JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

pubs.acs.org/jcim

Get Your Atoms in Order—An Open-Source Implementation of a Novel and Robust Molecular Canonicalization Algorithm

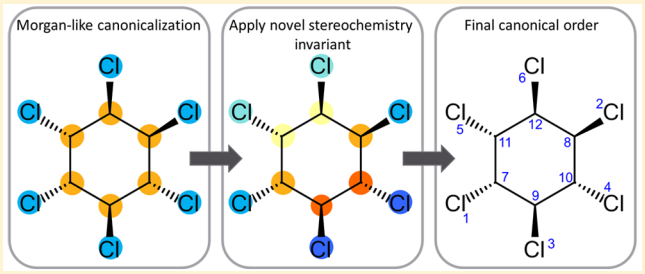
Nadine Schneider,*[,†] Roger A. Sayle,[‡] and Gregory A. Landrum[†]

[†]Novartis Institutes for BioMedical Research, Novartis Pharma AG, Novartis Campus, CH-4002 Basel, Switzerland

[‡]NextMove Software Ltd., Innovation Centre, Unit 23, Science Park, Milton Road, Cambridge CB4 0EY, U.K.

Supporting Information

ABSTRACT: Finding a canonical ordering of the atoms in a molecule is a prerequisite for generating a unique representation of the molecule. The canonicalization of a molecule is usually accomplished by applying some sort of graph relaxation algorithm, the most common of which is the Morgan algorithm. There are known issues with that algorithm that lead to noncanonical atom orderings as well as problems when it is applied to large molecules like proteins. Furthermore, each cheminformatics toolkit or software provides its own version of a canonical ordering, most based on unpublished algorithms, which also complicates the generation of a universal unique identifier for molecules. We present an alternative canonicalization approach that uses a standard stable-sorting algorithm instead of a Morgan-like index. Two new invariants that allow canonical ordering of molecules with dependent chirality as well as those with highly symmetrical cyclic graphs have been developed. The new approach proved to be robust and fast when tested on the 1.45 million compounds of the ChEMBL 20 data set in different scenarios like random renumbering of input atoms or SMILES round tripping. Our new algorithm is able to generate a canonical order of the atoms of protein molecules within a few milliseconds. The novel algorithm is implemented in the open-source cheminformatics toolkit RDKit. With this paper, we provide a reference Python implementation of the algorithm that could easily be integrated in any cheminformatics toolkit. This provides a first step toward a common standard for canonical atom ordering to generate a universal unique identifier for molecules other than InChI.



INTRODUCTION

A proper representation and a unique identifier of chemical structures are essential for storing and identifying molecules in databases, for the registration of novel compounds, and for searching for structures on the Web. Many different representations of molecules exist, ranging from sketches to line notations like SMILES[1] and SYBYL line notation (SLN)[2,3] to file formats that enable the storage of two- or three-dimensional coordinates (e.g., MOL/CTAB[4]) as well as miscellaneous properties related to the structure (e.g., SDF[4]). Allowing different representations of a molecule increases the difficulty of identifying duplicates and recovering structures in databases. Solving these problems requires a unique identifier that is able to capture and store the characteristic properties of a structure, like atoms, bonds, charge, and stereochemistry. About 25 years ago Weininger et al. published a method (CANGEN and CANON) to generate a canonical SMILES representation for molecules.[5] Since the Daylight software[6] that included this method was commercial, many different variations of this method have been developed and implemented in different toolkits and software packages. A drawback of the Weininger method was that, as published, it did not handle stereochemistry in molecules. In 2005 the International Chemical Identifier (InChI) representation was released, which was endorsed by IUPAC.[7−10] It provides a canonical representation of the molecule and offers different layers of description for the chemical structure. The implementation to generate an InChI for a molecule is freely available under an open-source license, and therefore, the code has been integrated in several toolkits. Two major limitations of InChI are that the normalization performed (e.g., of tautomers) imposes a particular model of chemistry and that this identifier is hardly human-readable compared with, e.g., canonical SMILES. On the basis of the canonical atom order provided by InChI, O'Boyle developed a new method to generate a canonical SMILES.[11] This approach has been proposed as a common standard for canonical SMILES.

The generation of a canonical atom order for a molecule can be used as the basis for the creation of a unique identifier. Usually, graph relaxation algorithms are employed to create a canonical order of the nodes. To accomplish this, the symmetry classes of the nodes have to be determined in order to allow the nodes to be discriminated from each other and ordered. In general, graph relaxation algorithms operate in two phases: in the initial phase, approximate values are assigned to the nodes on the basis of their local properties, and in the second phase, the relaxation step, these values are iteratively refined to obtain

Received: August 28, 2015
Published: October 6, 2015

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

Scheme 1. Pseudocode for the Canonicalization Algorithm[a]

```
def canonizeGraph(graph):
  # initialize the equivalence classes of the nodes/atoms of the graph
  equivalenceClassesNodes = createInitialEquivalenceClassesNodes(len(graph.nodes))
  # start the refinement process using these initial equivalence classes (see Scheme 2)
  refinePartitions(graph, equivalenceClassesNodes)
  # if nodes/atoms were assigned to the same equivalence class use more sophisticated invariants
  if len (equivalenceClassesNodes) < len(graph.nodes):
    # try more specialized properties before tiebreaking
    # first include a special invariant to handle dependent chirality
    graph.nodeProperties = setRDKitSpecialChiralityProperties(graph)
    # rerun the refinement to further discriminate the nodes/atoms
    refinePartitions(graph, equivalenceClassesNodes)
    # try a very specialized graph invariant which helps with highly symmetrical graphs
    graph.nodeProperties = setSpecialSymmetryPropertiesCanonGraph(graph)
    # rerun the refinement to further discriminate the nodes/atoms
    refinePartitions(graph, equivalenceClassesNodes)
    # finally if there are still ties break them
    breakTies(graph, equivalenceClassesNodes)
# return the canonical order
return equivalenceClassesNodes
```

[a]The Python code for this part of the algorithm can be found in pythonCanonImplementation/rdkitCanonizer.py in Additional file 1.

a unique value for each node. Within the field of cheminformatics, multiple different algorithms have been developed to obtain a canonical atom labeling (see, e.g., refs 12−26). The most widely used algorithm is the classical Morgan algorithm,[12] which was developed in 1965 for the Chemical Abstracts Service. This algorithm includes an extended connectivity: during the relaxation phase of the algorithm, the current equivalence classes of the immediate neighbors of an atom are employed to update its equivalence class. The initial equivalence class of an atom is assigned on the basis of its degree, atomic number, and bond type. In the refinement phase, the current equivalence classes of the immediate neighbors are added, and this sum is assigned to the atom being considered. Randić has pointed out that issues exist with this algorithm: for instance, nonequivalent atoms can be assigned the same extended connectivity values.[15] Different strategies have been developed to further improve the algorithm and address these problems (see, e.g., refs 5, 13, 19, 20, and 24). Randić developed a method that employs the eigenvector of the adjacency matrix exhibiting the largest eigenvalue.[15] Weininger et al. used the products of primes in their algorithm to obtain unique values.[5] For larger molecules these values can become quite large, leading to numerical problems even on systems supporting 64-bit integers. Many of the limitations of these approaches arise from the fact that the algorithms had been implemented in early versions of FORTRAN that allowed the sorting of only numbers and not objects like vectors or atoms. A different solution was proposed by Rohde, who employed a stable numbering for equivalence classes instead of a sequential index.[20]

Here we present a novel canonical ordering algorithm that also uses a stable index for equivalence classes and implements an efficient partition-based graph relaxation method.[27] In the next section of the paper, we outline the implementation of our new method and provide some details about two special invariants we have developed that allow the construction of a canonical atom order for highly symmetrical molecules. The subsequent section summarizes the results of extensive testing of the method on a large, diverse set of molecules. In addition, we provide a runtime analysis and discuss the limitations of the method. Finally, we conclude and outline some ideas for alternative applications of our algorithm.

