,260	0.853	0.006	131,318	3	0.022	568,038	17,063	30
$C_6H_2N_2O$	238,916	3	0.012	240,339	3	0.012	1,231,657	46,971	38
$C_6H_4N_2O$	736272	6	0.008	738,283	8	0.010	2,951,633	49,887	17
$C_5HN_3O_2$	846180	8	0.009	848,498	10	0.011	N/A	N/A	>24 h
$C_5H_2N_2O_3$	1196267	8	0.006	1,197,634	14	0.011	N/A	N/A	>24 h
$C_7H_{11}NO_2$	1495599	6	0.004	1,495,599	20	0.013	N/A	N/A	>24 h
$C_4H_3N_3O_3$	2407836	11	0.004	2,408,635	28	0.011	N/A	N/A	>24 h
$C_5H_8N_2O_3$	3223855	11	0.003	3,223,855	40	0.012	N/A	N/A	>24 h
$C_7H_9NO_2$	3236782	14	0.004	3,237,132	38	0.011	N/A	N/A	>24 h
$C_5H_6N_2O_3$	4513529	17	0.003	4,513,867	54	0.011	N/A	N/A	>24 h
$C_5H_5N_3O_2$	9386119	40	0.004	9,390,618	94	0.010	N/A	N/A	>24 h
$C_7H_6N_2O$	10486969	66	0.006	10,504,307	100	0.009	N/A	N/A	>24 h

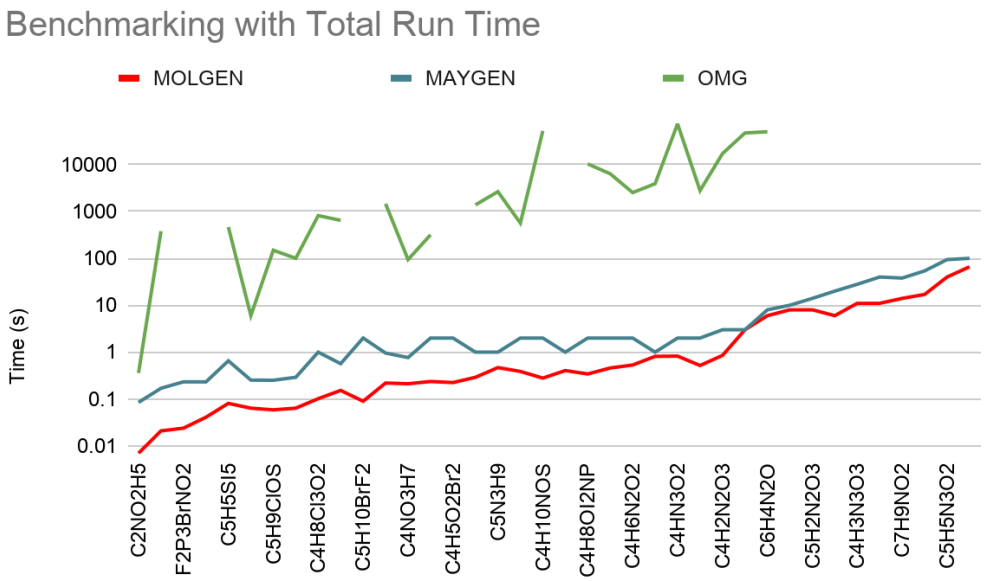

Figure 11. Times for structure generation runs with MOLGEN 5.0, MAYGEN and OMG for molecular formulae containing all allowed elements (carbon, hydrogen, oxygen, nitrogen, phosphorus, sulfur and halogens. In previous figures, N, S and P were omitted to keep fair conditions because OMG generates additional structures with higher oxidation states for those omitted elements. The total run times (s) are plotted. For a fairer comparison, Figure 12 shows the per-molecule run times.

