A Computationally-Realizable Rigorous Canonical Numbering Algorithm for Chemical Graphs with its Open-Source Implementation in Rust

Qin Wan*

Chiral Ltd., Fukuoka, Japan

E-mail: chiral.co.jp@gmail.com

Abstract

Canonical numbering of the vertices from a graph has been a challenging open issue for decades not only in the domain of graph theory but also in the cheminformatic applications. This paper presents an efficient, fast and rigorous approach for canonical numbering and symmetry perception as the first workable solution with theoretical completeness. The methodology is composed of a set of algorithms including extendable representation of vertex, high-performance sorting and graph reduction, etc. The canonical numbering of vertices can be generated in a short time through the novel vertex representation method. Furthermore, a new concept of graph reduction decreases the amount of computation to determine constitutional symmetry of complex graphs into the range of hardware capability. An open-source version of algorithms overall is implemented in Rust thanks to the features of safety, performance and robust abstraction of this modern programming language. The results of experiments on more than 2 million molecules from ChEMBL database has been given at the end.

Introduction

Canonical numbering is a fundamental algorithms in cheminformatics for a wide range of applications such as indexing of chemical compound, spectra interpretation, drug design, etc.[1] Related studies have been carried on for decades, as a result, numerous algorithms have been proposed and implemented. Morgan created the Extended Connectivity Algorithm[2] more than 40 years ago, which is considered to be the first efficient canonical numbering algorithm and used by Chemical Abstract Services. The Morgan algorithm was improved by many researchers later on, with the core concept of an iteration process, also named as graph relaxation, not altered. Weininger et al.[3] uses products of primes instead of sum of integers during the graph relaxation in order to remove the inherent ambiguity that will lead to false symmetry perceptions. Schneider et al.[4] introduces a chiral invariant and a highly-symmetry invariant so that the graph relaxation process can launch from a superior starting status. Out of graph-relaxation based algorithms, Randić[5] uses the eigenvalues and eigenvectors of the adjacency matrix to number the vertices. Jochum and Gasteiger[6] uses the topological molecular information to partition the atoms into several equivalent classes coarsely from the beginning, and then carry on more elaborate partitioning for each classes to differentiate the atoms from each other.

The drawbacks of the Morgan algorithm, pointed out by Jochum and Gasteiger,[6] are oscillatory behavior and inadequacy of maximum differentiation on the basis of connectivity. Besides, we have found out that, when a graph-relaxation algorithm encounters failure when dealing with special cases, to identify the root cause will be a too laborious task to be completed. This is caused by the intrinsic empirical characteristic of the graph relaxation algorithm. Eigenvalue based algorithms require a time-consuming process of solving eigenvalues for the adjacency matrix, leading to a loss of efficiency in comparison to their counterparts. Carhart[7] finds out a category of counterexamples towards Jochum and Gasteiger's algorithm. He is also of the opinion that, an explicit investigation on all the vertex permutations from the symmetric groups derived by a numbering algorithm, is inevitable for provable reliability of results given by the numbering algorithm.

Ivaniciuc[1] summarizes a universal scheme for canonical numbering algorithms. It consists of two parts: firstly Graph Invariant Vertex Partitioning (GIVP) and then Canonical Numbering generation by Automorphism Permutation (CNAP). During the GIVP process, the vertices are partitioned into several groups according to a comparable representation value for each vertex. The vertices in each group are considered to be symmetric, however, this symmetry given by the GIVP process could be false and the verification is unavoidable according to Carhart's declaration. The CNAP process has to be performed in which all possible permutations from these equivalent groups are generated for automorphism investigation to determine finally the authentic constitutional symmetry. We follow the custom from Ivaniciuc[1] to call a partitioned group of vertices as "an orbit" for the sake of a clear explanation.

From the standpoint of graph hashing,[8] the GIVP process can be described as, in abstracto, hashing all the vertices by the local properties and the topological environment into comparable values respectively, and then partitioning them according to these hash values into orbits. The necessity of the CNAP process is more intuitive from this perspective, as most of the time the topological molecular environment of each vertex is the most crucial factor to differentiate vertices, and this information is incomparable with higher dimensions in respect to the hashed comparable values. Therefore given any hash function, namely the vetex representation method in the GIVP process, there could always be "hash collisions" in which non-constitutionally-symmetric vertices are hashed to the same value and thus a false symmetry is concluded by the GIVP process. As shown in Figure 1, for some hash method, the symmetric vertices $C, C^{'}, C^{''}$ shall be hashed into equivalent values. Meanwhile the asymmetric vertices B, C could be hashed into equivalent values too, creating a situation of "hash collision" for the vertices B and C.

In this paper, we present a novel canonical numbering algorithm following the theoretical framework combining both GIVP and CNAP.[1] The vertex representation method and the