METHODS AND IMPLEMENTATION

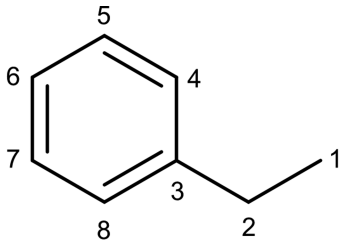
Canonical labeling of molecular graphs belongs to the class of symmetry perception problems. In the initial phase, these algorithms assign to each node an equivalence class that is based on local topological properties of the node. For molecules, these graph invariants can be augmented with chemical features. In subsequent relaxation steps, the symmetry class of each node is iteratively updated on the basis of the equivalence classes of its immediate neighbors. In general, our new approach retains this overall procedure. In the following we give an outline of the algorithm and highlight the differences with respect to existing methods. The C++ implementation of our novel approach was done within the open-source cheminformatics toolkit RDKit[28] and is available for version 2015.09 and higher. The implementation in an RDKit prerelease[29] corresponds to what is described here. In addition, a toolkit-agnostic Python implementation of the algorithm is provided in Additional file 1.

Overview. The new canonicalization algorithm involves three steps: the creation of an initial partition of the molecular graph, an iterative refinement of the partitions, and finally, if there are still atoms assigned to the same symmetry class, a tiebreaking step. The initial partition contains all of the atoms in the molecule. This is passed to the refinement process, in which atoms are iteratively divided into smaller partitions using

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

an atom/graph invariant until no further differentiation is possible. At this point the refinement process terminates, and tiebreaking is performed. Although this general version provides a canonical order for most molecules, molecules with dependent stereocenters and some highly symmetrical cyclic graphs require more elaborate graph invariants; these are described in Two New Invariants. These invariants are applied in two separate refinement steps before any potential tiebreaking is done. The pseudocode for the algorithm is given in Scheme 1.

The initial atom invariant is based on common atom properties: degree, atomic number, isotope, total number of hydrogen atoms attached, charge, atom stereochemistry, and bonds (bond type, bond stereo) (for more details, see pythonCanonImplementation/RDKit_Graph_Invariants.py in Additional file 1). Using this invariant leads to a set of different partitions for ordinary molecules. Each partition is assigned an index in the range of the number of atoms of the molecule. An essential feature of our algorithm is that these indices are not assigned consecutively but rather are assigned in a way that allows them to remain stable during the refinement (for more details, see lines 75ff in pythonCanonImplementation/CanonGraph.py in Additional file 1). Figure 1 shows an example



Atom	1	2	3	4	5	6	7	8	Index after:	
Index	1	1	1	1	1	1	1	1	Initialization	Iterative refinement
	1	7	8	2	2	2	2	2	Initial atom invariant	
	1	7	8	5	3	2	3	5	Including neighbors	
	1	7	8	5	3	2	4	6	Tiebreaking	

Figure 1. Derivation of the canonical atom order for a simple molecule. In the table, the development of the index/equivalence class of the atoms/nodes during the iterative refinement and the final tiebreaking are outlined. Partitions considered for refinement are marked in gray.

illustrating how a stable index is retained during our canonicalization approach. First, all of the atoms in the molecule are assigned the same index (see row 2 of the table in Figure 1). After the first round of refinement, the atoms are split into four different partitions, three of which are already finalized since they consist of only one atom (see row 3 of the table in Figure 1). The methyl group of the molecule is assigned to the lowest index since this atom has the smallest degree. Atom 3, with a degree of 3, has the highest index. The remaining six atoms all have a degree of 2, but atom 2 has two hydrogen atoms attached instead of one for the aromatic carbon atoms; therefore, it is assigned the second highest index. The five aromatic carbon atoms cannot be further differentiated using the atom invariant; hence, they are all assigned to the same partition with the second lowest index. It should be noted that the indices are stable during the refinement and tiebreaking. This is accomplished by leaving enough space for partitions to be split without affecting the surrounding indices: we assign the same index i_1 to all n_1 atoms in a partition and calculate the next higher index i_2 for the following partition as follows: $i_2 = i_1 + n_1$. In the example molecule in Figure 1, this leads to an index of 7 (=2 + 5) for atom 2. Providing a stable index allows iterative refinement of the individual partitions. In the following refinement process, only the index of the atom and its immediate neighbors are considered for each partition. This splits partition 2 into three smaller partitions (see row 4 of the table in Figure 1). Atoms 4 and 8 are assigned to the partition with the highest index since the index of neighboring atom 3 is higher than the indices of the atom neighbors of the remaining atoms in the partition. Atoms 5 and 7 are assigned to the partition with the second highest index, while atom 6 keeps its original index. In the final tiebreaking, partitions 3 and 5 are each split into two partitions (see row 5 of the table in Figure 1). Keeping the index stable during the canonicalization not only allows us to iteratively refine partition by partition but also makes the algorithm more efficient since only partitions that were affected by the previous iteration have to be updated.

In the following sections the refinement process, the tiebreaking procedure, and the two new invariants are described in more detail.

Partition-Based Graph Relaxation. Classical implementations of graph relaxation algorithms used to obtain a canonical labeling of the nodes, like the Morgan algorithm, require $O(n^2 \log(n))$ time, where n is the number of atoms in the molecule. This is due to the relaxation phase, which may require n passes to generate a canonical label for each of the n atoms. These conventional methods try to assign a unique numeric index to each equivalence class. Usually those indices are assigned consecutively, forcing the algorithm to re-sort all n atoms in each pass to guarantee consecutive indices. Our new implementation requires on average less than n passes by keeping the index stable and applying a partition-based approach in the relaxation phase. This allows re-evaluation of only those partitions that contain atoms whose neighbors' indices have been updated in the previous pass. The partitions that need to be re-examined are determined in each pass of the iterative refinement. The order of the atoms within a partition can be established using any stable-sorting algorithm (e.g., insertion sort). The sorting algorithm is provided with an atom−atom comparison function that uses the current indices/equivalence classes of the atoms. If the sort splits the evaluated partition, the indices of the atoms and the list of partitions are updated. Finally, partitions that may be affected by the newly created ones are identified, and these are appended to the list of partitions that need to be re-evaluated. During this refinement process, it is essential to process the partitions in a defined order to guarantee a canonical labeling of the atoms. In our approach, the partitions are sorted by the highest index/equivalence class before re-examination. The iterative refinement process terminates when all of the partitions assigned for re-evaluation have been investigated. The pseudocode for the iterative refinement process is given in Scheme 2, and more details can be found in the Python implementation in pythonCanonImplementation/CanonGraph.py in Additional file 1.