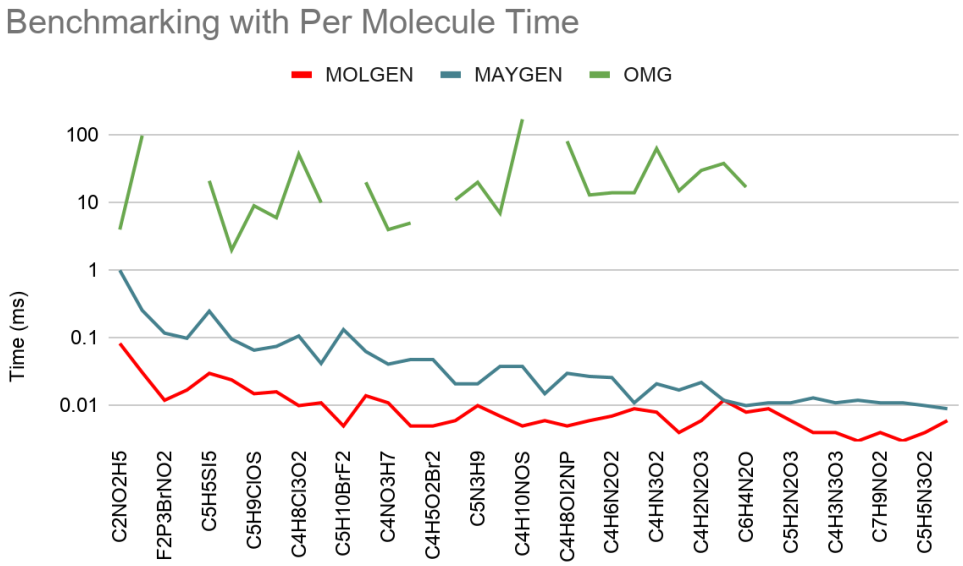

Figure 12. Times for structure generation runs with MOLGEN 5.0, MAYGEN and OMG for molecular formulae containing all allowed elements (carbon, hydrogen, oxygen, nitrogen, phosphorus, sulfur and halogens. Since OMG generates additional structures with higher

oxidation states for N, S and P the run times (ms) for the construction of per molecule are plotted.

For most structures containing all allowed elements, MOLGEN was slightly faster than MAYGEN and much faster than OMG (Figure 9-12); for carbohydrates and those containing additional oxygen, MAYGEN's execution speed was comparable to that of MOLGEN. OMG was not able to generate some of the tested molecular formulae within a day. For these formulae, "24>h" is added to the tables. Since the bromine (Br) atom type is not defined in OMG; "N/A" is added to the result tables. These results are visualized with spaces in the plots (Figure 9-12).

Limitations

MAYGEN is currently restricted to generate molecules with the lowest valence states of nitrogen, phosphorus and sulfur, and all testing and benchmarking was done under this boundary condition. This is no principle restriction - the algorithm will work with any given valence state - but the workflow logic of MAYGEN needs to be adapted to compute structures for higher valences of these elements.

5. Future Work

Being implemented in pure Java and with its code completely open, MAYGEN can be easily extended with additional functionalities and algorithmic improvements. The code availability through GitHub invites the scientific community to contribute to the further developments of MAYGEN. Obvious future work includes performance enhancements. Moreover, we aim to integrate MAYGEN into the Chemistry Development Kit (CDK) [5] in the near future which will enable an easy integration of the molecular structure generator in other software programmatically.
Furthermore, it is desirable that MAYGEN can use substructures in its input as building blocks, in order to include them as badlists or goodlists into the generation and therefore reduce the number of candidate structures to generate. This will enable its use in systems for computer-assisted structure elucidation (CASE) whose aim is to elucidate chemical structures from NMR and mass spectral data.

6. Conclusion

In this manuscript we presented MAYGEN, an open-source constitutional isomer generator completely written in Java. MAYGEN generates constitutional isomer spaces exhaustively and avoids isomorphic structures during the generation using the principles of orderly canonical graph generation.
We presented extensive testing of MAYGEN against two alternative solutions:
MAYGEN outperforms the current best single-thread structure generator OMG by one order of magnitude and is only marginally slower than the fastest in the current state-of-the-art software

MOLGEN. We expect MORGEN to be a starting point for further developments in the area of chemical structure generation by the open source, open science community.

Availability and Requirements

- Project name: MAYGEN
- Project home page: https://github.com/MehmetAzizYirik/MAYGEN
- Operating system(s): Platform independent
- Programming language: Java
- License: MIT

Competing Interests

All authors declare no competing interests.

Funding

MAY and CS acknowledge funding by the Carl-Zeiss-Foundation.
MS was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Project-ID 239748522, SFB 1127 ChemBioSys.

Acknowledgements

We wish to acknowledge the helpful discussion with Dr. Ronald Grund about the principles of orderly generation and the help from Valentyn Kolesnikov for the performance tuning.

Author contributions

MAY developed MAYGEN and performed the evaluation and testing. MS contributed with advice, code-review and made the figures. CS conceived the project and guided the development. All authors wrote, read and approved the manuscript.

Reference

1. Yirik MA, Steinbeck C. Chemical graph generators. PLOS Comput Biol. 2021;17:e1008504. doi:10.1371/journal.pcbi.1008504.
2. Peironcely JE, Rojas-Chertó M, Fichera D, Reijmers T, Coulier L, Faulon JL, et al. OMG: Open molecule generator. J Cheminformatics. 2012;4:1–13.
3. Grund R, Müller R. Konstruktion molekularer Graphen mit gegebenen Hybridisierungen und überlappungsfreien Fragmenten. Lehrstuhl II für Mathematik; 1995.

4. Gugisch R, Kerber A, Kohnert A, Laue R, Meringer M, Rücker C, et al. MOLGEN 5.0, a Molecular Structure Generator in Advances in Mathematical Chemistry. Adv Math Chem Basak SC Restrepo G Villaveces JL Eds.
5. Willighagen EL, Mayfield JW, Alvarsson J, Berg A, Carlsson L, Jeliazkova N, et al. The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching. J Cheminformatics. 2017;9:1–19.