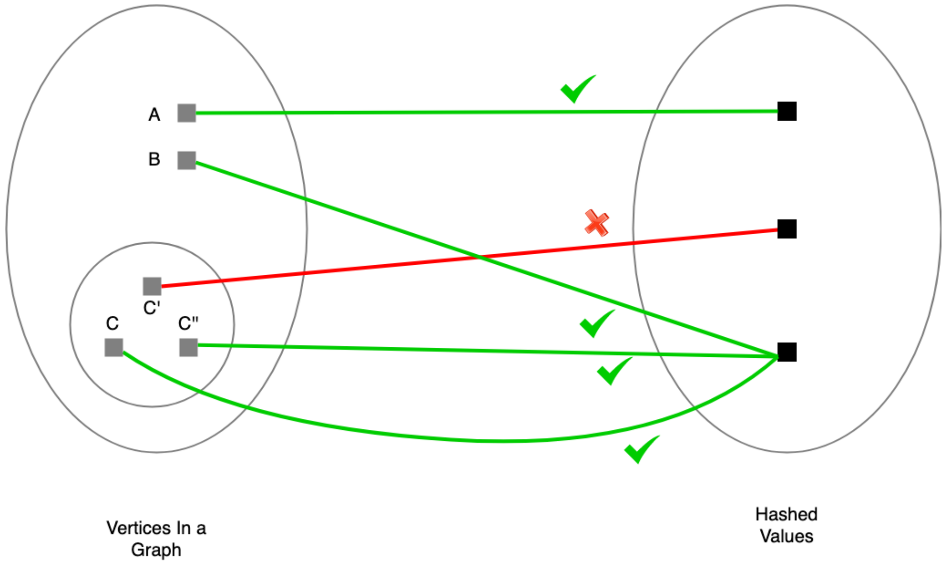

Figure 1: Vertex hashing

fast sorting algorithm are described in the Section GIVP. The CNAP algorithm is more easily comprehensible, and will be explained in the Section CNAP. Although the CNAP algorithm is theoretically straightway, the challenge remains that in practices the total computation often goes beyond hardware limitation if there is a considerable amount of symmetric vertices which cannot be differentiated by the GIVP process. In order to downscale the required amount of computation for the CNAP process, we have proposed a recursive graph reduction algorithm which turns a complicated graph into a simple one gradually until it can be solved directly by the CNAP algorithm. The details about graph reduction algorithm will be presented in Section Graph Reduction. Finally, explanation on implementation in the programming language Rust[9] of the algorithms mentioned above and the whole workflow are presented in Section Implementation. The experimental results on database ChEMBL[10] will be shown in the Section Experiment.

Graph Invariant Vertex Partitioning (GIVP)

In order to partition the vertices of a graph, a comparable value assigned to each vertex is required. The way how we get this comparable representation value from the local properties of the vertex and its topological graph environment, is called a representation method. Two types of vertex representation methods are defined in the GIVP algorithm of this paper, a fixed one based on the local properties and an extendable one based on the topological graph environment. A sorting method based on quicksort algorithm,[11] namely partition-exchange sort, together with a lazy extending strategy, is used for comparing the vertex representation values with high performance.

Vertex Representation

The fixed representation values are used to partition the vertices into orbits as an initial status for the GIVP process. The extendable representation values are then used to perform an elaborate differentiation step-by-step for each orbit. For molecular applications, the fixed representation value is composed of local properties of atom, such as degree, aromaticity, atomic number, charges etc. The extendable representation value is related to the topological molecular environment of atom. A sorted array of neighbour atom representation values is used as the extendable representation value, where a neighbour atom representation value is composed of bond type and atom numbering, as illustrated in Figure 2.

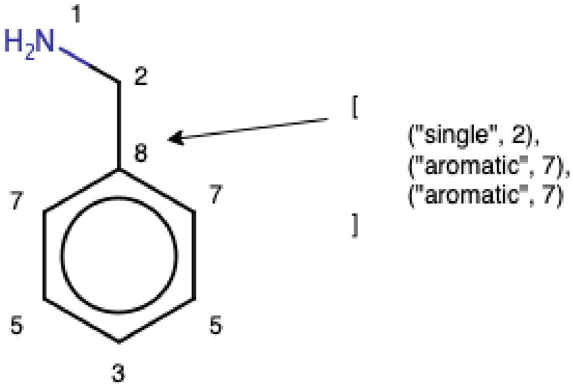

Figure 2: A representation value with bond type and neighbour atom numbering

The reason of making the representation for topological environment extendable, is that comparing topological connectivities could be a computationally heavy task for a sophisticated graph, e.g. a molecules with a large number of atoms. Figure 3 demonstrates the extending operations on the atom from aniline indexed as 5 with the extending step of 1 bond each time. In order to avoid unnecessary extension and comparison, the vertex representation value is extended with only a fixed number of edges before partitioning. Along with the "extending and partitioning in tandem" strategy explained in next section, the "lazy extending" strategy is applied that the extending operation is performed only inside the orbits for which further partitioning is required.

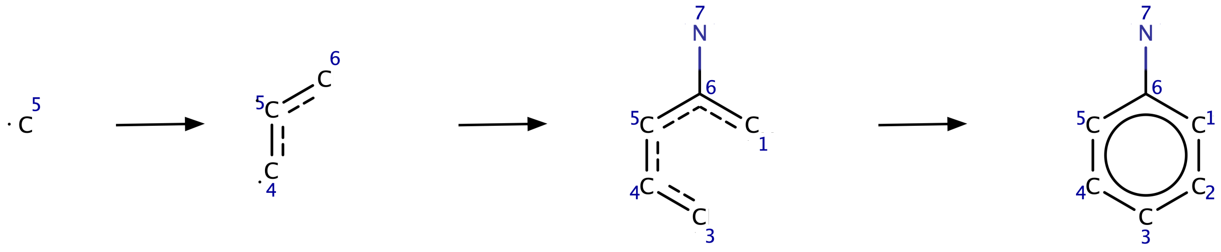

Figure 3: Extending operations of an aromatic carbon atom from aniline

Extending and Partitioning in Tandem

The “lazy-extending” together with the “extending-partitioning-iteration” is the crucial part of the GIVP algorithm. If a vertex is represented by a unique value, meaning that the vertex stays in its own orbit alone and it’s differentiated from all the other vertices. The partitioning process for the vertex has been completed in this case. The extending-partitioning process continues for those orbits with more than one vertex. This strategy will save the execution time substantially since the sorting operation is considered to be an algorithm with an average complexity $O(n \log n)$. Only the represented values inside the same orbit are compared and sorted which make a quite small n for the quicksort algorithm compared to the total number of vertices in the graph. Another advantage of this tandem strategy is memory saving. The temporarily equivalent vertices in the middle of the process are kept in the same orbit, therefore the representation values of the last extending step can be discarded.