As mentioned above, any stable-sorting algorithm can be used to determine the order of the atoms within a partition. In the C++ version of our new approach included in the RDKit, we implemented a customized sort algorithm based on the mergesort "Towers of Hanoi" algorithm.[30,31] This algorithm was selected because it can efficiently sort data sets with many

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

Scheme 2. Pseudocode for the Iterative Refinement Process[a]

```
def refinePartitions(graph, equivalenceClassesNodes):
  # determine the partitions which include more than one node/atom, these need to be refined
  partitionsToRefine = determinePartitionsToRefine(equivalenceClassesNodes)
  # sort the partitions based on largest indices of the equivalence classes
  partitionsToRefine = sorted(partitionsToRefine , reverse=True)
  # refine the equivalence classes of the nodes/atoms until no further refinement is possible
  while len(partitionsToRefine) > 0:
    # get the partition with highest equivalenceClass index
    partitionToSort = partitionsToRefine.pop()
    # sort the current partition using a stable sorting algorithm and providing a comparison key
    sortedElements = sorted(partitionToSort, key=graph.getKeyNode)
    # check if new equivalence classes has been found
    newEquiClasses = checkForNewEquivalenceClasses(sortedElements, equivalenceClassesNodes)
    if len(newEquiClasses) > 0:
      # update the equivalence classes of the graph nodes considering a stable index of the classes
      updateEquivalenceClassesGraph(graph, equivalenceClassesNodes)
      # get the partitions which are affected by the new equivalence classes
      affectedPartitions = getAffectedPartitions(graph, equivalenceClassesNodes)
      # add the partitions at the beginning which should be re-analyzed
      for partition in sorted(affectedPartitions):
        partitionsToRefine.insert(0, partition)
```

[a]The Python code for this part of the algorithm can be found in pythonCanonImplementation/CanonGraph.py in Additional file 1.

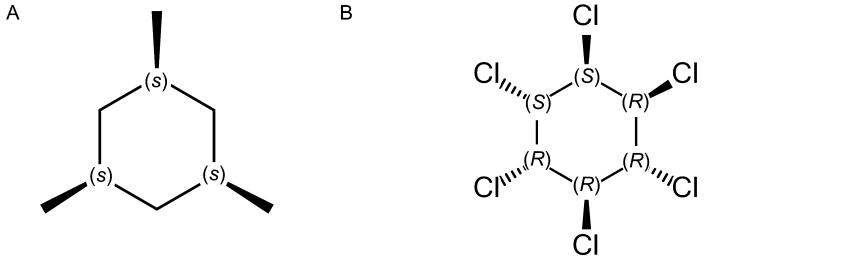


Figure 2. Two exemplary molecules with dependent stereochemistry (molecule B is ChEMBL1714528). For these molecules, no canonical atom order could be obtained using the usual atom invariant. The isomeric SMILES indicating the stereochemistry are shown below the molecules.

duplicate keys, obtaining an upper bound of $O(m \log(m) + (n - m))$ for sorting n elements with m different values. The implementation details can be found in ref 30 and in the RDKit source code.[29]

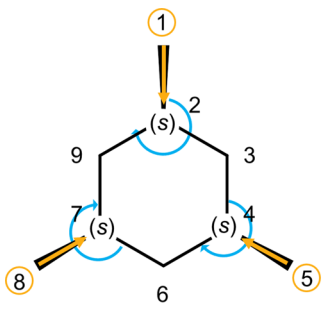
Tiebreaking. For many molecules the iterative refinement process will not provide unique indices for all of the atoms, for instance, if symmetric phenyl groups are present (see Figure 1). These atoms cannot be further discriminated using either the atom invariants or in the subsequent relaxation phase (see row 4 of the table in Figure 1). In the tiebreaking routine of our new approach, all partitions containing more than one member are considered for re-evaluation. Again it is important to examine the partitions in a defined order. In each iteration of the tiebreaking procedure, the largest member of the current partition is assigned to the highest possible index within that partition. For example, in Figure 1, atom 7 is assigned to index 4 since it is the highest possible index within partition 3, which consists of two members (see row 4 of the table in Figure 1). It is worth mentioning that the ordering of the atoms within a partition is not arbitrary if the sorting algorithm is guaranteed to be stable. This is an important prerequisite to be able to find a canonical order of the atoms during the tiebreaking phase. Resolving a tie leads to splitting of the current partition into two partitions, one of which is already finalized. Afterward, the iterative refinement process is applied again in order to find unique indices for the remaining atoms. This process is continued until all ties are broken and a canonical labeling of the atoms can be provided. More details can be found in the Python implementation (see pythonCanonImplementation/CanonGraph.py in Additional file 1).

Two New Invariants. In the previous section we described the tiebreaking procedure, which is necessary to be able to provide a canonical order of the atoms for many molecules. In some cases this effort may not be sufficient for a proper canonicalization of a molecule. For instance, molecules exhibiting dependent stereochemistry (see the examples in Figure 2) would have atoms incorrectly assigned to the same equivalence class that would be distinguishable if stereochemistry is correctly incorporated. This is indispensable when unique isomeric SMILES are required. We have developed a

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

novel invariant in our canonicalization approach that is able to handle such molecules without knowing the explicit Cahn–Ingold–Prelog (CIP) chirality of the molecule. A second type of molecule or graph that would fail when the ordinary atom invariant is used is shown in Figure 4 below. The examples in Figure 4 are highly symmetrical and would result in one or two equivalence classes after the relaxation phase. We have also developed a novel invariant to handle these rather exotic molecules. Both invariants are described in more detail below.

The invariant taking chirality into account will help to distinguish atoms attached to chiral centers. For standard chiral centers, the atom neighbors should easily be distinguishable during the refinement because they are, by definition, different. However, in the case of dependent chirality, which occurs only for highly symmetric molecules, at least two of the neighbors will seem to be the same. This kind of chirality is determined only by a different constitution of the neighbors, and as indicated by the term "dependent chirality", at least two chiral centers in the molecule are necessary. As a prerequisite for the chirality invariant, all of the chiral centers need to be marked as such and the constitution (clockwise or anticlockwise) must be given, so no further specification of CIP codes (*R* or *S*) is necessary. The invariant is dependent on the current index/equivalence class of the immediate neighbors of the chiral atom and is recalculated in each iteration. The original constitution of the chiral atom is compared with the current constitution that would arise if the current index of the neighbors determines their order. This is used to calculate the number of swaps required to interconvert from the original constitution to the current constitution. If the number of swaps is even, the chiral atom retains its constitution, and otherwise it is inverted. In the current constitution, the atom neighbor currently being investigated is taken as the starting atom while the other neighbors are sorted according to their indices (highest index first). More details can be found in lines 124 ff of pythonCanonImplementation/RDKit_Graph_Invariants.py in Additional file 1. Figure 3 outlines the derivation of the invariant for an example molecule. Without the chirality invariant, three equivalence classes are obtained after the refinement process. When the invariant is applied, atoms 1, 5, and 8 can be distinguished from each other. This breaks the tie in the symmetrical molecule, and a canonical order is obtained for all of the atoms.



Atom 1: Origin constitution: 1 3 9
Current constitution: 1 3 9
Swaps: 0
Final constitution: @

Atom 5: Origin constitution: 3 5 6
Current constitution: 5 3 6
Swaps: 1
Final constitution: @@

Atom 8: Origin constitution: 6 8 9
Current constitution: 8 6 9
Swaps: 1
Final constitution: @@

Atom	1	2	3	4	5	6	7	8	9	Index after:
Index	1	7	4	7	1	4	7	1	4	Refinement
	1	7	4	7	2	4	7	2	4	1. iteration special invariant
	1	7	4	8	2	6	9	3	5	Refinement with special invariant

Figure 3. Schematic derivation of the chirality invariant. Atoms that can be discriminated using this invariant are circled in orange. The origin constitution is shown in blue. The current constitution is dependent on the investigated atom and the indices of the other neighbors.

The second invariant was also developed for highly symmetrical molecules containing many fused rings of different sizes, such as the molecules/graphs in Figure 4. For graph A,

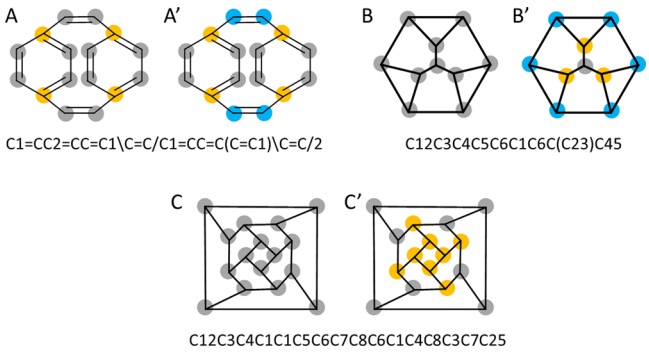


Figure 4. Three highly symmetrical molecules/graphs. For these molecules, no canonical atom order could be obtained using the usual atom invariant. The isomeric SMILES are shown below the molecules. SMILES were generated using Marvin Sketch.[39] Examples B and C were taken from Razinger et al.[36] (see structures I and Q, respectively, in Figure 8 of ref 36). Different equivalence classes are colored differently. In A′, B′, and C′, the additional equivalence classes obtained with the high-symmetry invariant are highlighted.

two equivalence classes would be found with the usual atom invariant (gray- and orange-circled nodes), obviously leading to one missing class (Figure 4, blue-circled nodes in A′). All of the atoms/nodes in graphs B and C in Figure 4 would be assigned to the same equivalence class (gray-circled nodes) using the usual atom invariant. When the high-symmetry invariant is applied, the nodes of graph B′ are split into three different equivalence classes (gray-, orange-, and blue-circled nodes), while the nodes of graph C′ are split into two different classes (gray- and orange-circled nodes).

To calculate this special invariant, a breadth-first search (BFS) is conducted from each ring node in the graph. The BFS is terminated when a neighbor is not a ring member. During the BFS traversal, the number of newly found neighbors at each level is recorded, as well as the number of neighbors that are revisited. These numbers are concatenated to build the new invariant, which is then used in the refinement process. More details can be found in pythonCanonImplementation/Special-SymmetryGraph_Invariant.py in Additional file 1. Figure 5 outlines the derivation of this invariant for the simple symmetrical graph shown in Figure 5A.

Figure 5B shows the BFS traversal for node 1. In the first iteration, three new neighbors are visited. To determine the number of revisited neighbors, the neighbor lists of the newly found neighbors are scanned to detect nodes that were present at the current or previous level. In the first iteration, this results in scanning the neighbor lists of nodes 2, 7, and 8. In the neighbor list of node 2, the only node that had been previously visited was node 1. In the neighbor list of node 7, nodes 1 and 8 were found ("revisited"). Finally, in the neighbor list of node 8, nodes 1 and 7 were revisited. This yields five revisited neighbors in the first iteration and corresponds to a revisited-neighbor characterization code of "113" (1 for nodes 7 and 8 and 3 for node 1). Two further iterations are executed before the BFS traversal terminates for node 1 (see Figure 5B).

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111–2120

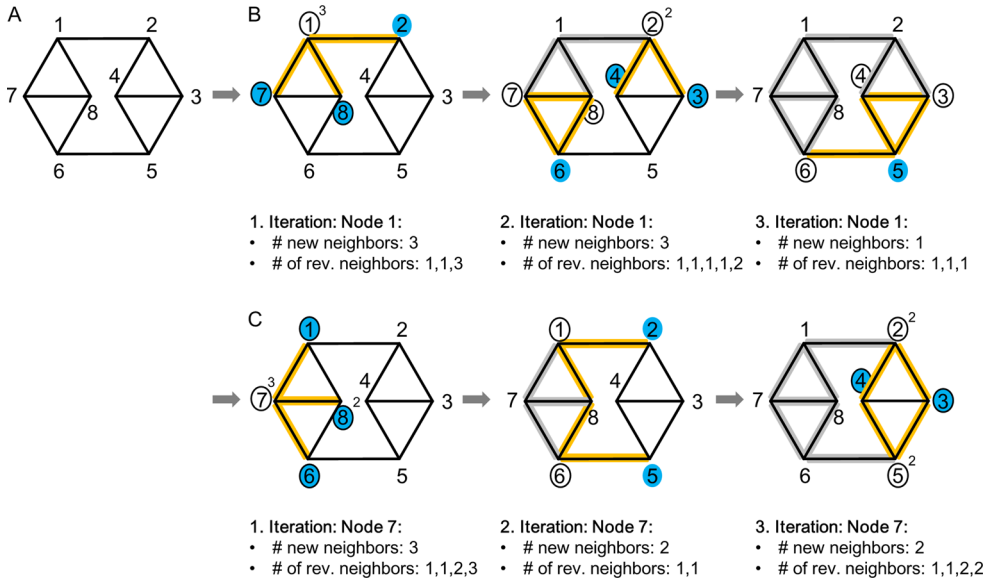


Figure 5. Schema of the derivation of the high-symmetry invariant for highly symmetrical cyclic graphs. (A) Exemplary graph/molecule for which the usual invariant does not result in a canonical order. The graph nodes are consecutively numbered. (B) BFS traversal starting from node 1 to calculate the invariant for node 1. Newly found neighboring nodes are marked in blue; neighboring nodes that are revisited in this level are outlined in black; superscript numbers indicate the number of times a node is revisited; edges to reach the neighboring nodes are marked in orange; edges already finished are marked in gray. (C) BFS traversal starting from node 7 to calculate the invariant for node 7.

Finally, the invariant calculated for node 1 is the "number of neighbors per level" = 3|3|1| and the "revisited-neighbor characterization code per level" = 113|11112|111|. The same result is also obtained for nodes 2, 5, and 6. In contrast, for node 7 (see Figure 5C), node 2, node 3, and node 8, the invariant produces the code 3|2|2| for the number of neighbors per level and the code 1123|11|1122| for the number of revisited neighbors per level. This difference in invariants yields a secondary equivalence class for the highly symmetrical graph in Figure 5A.

More details concerning both invariants can be found in the Python code in Additional file 1. While the first invariant is currently dependent on the RDKit toolkit, the second invariant can be directly calculated on the basis of the underlying graph.

RESULTS AND DISCUSSION

In this section we describe the validation of the robustness and correctness of our new canonicalization. In addition, we compare the runtimes for canonical atom ranking using the previous RDKit version (2014.09),[32] a version produced for this paper[29] that includes our new algorithm, and the Python implementation of the canonicalization that is provided with this publication. Our primary validation set was the ChEMBL 20 data set[33] for validation since it represents a large set of diverse molecules spanning a range of sizes and complexities. Other data sets, including subsets of ZINC[34,35] and a set of very symmetrical graphs,[36] were also evaluated.