The results from one partitioning process will be a set of partitioned orbits of vertices. In an orbit with more than one vertex, the representation values of any vertex in this orbit are equal. The extending-partitioning process will continue until the vertices are all differentiated or the extendable representation value cannot be extended any more. An extending-partitioning algorithm can be defined as follows:

Algorithm 1: The extending-partitioning algorithm

```
Input: An orbit of vertices InputOrbit to be partitioned
Output: OutputOrbits and the Numbering for each vertex
create an empty array ResidualOrbits;
add the InputOrbit as the first element of ResidualOrbits;
create an empty array OutputOrbits for storage of the results;
create an array Numbering for storage of the numbering after sorting each vertex;
while ResidualOrbits is not empty do
    pop one orbit out of ResidualOrbits;
    if the representative values reach the end of extending then
        save this orbit into OutputOrbits;
    else
        extend the representation values for the vertices in this orbit;
        partition the vertices according to the updated representation values via
          quicksort algorithm;
        update the Numbering accordingly;
        push those orbits with more than 1 vertex from the partitioned results into
          the ResidualOrbits;
    end
end
return OutputOrbits, Numbering
```

Workflow for the GIVP Process

The whole GIVP workflow based on the extending-partitioning algorithm could be described as follows:

```
Algorithm 2: The GIVP workflow
  Input: Properties for each vertex and topological connectivities of the graph
  Output: OutputOrbits and the Numbering for each vertex
  calculate the fixed representation values for each vertex;
  create an empty array ResidualOrbits;
  create an array Numbering to store the numbering for each vertex;
  partition all the vertices according to the fixed representation values via quicksort
    algorithm, save the orbits resulted into ResidualOrbits and update Numbering
    accordingly;
  create an empty array OutputOrbits for the storage of the final results;
  for an orbit with more than one vertex in ResidualOrbits do
      apply the extending-partitioning algorithm on this orbit;
      save the orbits from the output into OutputOrbits;
      update the Numbering accordingly;
  end
  return OrbitsOutput, Numbering
```

Figure 4 and Table 1 shows a very simple example as demonstration on how the GIVP process works on the molecule of aniline. From the fixed representation values, Atom 6 and 7 are partitioned into a solo orbit respectively. The extending-partitioning process continues for the orbit with Atoms 1, 2, 3, 4 and 5. The first extending-partitioning process differentiates Atoms 1 and 5 from Atoms 2, 3, 4. The second extending-partitioning process differentiates Atom 3 from Atom 2 and 4. Until the extending operations can be no more continued, Atom 1 cannot be differentiated from Atom 5, and Atom 2 cannot be differentiated from Atom 4.

Table 1: The partitioned orbits from the GIVP process for aniline

Extending operations	Orbits before partitioning	Orbits after partitioning
Fixed representation values	[1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 5], [6], [7]
Extend 1 bond	[1, 2, 3, 4, 5]	[1, 5], [2, 3, 4]
Extend 2 bonds	[1, 5], [2, 3, 4]	[1, 5], [2, 4], [3]
Extend 3 bonds	[1, 5], [2, 4]	[1, 5], [2, 4]

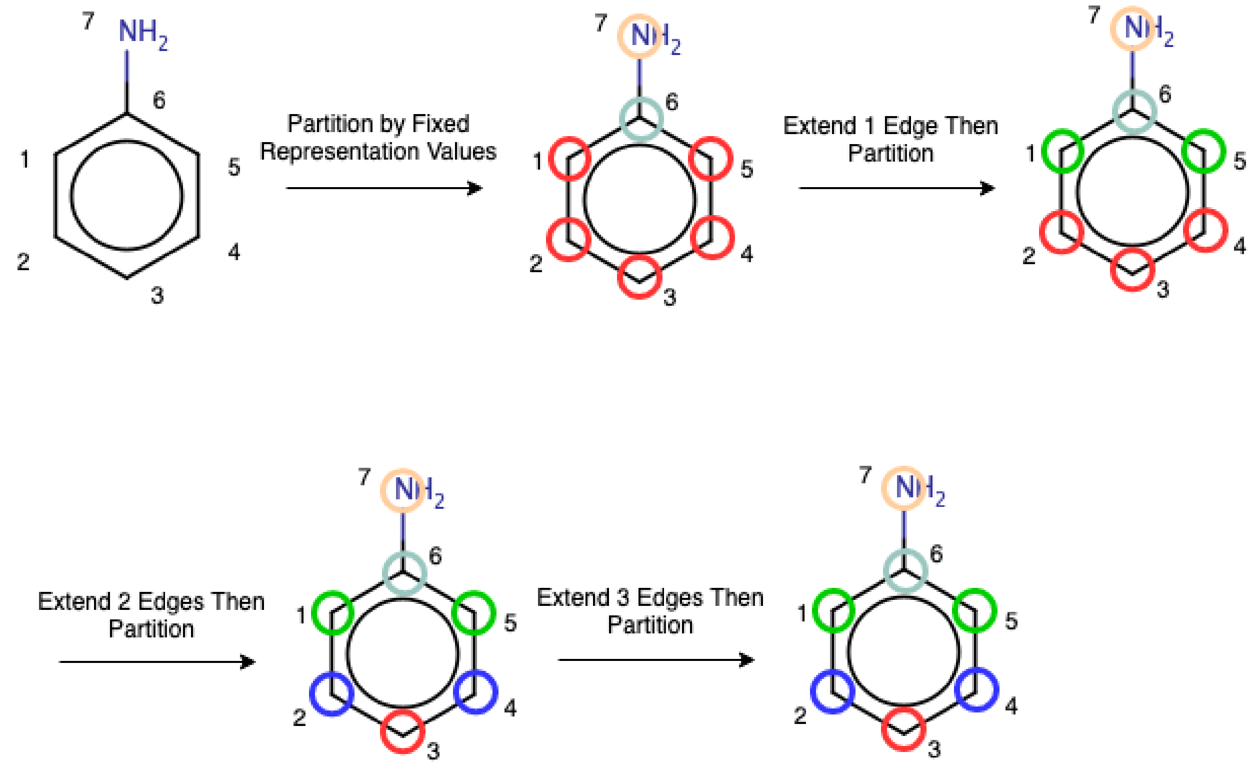
7
NH2
6
1
5
2
4
3
Partition by Fixed Representation Values
Extend 1 Edge Then Partition
Extend 2 Edges Then Partition
Extend 3 Edges Then Partition