Renumbering and SMILES Round-Tripping Tests. To evaluate the robustness of our new approach, we used the entire ChEMBL 20 data set, which consists of 1 456 020 different molecules taken from the medicinal chemistry literature and other public sources. Many of these represent marketed drugs or drug candidates. Figure 6 middle shows the distribution of molecular sizes in the ChEMBL 20 data set. A median-sized molecule contains 27 heavy atoms, while the largest one is composed of 878 heavy atoms. All of these molecules were parsed in sdf format using the RDKit Python interface. A unique isomeric SMILES was created for each of the molecules. From that SMILES a new molecule was generated, and the atoms were randomly renumbered. This renumbered molecule was used to produce a unique isomeric SMILES that was compared to original one. The random renumbering was repeated 50 times for each molecule. The Python code for this test is included in Additional file 1.

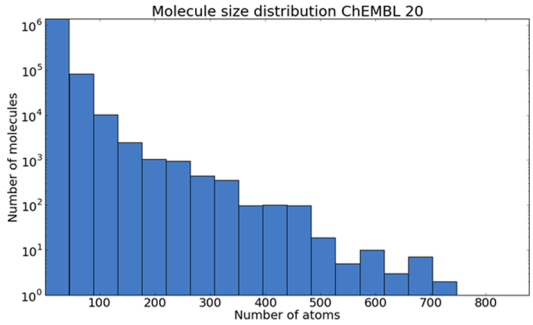
Of the 1 456 020 molecules, 256 led to errors during the parsing, most of which were related to problems in the aromaticity perception. All of the remaining 1 455 764 molecules except for two passed the renumbering and round-tripping test. The results are summarized in Figure 6. The two molecules that did not result in a canonical SMILES are shown in Figure 6 bottom. Both molecules are rather complex, the first one in terms of the number of fused rings and the second one in terms of a form of dependent chirality that propagates through the whole molecule. For both of these molecules, the error occurred in the SMILES round-tripping phase of the test and hence is an error in the SMILES writing rather than the canonicalization itself.

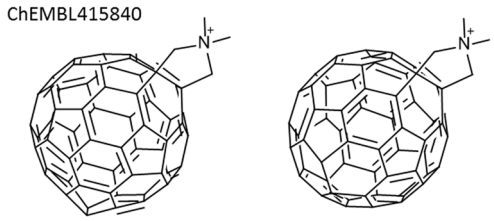
We also tested our new method on the ZINC Biogenic Compounds (ZBC) (180 313 molecules) and ZINC in Man (ZIM) (11 421 molecules) data sets[34] using the same test setup as described above. The ZBC data set presents an important source for the testing of our canonicalization since many of the molecules exhibit multiple chiral centers. Both data sets were provided in sdf format. The renumbering and SMILES round-tripping test ran successfully without errors on both sets, but one of the molecules of the ZBC set could not be correctly parsed.

To test our high-symmetry invariant, we used the highly symmetrical graphs from Razinger et al.[36] All of these graphs passed our renumbering and round-tripping test after our symmetry invariant was applied. SMILES of these special graphs are provided in Additional file 1. Another interesting and difficult set of graphs, including some nonplanar examples, was provided by one of the reviewers. Again, all of these

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

ChEMBL [sdf]	# molecules	# errors
Parsing	1,456,020	256
Canonicalization	1,455,764	2





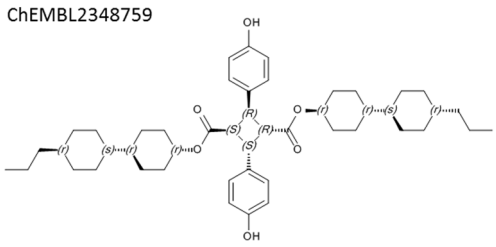


Figure 6. Results of the renumbering and SMILES round-tripping test on the molecules of ChEMBL 20. (top) Overview of errors occurring in different steps of the molecule processing. (middle) Distribution of molecular sizes within ChEMBL 20. (bottom) The two molecules that failed in the SMILES round-tripping test.

examples successfully passed our renumbering and round-tripping test and can be found in Additional file 1.

To rigorously prove that the atoms within one equivalence class obtained after the refinement phase are symmetry-equivalent, which would make it valid to use our tiebreaking procedure, a systematic test by renumbering using all permutations within the equivalence class would be required. More sophisticated approaches to this rather time-consuming brute-force process have been proposed in the literature. These include the use of generating sets—allowing many of the permutations to be skipped[20]—and a combination of backtracking and comparison of candidate adjacency matrices to prune the backtracking tree.[24] Since the use of appropriate atom invariants removes the need for this elaborate step in the canonicalization of almost all kinds of chemical graphs, we decided to not include it in our novel approach. We believe that the successful results from the extensive renumbering and round-tripping tests on a large set of diverse molecules as well as on very symmetrical graphs demonstrate the robustness of our new method.

Runtime. We measured the runtime of our new canonicalization methods on the ChEMBL 20 data set since it represents a broad range of different molecule sizes and complexities. In this test, the time to calculate a canonical atom order using the RDKit Python interface was measured. We did this test using the previous RDKit release (version 2014.09), in which the old Morgan-based canonicalization approach was implemented, and with the current prerelease,[29] which utilizes our novel algorithm. We found a 40% improvement in runtime in the prerelease compared with the previous one. With the 2014.09 release, 128 s was needed to generate a canonical order for 1 455 763 molecules on an Intel Xeon 64-bit 3.6 GHz eight-core processor. With the RDKit prerelease, only 77 s was needed to produce 1 455 764 canonical atom orderings. Figure 7 shows the runtime (in seconds per molecule) for the

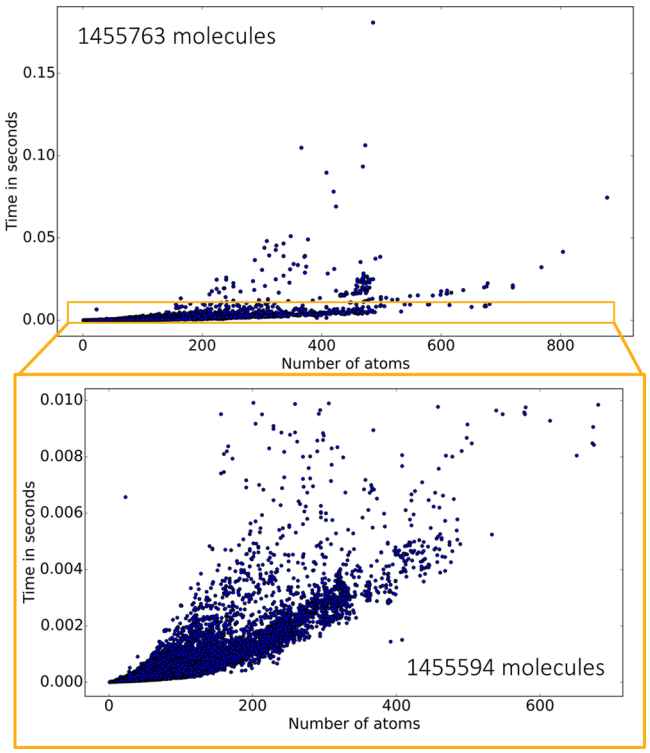


Figure 7. (top) Correlation of the number of atoms with the time needed to generate a canonical atom ordering for the molecules of the ChEMBL 20 data set. The numbers were obtained using the cheminformatics toolkit RDKit,[32] version 2014.09. (bottom) Detailed depiction of the range from 0 to 0.01 s.