Figure 4: The GIVP process for aniline

Canonical Numbering generation by Automorphism Permutation (CNAP)

The CNAP process is more intuitive as a brutal-force method comparing to the GIVP process. It follows the definition of graph automorphism, and investigates all possible permutations of vertices.

Investigation of Automorphic Permutations

Before introducing the definition of automorphism, we have to explain what a permutation is at first. A permutation P can be represented in a two-row notation as follows:[1]

$$\begin{pmatrix} 1 & 2 & 3 & \cdots & i & \cdots & N \\ p_1 & p_2 & p_3 & \cdots & p_i & \cdots & p_N \end{pmatrix}$$

This is a permutation with the meaning that, vertex 1 is permuted to vertex p_1, vertex 2 is permuted to vertex p_2, and vertex i is permuted to vertex p_i etc. A permutation can be considered as an order change of vertex indexes. However for the case when each vertex is permuted to itself, it's not viewed as a permutation.

Now, it's easier to explain what automorphism is based on the definition of a permutation. For a graph $G = G(V, E)$ where V is the vertex set and E is the edge set, G is called automorphic if the adjacency of vertices is preserved after a permutation of vertices is applied. Expressing the automorphic condition in a mathematical way will be, for any $e_{ij} \in E$, then $e_{p_ip_j} \in E$. The CNAP algorithm is composed of repeated investigations on whether each permutation is automorphic or not. If one permutation is proved to be automorphic, the permuted two vertices are considered to be symmetric.

The CNAP Workflow

From the way how a vertex is represented, it can be inferred that if two vertices are constitutionally symmetric, they will be represented as the same value. Conversely two vertices with different representation values will definitely not be symmetric. Therefore, the constitutionally symmetric orbits that we are looking for at the end of the CNAP process will be a subset of the partitioned orbits from the GIVP process. To express this statement in a mathematical way, given the partitioned result from the GIVP process, a set of orbits $\{O_i\}_{partition}$, and the constitutionally symmetric orbits $\{O_j\}_{symmetry}$, for any O_k from the set of symmetric orbits $\{O_j\}_{symmetry}$, there exists an $O_l \in \{O_i\}_{partition}$ that all the vertices in Orbit O_k can be found in Orbit O_l.

Based on this conclusion, it's only necessary to investigate all the permutations from the set of orbits $\{O_i\}_{partition}$ instead of those permutations from all the vertices. Precisely speaking, there will be $\prod k_i! - 1$ permutations to be investigated where k_i is the number of vertex in Orbit O_i from the orbit set $\{O_i\}_{partition}$.

Algorithm 3: The CNAP Algorithm

```
Input: A set of orbits generated by the GIVP process OrbitsGIVP
Output: The set of constitutionally symmetric orbits of the graph OrbitsCNAP
generate all the permutations from OrbitsGIVP;
for one permutation do
    if automorphic condition is fulfilled then
        save the permuted vertex pair as symmetric vertices into OrbitsCNAP
    end
end
return OrbitsCNAP
```

Graph Reduction

The number of permutations to be investigated from the CNAP process is a product of factorials, therefore the amout of computation explodes when there are quite a lot of symmetric vertices. The computational capability of hardware can be surpassed easily. To make the CNAP process computable, A novel graph reduction algorithm based on mapping equal subgraphs is proposed in this paper. It mimics the way how human brains perceive a sophisticated graph: starting from a global view, then reducing the topologically equal subgraphs into single vertices and finally calculating the results from the simplified new graph.

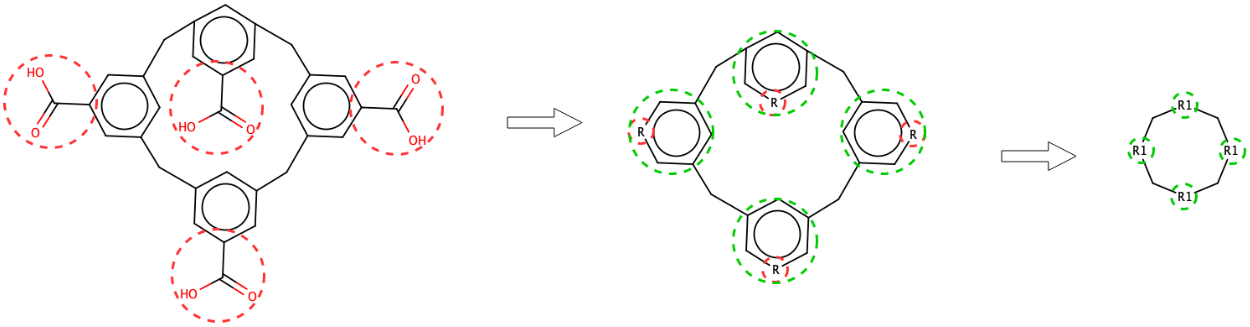

Figure 5: Reducing a sophisticated graph in two steps

The reduction process is composed of two reducing strategies, the breakable case and the cyclic case. The reduction process goes recursively until the simplified graph becomes

CNAP computable, or none of the reducing strategies is applicable anymore.