ChEMBL 20 data set using the previous release. The plot demonstrates that the old canonicalization algorithm scales rather badly with the size of the molecule (Figure 7 top, upper right corner). The median time to generate a canonical atom order was 0.07 ms. Figure 7 bottom provides a detailed version of the scatter plot in the range of 0 to 0.01 s. This plot ignores the outliers (169 molecules).

In contrast, Figure 8A shows the results using the RDKit prerelease. The median time in seconds to create a unique atom ordering was reduced to 0.05 ms. The new canonicalization approach scales much better with molecular size than the old version. Even for the largest molecule, a peptide with 878 heavy atoms, the unique atom ordering was generated in less than 2 ms. We also measured the runtime for the Python implementation of our new algorithm. This implementation, which is not highly optimized, also uses the standard sorting algorithm provided in Python release 2.7 instead of the customized mergesort algorithm used in the C++ version. Generating the canonical atom ranking for all of the ChEMBL

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

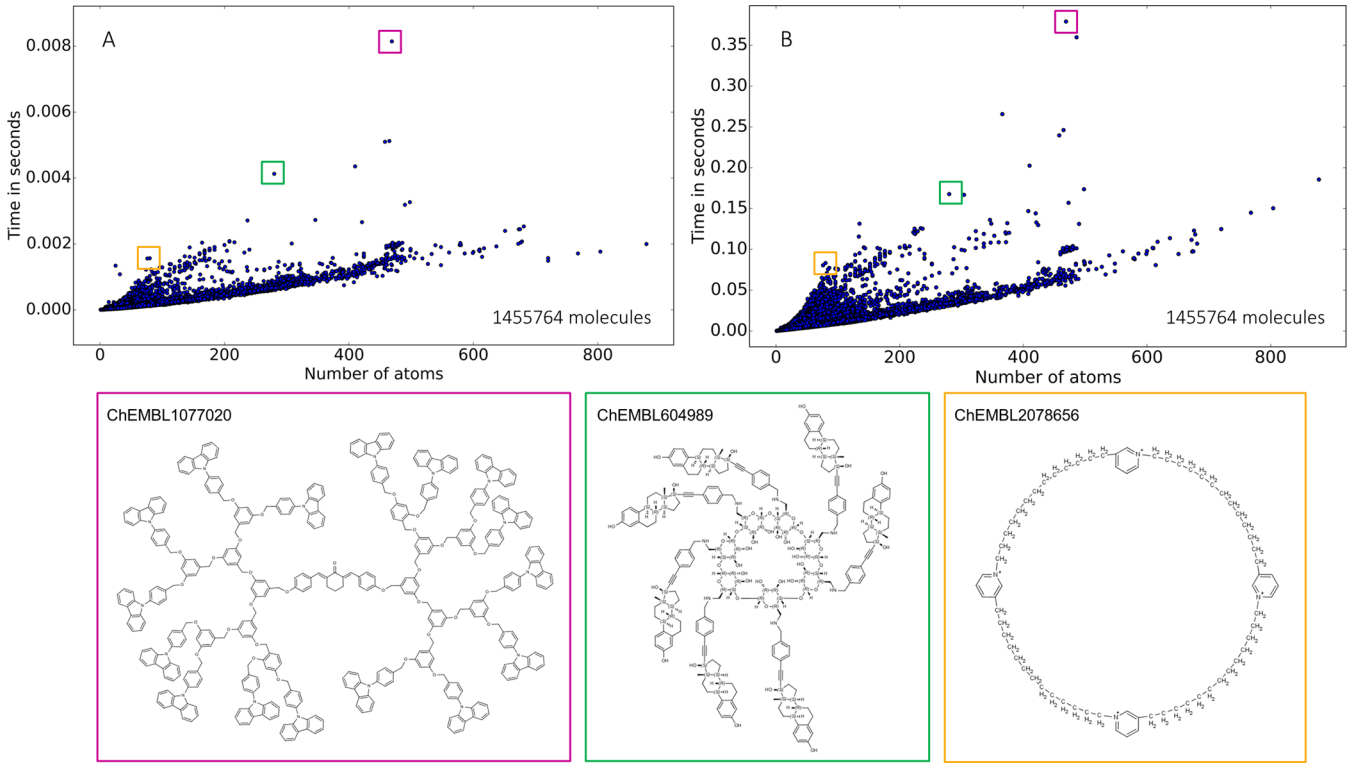


Figure 8. (A, B) Correlations of the number of atoms with the time needed to generate a canonical atom ordering of the molecules of the ChEMBL 20 data set based on numbers obtained using (A) the cheminformatics toolkit RDKit[29] and (B) the Python implementation of the new canonicalization approach and RDKit. (bottom) Structures of three outliers. ChEMBL identifiers are given above the molecules.

molecules took 1.35 h on the Intel Xeon 64-bit 3.6 GHz eight-core processor using the Python implementation. The median time to create a unique atom ordering was 3.1 ms, about 60 times slower than the optimized C++ implementation. Figure 8B shows the scatter plot of the results for the Python implementation. The scaling is very similar for the two versions. In both plots (C++ and Python version), we picked a few outliers and looked into the details (see Figure 8 bottom). The first outlier (ChEMBL1077020) is a highly symmetrical molecule that contains 469 heavy atoms but for which only 104 equivalence classes were identified during the refinement process. The atoms in the same equivalence class belong to different isolated ring systems, leading to many calls of the refinement process during the tiebreaking since each new assigned equivalence class does not propagate through the molecule. The second outlier (ChEMBL604989) is a molecule containing several carbohydrate and steroid groups (280 atoms). This molecule exhibits a high level of symmetry, leading to the application of both new invariants. For this particular molecule, the heuristics used—calculating the chirality invariant only for chiral neighbors and terminating the BFS traversal if a nonring atom is hit in the high-symmetry invariant—did not substantially decrease the number of calculations. The third outlier (ChEMBL 2078656) is a cyclic molecule consisting of 76 heavy atoms. In this case, the time-consuming step is also the derivation of the high-symmetry invariant. For this molecule, the invariant was calculated for almost all of the atoms since they all are members of the large ring formed by the molecule. This required a long BFS. A solution to this would be to restrict the ring size during the calculation of the invariant.

This analysis of the ChEMBL data set shows that we could significantly reduce the runtime of the new canonicalization in comparison with the previous Morgan-based implementation (see Figures 7 and 8A). The boundary for the latter is in the complexity class of $O(n^2 \log(n))$. Figure 8A strongly implies that this bound is reduced for our new algorithm. This is possible because of the partition-based relaxation, which does not require all of the equivalence classes to be resorted in each iteration. We propose that this bound is reduced to $O(n^2)$ for our new algorithm for two reasons. The first is that if at each iteration/refinement only a single atom is distinguished, we repeatedly/recursively sort a partition of n atoms into $n - 1$ identical values and one new value. The "Towers of Hanoi" mergesort guarantees that this process is $O(n)$, and hence, we have $O(n) + O(n - 1) + O(n - 2) + \cdots$, which is $O(n^2)$. The other is that the high-symmetry invariant requires $O(n)$ time for the BFS traversal for each of the n atoms, which is again an $O(n^2)$ process.