The partitioned orbits and the numbering from the GIVP process play an indispensable role during the reduction process. It provides valuable guidance not only on where to find the reducible subgraphs but also on the constructing process of mapping among the vertices from the reducible candidates.

To start one process of graph reduction, at first we need to find out a set of vertices to start the mapping process. We name this set of vertices as "the starting orbit". The starting orbit is selected from the GIVP orbits and a mapping among the neighbour vertices is created according to the GIVP numbering. The mapping process continues further with new neighbour vertices until terminating conditions are met. The breakable strategy and the cyclic strategy both have their own terminating conditions respectively.

For CNAP incomputable and irreducible case, the graph will be processed tentatively further as specific cases: highly symmetric ones or separable ones. If the graph remains to be unsolvable by the algorithms designed for specific cases, it will be considered as failure of the CNAP process.

The Breakable Case

The breakable strategy is defined on a concept of "breakable neighbours". Intuitively a vertex becomes a breakable neighbour of a target vertex in a situation that when the edge between the target vertex and this neighbour vertex is removed, no route can be found to traverse from the neighbour vertex to the target vertex. For instance, the first reducing step in Figure 5 is a breakable case according to this definition.

In the graph reduction process of our algorithm, since we are trying to find a set of vertices to proceed the mapping process without vertex conflicts, the definition of breakable neighbour can be generalized more widely. That is to say, the breakable neighbour of a target vertex is the neighbour vertex from which there is no route to traverse to the other vertex in the starting orbit except via the target vertex. Here below Figure 6 is an example that taking the nitrogen atoms as the starting orbit, the oxygen atoms will meet the generalized definition of breakable neighbour instead of the intuitive one, thus the breakable strategy is nevertheless applicable in this case.

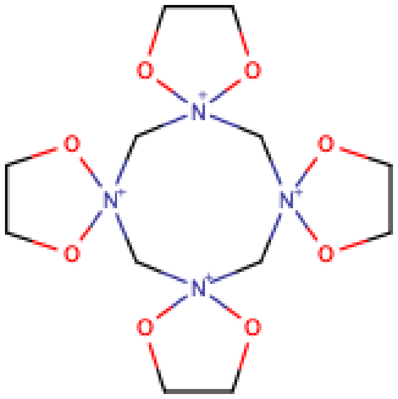

Figure 6: The oxygen atoms are breakable neighbours of the nitrogen atom.

The starting orbit is selected from the GIVP results with the conditions that the vertices should have breakable neighbours. The mapping process stops when there are no new neighbours remaining as the mapping candidates.

The Cyclic Case

The objective of the cyclic strategy is to establish a mapping among small cycles in the graph, as shown in the example from Figure 7. The maximum size of cycles to be identified is a configurable parameter depending on the type of application. For organic molecules, 5, or 6, or 7-atom rings are commonly seen from molecular data, therefore a maximum cycle size of 8 is used.

The cyclic strategy is applied in a moderate way that not all the vertices in the cycle shall be thrown into the mapping process. Only the vertex with two edges will be reduced while the others shared by two identified cycles shall not. Otherwise, there could be an unexpected situation that the graph is over reduced. For instance two edges occur between two vertices, as a phenomenon that shakes the fundamental of the graph definition. Figure 8 shows an

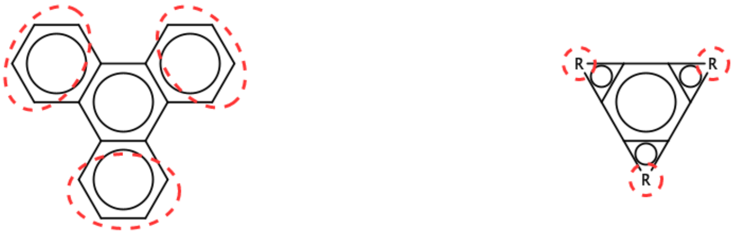

Figure 7: Graph reduction with the cyclic strategy

example when an ambitious strategy is applied with the 3-edge vertices 3, 4, 7 13 reduced, the whole graph will be simplified into two vertices with two edges between them, which is obviously not a valid graph any more.

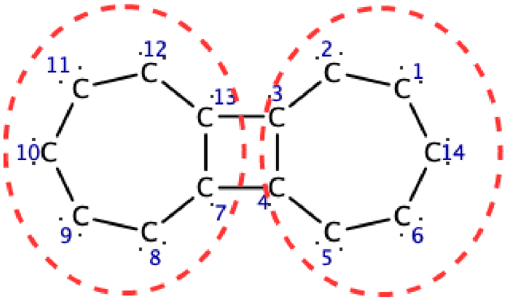

Figure 8: An ambitious cyclic strategy

The starting orbit is selected from the GIVP results with the conditions that any vertex in the orbit should be contained in an identified cycle, and the identified cycles shall not overlap each other. The mapping process is limited to those vertices in the identified cycles with a degree of 2. When all these 2-degree vertices have been mapped, the mappging process will stop and quit.

The Mapping Process

The mapping process starts from the vertices in the starting orbit and continues for the neighbours of those vertices. Those neighbours that have been mapped will be removed from the list of mapping candidates. A mapping between two vertices can be establish only if their numbering values from the GIVP process are equal. The GIVP results decrease the neighour mapping permutations, consequently the amount of computation for mapping.

The Reducible Graph

If the breakable strategy or the cyclic strategy can be applied, the graph is called reducible. When the mapping process of the graph reduction completes, the reducible graph will be reduced to two simpler graphs, as shown in Figure 9. The one who represents the simplified form of the original graph is the "reduced" graph. The other one who represents one of the repeated subgraphs to be hidden during the reduction process is the "folded" graph.