To get an impression of the average real-world scaling, we used the curve_fit function in the Python scipy package[37] to fit both curves—the previous Morgan-based implementation (Figure 7 top) and the new canonicalization (Figure 8A) including all of the outliers—to two different target functions. First we used $f(n) = an^b \log(n)$ as the target function, applying a least-squares fit. For the Morgan-based implementation, this indeed resulted in an average-case running time of $O(n^2 \log(n))$ (coefficients: $a = 1.62 \times 10^{-8}$, $b = 1.84$; standard errors (SEs): 1.43×10^{-10} for a and 1.52×10^{-3} for b; root-mean-square error (RMSE) = 0.00029 s), while for the new canonicalization an $O(n \log(n))$ fit was obtained (coefficients: $a = 3.7 \times 10^{-7}$, $b = 0.98$; SEs: 3.67×10^{-10} for a and 2.25×10^{-4} for b; RMSE = 1.8×10^{-5} s). As the second target function we tested $f(n) =$

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120

an^b. This resulted in an average-case running times of $O(n^2)$ for the Morgan-based implementation (coefficients: $a = 4.65 \times 10^{-8}$, $b = 2.02$; SEs: 3.95×10^{-10} for a and 1.46×10^{-3} for b; RMSE = 0.00029 s) and $O(n^{1.2})$ (coefficients: $a = 8.5 \times 10^{-7}$, $b = 1.2$; SEs: 8.16×10^{-10} for a and 2.17×10^{-4} for b; RMSE = 1.79×10^{-5} s) for the new canonicalization. Because similar RMSEs were obtained for the two target functions, it is not possible to select the best target function for the two implementations. Nevertheless, these results indicate that in our real-world test the previous Morgan-based implementation has at least quadratic scaling while the new implementation is closer to linear/quasilinear scaling.

We also tested the performance of the new canonicalization on a few example proteins from the Protein Data Bank (PDB).[38] For these large molecules we also obtained reasonable runtimes using the RDKit prerelease: 0.492 ms for 1CRN (327 atoms), 4.63 ms for 2NW4 (2031 atoms), and 5.75 ms for 2FVD (2501 atoms). Compared with the previous release, the new version is up to about 200 times faster for the largest test protein (2FVD, 1090 ms), and a reasonable amount of runtime was also saved for the other proteins (1CRN, 2.29 ms; 2NW4, 198 ms). This also confirms our observation that the new canonicalization scales much better for larger molecules than the old Morgan-based canonicalization approach did. Another evaluation of the performance of our new algorithm on biopolymers was recently done by Sayle et al.[40,41] They compared the performances of several cheminformatics toolkits on proteins containing up to 1000 amino acids. This resulted in reasonably good performance, with our new canonicalization being among the most efficient approaches.

In this evaluation we used a patched version of the current RDKit release (2015.03). The three patches can be found on GitHub[28] and will be part of the upcoming release. Two of these patches are related only to performance improvements, not the canonicalization itself. The third patch resolves a gap in our algorithm pointed out by one of the reviewers. A prerelease version of the RDKit with these patches applied is available.[29]

CONCLUSION

The novel canonicalization method presented here is an efficient and robust alternative to existing published approaches. Using a partition-based graph relaxation algorithm and keeping the index of the equivalence classes stable during the relaxation phase allows iterative refinement without requiring the calculation of unique attributes like polynomials or products of primes. Furthermore, the new algorithm shows better performance since not all of the atoms have to be resorted in each iteration. The two more-sophisticated invariants described herein allow the assignment of canonical labels even to highly symmetrical molecules. In our tests, the new implementation proved to be more efficient and robust compared with the Morgan-based method used in the previous version of RDKit.

The Python implementation of the approach that is provided in Additional file 1 is designed to be easily integrated into any cheminformatics toolkit to provide a canonical labeling of the atoms. Thereby it could be the first step to a common standard canonicalization approach to generate a universal unique identifier for a molecule. The canonical labeling could be combined with the universal SMILES representation proposed by O'Boyle.[11] In addition, the canonical ranking of the atoms could be used to generate a canonical Kekulé form of aromatic rings, which would allow the creation of canonical MOL files. Another application would be the construction of a molecular hash code (e.g., for proteins or peptides) to store in a database. In future work, we will investigate the use of the algorithm for chirality perception and CIP assignment.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acs.jcim.5b00543.

Additional file 1 containing the source code (ZIP). The implementation is based on RDKit version 2015.03[28] but can easily be adapted to any other cheminformatics toolkit. In the evaluation we used a patched version of the current release of RDKit version 2015.03. The three patches can be found on GitHub[28] or as a prerelease version[29] and will be part of the upcoming release. The folder "python_scripts" within Additional file 1 contains the source code for the canonicalization algorithm, the SMILES round-tripping and renumbering test, and some of the tough molecules.

Additional file 2 containing the testing webpage (ZIP). The folder "webpage" within this zip file contains an html file, "canonizer.html", that can be used to test the new RDKit canonicalization algorithm without having to install it.

AUTHOR INFORMATION

Corresponding Author

*E-mail: nadine-1.schneider@novartis.com.

Author Contributions

N.S. participated in the conception of the implementation and the algorithm, conducted the testing and validation, and drafted the manuscript. R.A.S. proposed the basic idea of the novel algorithm and participated in the conception of the implementation and the algorithm. G.A.L. participated in the conception of the implementation and the algorithm and helped to draft the manuscript. All of the authors read and approved the final manuscript.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

N.S. thanks the Novartis Institutes of BioMedical Research Education Office for a Presidential Postdoctoral Fellowship. The authors thank Bernhard Rohde for helpful discussions.

REFERENCES

(1) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Model.* **1988**, *28*, 31−36.

(2) Ash, S.; Cline, M. A.; Homer, R. W.; Hurst, T.; Smith, G. B. SYBYL Line Notation (SLN): A Versatile Language for Chemical Structure Representation. *J. Chem. Inf. Model.* **1997**, *37*, 71−79.

(3) Homer, R. W.; Swanson, J.; Jilek, R. J.; Hurst, T.; Clark, R. D. SYBYL Line Notation (SLN): A Single Notation To Represent Chemical Structures, Queries, Reactions, and Virtual Libraries. *J. Chem. Inf. Model.* **2008**, *48*, 2294−2307.

(4) Dalby, A.; Nourse, J. G.; Hounshell, W. D.; Gushurst, A. K.; Grier, D. L.; Leland, B. A.; Laufer, J. Description of Several Chemical Structure File Formats used by Computer Programs Developed at Molecular Design Limited. *J. Chem. Inf. Model.* **1992**, *32*, 244−255.