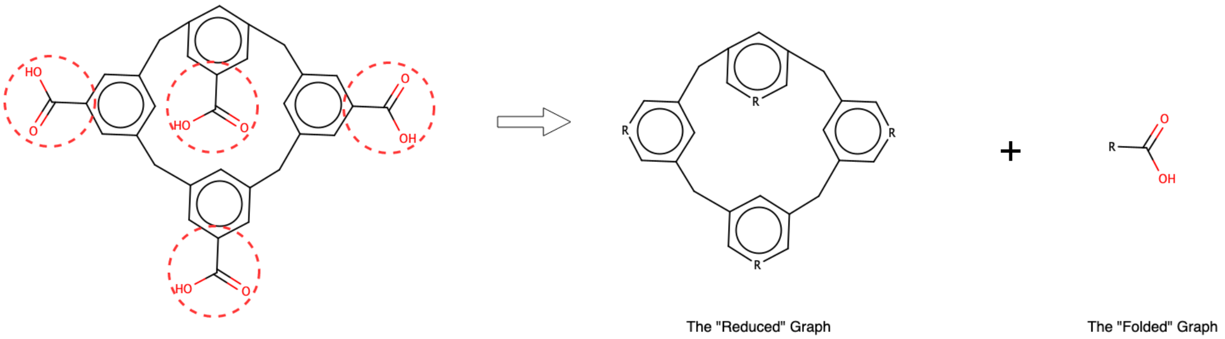

Figure 9: The reduced graph and the folded graph after graph reduction

Irreducible Graphs

When none of the reducing strategies can be applied and the CNAP process remains being incomputable, the graph becomes an irreducible one. The CNAP process for irreducible graphs can proceed further by considering the graph to be separable or to be highly symmetric. When both of the dedicated algorithms for separable graphs and highly symmetric graphs fail, we consider it to be a final failure for the CNAP process.

Separable Graphs

A separable graph is a graph in which the symmetry happens in local areas independently meanwhile the CNAP process is incomputable when all the symmetric orbits mix and generate more permutations to be investigated. In this case the graph could be splitted into multiple graphs for carrying on the CNAP process respectively, the amount of computation will be decreased and the symmetric orbits from each CNAP process will be consolidated together at the end.

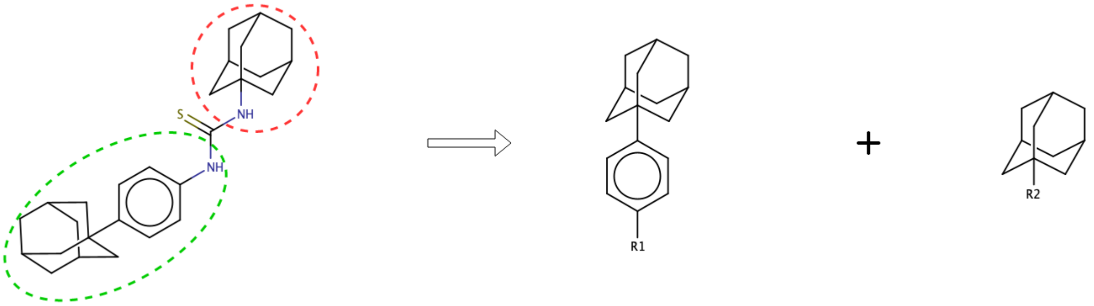

Figure 10: A separable graph for the CNAP process

Figure 10 shows an example of a separable graph where the red part and the green part will be treated separately with a new CNAP process for each of them.

Highly Symmetric Graphs

A highly symmetric graph is a graph with few symmetric orbits meanwhile a large number of vertices in each symmetric orbit. Neither the breakable strategy nor the cyclic strategy can be applied to reduce the highly symmetric graphs into simpler ones. They are not separable too with no localized symmetry. We can list quite a lot of highly symmetric molecules such as benzene, cubane, fullerene, etc. The algorithm for highly symmetric graphs will be last tentative measure of the CNAP process.

Weininger et al.[3] had used a method named "breaking ties" in their unique SMILES algorithm when there are constitutionally symmetric atoms in the molecule. The method marks one of the symmetric atoms, making it to be different from the others, and then the numbering for all the atoms of the molecule will change entirely. We follow this idea to design a "symmetry breaking" algorithm for highly symmetric graphs. When the a graph becomes less symmetric after breaking symmetry by marking a vertex or an edge, the new graph will become much simpler for the CNAP process. The final results of symmetric orbits come from the consolidation of the CNAP results from breaking each vertex and each edge respectively. Since it's possible that the symmetry will completely disappear after symmetry breaking, the CNAP process for highly symmetric graphs is considered as successful only if the resulted symmetric orbits are totally equal to the partitioned orbits from the GIVP process.

Algorithm 4: The symmetry breaking algorithm for highly symmetric graphs

```
Input: A set of orbits generated by the GIVP process OrbitsGIVP
Output: The set of constitutionally symmetric orbits of the graph OrbitsCNAP
for each vertex from any orbit of OrbitsGIVP do
    mark this vertex and generate a new graph;
    carry on the GIVP process and the CNAP process for this new graph to get the
      symmetric orbits
end
for each edge of the graph do
    mark this edge and generate a new graph;
    carry on the GIVP process and the CNAP process for this new graph to get the
      symmetric orbits
end
Consolidate all the symmetric orbits from each loop into OrbitsCNAP;
if OrbitsCNAP equals to OrbitsGIVP then
    return OrbitsCNAP
else
    the CNAP process fails for this highly symmetric graph
end
```

The Canonical Numbering Process with Graph Reduction

Right now all the elements for an improved canonical numbering algorithm have been described. It's time to assemble them together in a recursive way in order to create a more robust algorithm with graph reduction.

Algorithm 5: The canonical numbering process with graph reduction

```
Input: A graph with properties for each vertex and topological connectivities
Output: The set of constitutionally symmetric orbits of the graph OrbitsCNAP
        and the Numbering for each vertex
run the GIVP process and get a set of orbits OrbitsGIVP generated by the GIVP
  process and the Numbering;
if OrbitsGIVP can be solved by the CNAP process without graph reduction then
    run the CNAP process without graph reduction and consolidate the results into
      OrbitsCNAP;
else
    if the breakable strategy or the cyclic strategy can be applied then
        apply the strategy, get a reduced graph and a folded graph;
        run the CNAP process with graph reduction recursively on the reduced
          graph and consolidate the results into OrbitsCNAP;
        run the CNAP process with graph reduction recursively on the folded graph
          and consolidate the results into OrbitsCNAP;
    else
        if the graph can be solved as a separable graph then
            split the graph accordingly into subgraphs;
            for a splited subgraph do
                run the CNAP process with graph reduction recursively on this graph
                  and consolidate the results into OrbitsCNAP;
            end
        else
            consider the graph as a highly symmetric one, run the symmetry breaking
              algorithm and consolidate the results into OrbitsCNAP;
        end
    end
end
return OrbitsCNAP and the Numbering
```

Implementation

All the algorithms and the entire workflow have been implemented in Rust[9] with the open-sourced codes available[12] The modern features of Rust for programming facilitates the work of designing, coding and testing a complicated set of algorithms presented in this paper. Thanks to its feature of generics, the implementation is splited into two major parts, the core module and the extension module. The core module is purely graph related algorithms, meanwhile the extension module is application-oriented. This kind of architecture is designed for the intention of applying the methodology not only to chemical graphs but also other types of graph applications.

For applications of chemical graphs, an extension module of molecule has been created in our implementation, where parsing SMILES strings relies on a third-party open-sourced library named Purr.[13]

Experiments and Results

We run the implementation on an Apple 8-Core M1 chip with 8GB memory for molecules from the database ChEMBL. The maximum number of permutations to be handled directly by the CNAP process without graph reduction is set as 8192. The maximum cycle size for the cyclic strategy is set as 8.

Molecules with more than 250 atoms are not taken into consideration in order to save time with limited computational power. 2,608 (0.13%) large molecules were removed from 2,066,311 molecules in total.

The experimental results for the remaining 2,063,703 molecules are summarized in the Table 2 below. For 442,866 (21.46%) molecules, each atom is differentiated from the others and stays in a sole orbit by the GIVP process, in which case the CNAP process is not necessary any more. For 1,620,577 (78.53%) molecules, the results from the CNAP process matches the results from the GIVP process. At last, 260 (0.01%) molecules failed for the

CNAP process since the two results do not match. Adding more dedicated algorithms for specific graphs together with separable graphs and highly symmetric graphs could be a potential solution to tackle these failure cases.

Table 2: The experimental results of the ChEMBL database

Total	2,063,703	100%
CNAP not necessary	442,866	21.46%
GIVP matches CNAP	1,620,577	78.53%
GIVP does NOT match CNAP	260	0.01%

The durations for the GIVP process and the CNAP process are recorded for each molecule respectively and drawn versus the counts of the heavy atoms as in Figure 11 and Figure 12. Most of the GIVP processes complete within 1 second, and the average duration is around 10 milliseconds. Meanwhile most of the CNAP processes complete within 1 second too. However, the duration difference for molecules with the same total number of heavy atoms is enlarged. This is reasonable that the CNAP process is not only related to how many atoms there are, but also to how these atoms are connected in the graph.