(5) Weininger, D.; Weininger, A.; Weininger, J. L. SMILES. 2. Algorithm for Generation of Unique SMILES Notation. *J. Chem. Inf. Model.* **1989**, *29*, 97−101.
(6) Daylight Chemical Information Systems, Inc. http://daylight.com (accessed May 19, 2015).
(7) Stein, S. E.; Heller, S. R.; Tchekhovskoi, D. An Open Standard for Chemical Structure Representation: The IUPAC Chemical Identifier. In *Proceedings of the 2003 International Chemical Information Conference, Nimes, France, October 19−22, 2003*; Collier, H., Ed.; Infonortics: Malmesbury, U.K., 2003; pp 131−143.
(8) The IUPAC International Chemical Identifier (InChI). http://www.iupac.org/home/publications/e-resources/inchi.html (accessed May 19, 2015).
(9) Heller, S. R.; McNaught, A.; Stein, S.; Tchekhovskoi, D.; Pletnev, I. V. InChI - The Worldwide Chemical Structure Identifier Standard. *J. Cheminf.* **2013**, *5*, 7.
(10) Heller, S. R.; McNaught, A.; Pletnev, I.; Stein, S.; Tchekhovskoi, D. InChI, the IUPAC International Chemical Identifier. *J. Cheminf.* **2015**, *7*, 23.
(11) O'Boyle, N. M. Towards a Universal SMILES representation - A standard method to generate canonical SMILES based on the InChI. *J. Cheminf.* **2012**, *4*, 22.
(12) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures - A Technique Developed at Chemical Abstracts Service. *J. Chem. Doc.* **1965**, *5*, 107−113.
(13) Wipke, W. T.; Dyott, T. M. Stereochemically Unique Naming Algorithm. *J. Am. Chem. Soc.* **1974**, *96*, 4834−4842.
(14) Randić, M. On the Recognition of Identical Graphs Representing Molecular Topology. *J. Chem. Phys.* **1974**, *60*, 3920−3928.
(15) Randić, M. On Unique Numbering of Atoms and Unique Codes for Molecular Graphs. *J. Chem. Inf. Model.* **1975**, *15*, 105−108.
(16) Randić, M. On Canonical Numbering of Atoms in a Molecule and Graph Isomorphism. *J. Chem. Inf. Model.* **1977**, *17*, 171−180.
(17) Jochum, C.; Gasteiger, J. Canonical Numbering and Constitutional Symmetry. *J. Chem. Inf. Model.* **1977**, *17*, 113−117.
(18) McKay, B. D. Practical Graph Isomorphism. *Congr. Numer.* **1981**, *30*, 45−87.
(19) Randic, M.; Brissey, G. M.; Wilkins, C. L. Computer Perception of Topological Symmetry via Canonical Numbering of Atoms. *J. Chem. Inf. Model.* **1981**, *21*, 52−59.
(20) Rohde, B. GM-Search. A System for Stereochemical Substructure Search. Ph.D. Thesis, Philosophical Faculty II, University of Zürich, Zürich, Switzerland, 1988.
(21) Agarwal, K. K.; Gelernter, H. L. A Computer-Oriented Linear Canonical Notational System for the Representation of Organic Structures with Stereochemistry. *J. Chem. Inf. Model.* **1994**, *34*, 463−479.
(22) Faulon, J. L. Isomorphism, Automorphism Partitioning, and Canonical Labeling can be Solved in Polynomial-time for Molecular Graphs. *J. Chem. Inf. Model.* **1998**, *38*, 432−444.
(23) Kelley B.: Graph Canonicalization (2003). http://www.drdobbs.com/graph-canonicalization/184405341 (accessed May 19, 2015).
(24) Braun, J.; Gugisch, R.; Kerber, A.; Laue, R.; Meringer, M.; Rücker, C. MOLGEN-CID - A Canonizer for Molecules and Graphs Accessible through the Internet. *J. Chem. Inf. Model.* **2004**, *44*, 542−548.
(25) Faulon, J. L.; Collins, M. J.; Carr, R. D. The Signature Molecular Descriptor. 4. Canonizing Molecules using Extended Valence Sequences. *J. Chem. Inf. Model.* **2004**, *44*, 427−436.
(26) Carbonell, P.; Carlsson, L.; Faulon, J. L. Stereo Signature Molecular Descriptor. *J. Chem. Inf. Model.* **2013**, *53*, 887−897.
(27) Paige, R.; Tarjan, R. E. Three Partition Refinement Algorithms. *SIAM J. Comput.* **1987**, *16*, 973−989.
(28) Landrum, G. A.; et al. RDKit: Open-Source Cheminformatics Software, version 2015.03. http://www.rdkit.org, https://github.com/rdkit/rdkit (accessed Sept 30, 2015).
(29) Landrum, G. A.; et al. RDKit: Open-Source Cheminformatics Software, pre-release version. DOI: 10.5281/zenodo.28640.
(30) "Towers of Hanoi" Mergesort Implementation. https://sourceware.org/ml/libc-alpha/2002-01/msg00218.html (accessed on May 19, 2015).
(31) Knuth, D. E. *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed.; Addison-Wesley Professional: Reading, MA, 1998; pp 409− 417.
(32) Landrum, G. A.; et al. RDKit: Open-Source Cheminformatics Software, version 2014.09 RDKit. DOI: 10.5281/zenodo.13217.
(33) Bento, P. A.; Gaulton, A.; Hersey, A.; Bellis, L. J.; Chambers, J.; Davies, M.; Krüger, F. A.; Light, Y.; Mak, L.; McGlinchey, S.; Nowotka, M.; Papadatos, G.; Santos, R.; Overington, J. P. The ChEMBL Bioactivity Database: An Update. *Nucleic Acids Res.* **2014**, *42* (D1), D1083−D1090.
(34) Irwin, J. J.; Sterling, T.; Mysinger, M. M.; Bolstad, E. S.; Coleman, R. G. ZINC: A Free Tool to Discover Chemistry for Biology. *J. Chem. Inf. Model.* **2012**, *52*, 1757−1768.
(35) ZIM and ZBC ZINC special subsets. http://zinc.docking.org/browse/subsets/special (accessed May 11, 2015).
(36) Razinger, M.; Balasubramanian, K.; Munk, M. E. Graph Automorphism Perception Algorithms in Computer-enhanced Structure Elucidation. *J. Chem. Inf. Model.* **1993**, *33*, 197−201.
(37) Jones, E.; Oliphant, T.; Peterson, P.; et al. SciPy: Open Source Scientific Tools for Python (2001). http://www.scipy.org/ (accessed May 19, 2015).
(38) Bernstein, F. C.; Koetzle, T. F.; Williams, G. J.; Meyer, E. E., Jr.; Brice, M.D.; Rodgers, J. R.; Kennard, O.; Shimanouchi, T.; Tasumi, M. The Protein Data Bank: A Computer-based Archival File for Macromolecular Structures. *J. Mol. Biol.* **1977**, *112*, 535−542.
(39) *MarvinSketch*, version 14.10.6.0; ChemAxon: Budapest, Hungary, 2014; http://www.chemaxon.com (accessed May 19, 2015).
(40) https://nextmovesoftware.com/blog/2015/08/24/biopolymer-canonicalisation-scaling-between-toolkits/ (accessed Aug 27, 2015).
(41) Sayle, R. A.; May, J.; O'Boyle, N. Generating Canonical Identifiers for Glycoproteins and Other Chemically Modified Biopolymers. Presented at the 250th ACS National Meeting, Boston, MA, Aug 16−20, 2015; Paper CINF-1.

DOI: 10.1021/acs.jcim.5b00543
J. Chem. Inf. Model. 2015, 55, 2111−2120