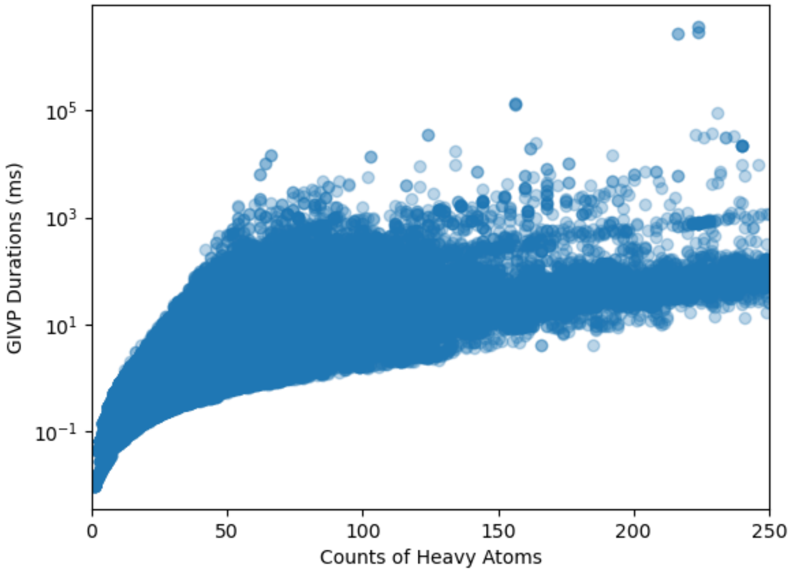

Figure 11: The GIVP durations

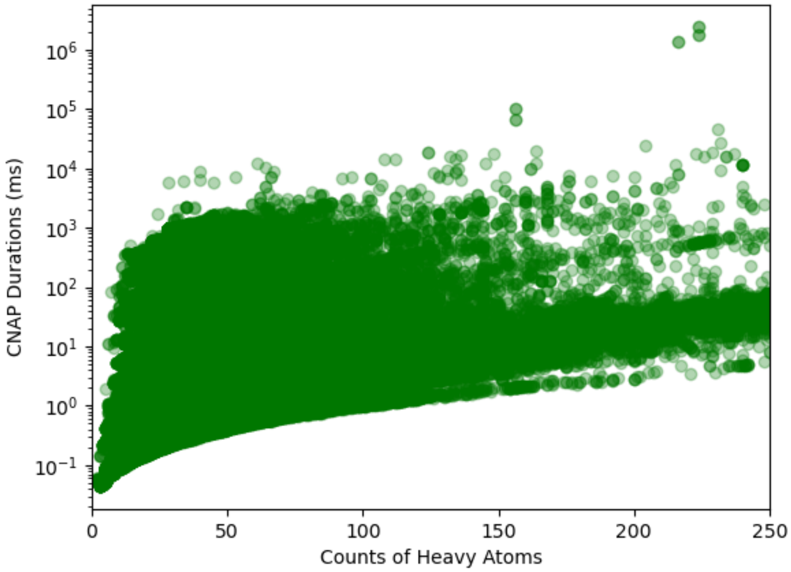

Figure 12: The CNAP durations

Furthermore, the GIVP and CNAP algorithms list in this paper are effective especially for sophisticated molecules with few symmetric atoms. For instance, the molecule with the ID CHEMBL415840 was considered as a failure case from a graph relaxation algorithm.[4] However, canonical numbering for this molecule with only two symmetric atoms can be solved in milliseconds by our algorithms. Another molecule with ID CHEMBL2348759 from the failure cases of the same reference is meanwhile a typical showcase of the breakable strategy of graph reduction. The results of the canonical numbering for these two molecules are shown in Figure 13.

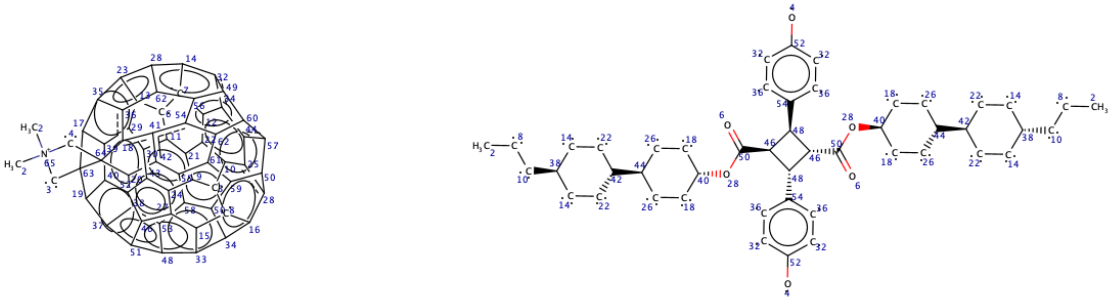

Figure 13: The canonical numberings for CHEMBL415840 and CHEMBL2348759

Future Works

Chirality of molecule for chemical graph application is not taken into consideration in this paper. We are of the opinion that chirality is a 3D spacial phenomenon and it's better to solve the chirality issue with additional application-oriented algorithms on top of our methodology.

To make a parallel version of the algorithms is another interesting field to be explored. For a graph with a large amount of vertices, parallelizing the GIVP process could save the execution time and boost the whole workflow to be faster.

The methodology cannot guaranty that the CNAP process will complete with successful results for each case, therefore to reduce the failure cases is an important topic too. The advantage of the algorithms presented in this paper is that how the failure happens could be traced and analyzed in an easy way, so that it's possible to design specific algorithms targeting at those failure cases.

Conclusion

A workable theoretically complete solution for canonical numbering is presented in this paper. It is composed of two part: GIVP and CNAP. The GIVP part calculates the canonical numbering and the CNAP part proves the symmetry suggested by the GIVP part. According to the experimental results on the ChEMBL database, the results from the GIVP is reliable and might be used solely depending on the requirements of the applications. The graph reduction in the CNAP part provides a perspective of multiple-scale representation on how to perceive a graph, thus leading to a computable solution for hardwares with limited computational capability. The methodology is modularized, traceable and extendable and we hope it could grow to be a fundamental framework for the issue of canonical numbering and symmetry perception in graph applications.

References

(1) Ivaniciuc, O. In *Handbook of Cheminformatics*; Gasteiger, J., Ed.; Wiley-VCH: Weinheim, Germany, 2003; Chapter 12, pp 139–160.

(2) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures-A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation* **1965**, *5*, 107–113.

(3) Weininger, D.; Weininger, A.; Weininger, J. L. SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences* **1989**, *29*, 97–101.

(4) Schneider, N.; Sayle, R. A.; Landrum, G. A. Get Your Atoms in Order—An Open-Source Implementation of a Novel and Robust Molecular Canonicalization Algorithm. *Journal of Chemical Information and Modeling* **2015**, *55*, 2111–2120, PMID: 26441310.

(5) Randić, M. On Unique Numbering of Atoms and Unique Codes for Molecular Graphs. *Journal of Chemical Information and Computer Sciences* **1975**, *15*, 105–108.

(6) Jochum, C.; Gasteiger, J. Canonical Numbering and Constitutional Symmetry. *Journal of Chemical Information and Computer Sciences* **1977**, *17*, 113–117.

(7) Carhart, R. E. Erroneous Claims Concerning the Perception of Topological Symmetry. *Journal of Chemical Information and Computer Sciences* **1978**, *18*, 108–110.

(8) O'Boyle, N.; Sayle, R. Making a hash of it: the advantage of selectively leaving out structural information. https://www.nextmovesoftware.com/talks/OBoyle_MolHash_ACS_201908.pdf.

(9) Why scientists are turning to Rust. 2000; https://www.nature.com/articles/d41586-020-03382-2.

(10) Database ChEMBL.

(11) Hoare, A. Algorithm 64: Quicksort. *Communications of the ACM* **1961**, *4*.

(12) `https://github.com/chiral-data/rust-canonical-numbering`.

(13) Purr: Primitives for reading and writing the SMILES language in Rust. `https://github.com/rapodaca/purr`.