Naming Polyhedra by General Face-Spirals - Theory and Applications to Fullerenes and other Polyhedral Molecules

Lukas N. Wirz, Peter Schwerdtfeger & James E. Avery

To cite this article: Lukas N. Wirz, Peter Schwerdtfeger & James E. Avery (2017): Naming Polyhedra by General Face-Spirals - Theory and Applications to Fullerenes and other Polyhedral Molecules, Fullerenes, Nanotubes and Carbon Nanostructures, DOI: 10.1080/1536383X.2017.1388231

To link to this article: https://doi.org/10.1080/1536383X.2017.1388231

Accepted author version posted online: 06 Dec 2017.

Article views: 21

View related articles

View Crossmark data
Naming Polyhedra by General Face-Spirals { Theory and Applications to Fullerenes and other Polyhedral Molecules

<table>
<thead>
<tr>
<th>Journal:</th>
<th><em>Fullerenes, Nanotubes and Carbon Nanostructures</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript ID:</td>
<td>FNCN17169.R1</td>
</tr>
<tr>
<td>Manuscript Type:</td>
<td>Original Article</td>
</tr>
<tr>
<td>Date Submitted by the Author:</td>
<td>29-Sep-2017</td>
</tr>
<tr>
<td>Complete List of Authors:</td>
<td>Schwerdtfeger, Peter; New Zealand Institute for Advanced Study, Wirz, Lukas; University of Oslo, Department of Chemistry Avery, James; Kobenhavns Universitet Niels Bohr Institutet, Niels Bohr Institute</td>
</tr>
<tr>
<td>Keywords:</td>
<td>Molecular Polyhedra, General Face Spiral, IUPAC Rules, Fullerenes</td>
</tr>
</tbody>
</table>
Naming Polyhedra by General Face-Spirals – Theory and Applications to Fullerenes
and other Polyhedral Molecules

Lukas N. Wirz,1,2,† Peter Schwödtfeger,2,‡ and James E. Avery3,4,§

1Hylleraas Centre for Quantum Molecular Sciences, Department of
Chemistry, University of Oslo, P.O. Box 1033 Blindern, 0315 Oslo, Norway
2Centre for Theoretical Chemistry and Physics, The New Zealand Institute for Advanced
Study, Massey University Auckland, Private Bag 102904, 0745 Auckland, New Zealand
3Niels Bohr Institute, University of Copenhagen (NBI), 2100 Copenhagen, Denmark
4Dep. of Computer Science, University of Copenhagen (DIKU), 2100 Copenhagen, Denmark

We present a general face-spiral algorithm for cubic polyhedral graphs (including fullerenes and
fulleroids), and extend it to the full class of all polyhedral graphs by way of the leapfrog transform.
This yields compact canonical representations of polyhedra with a simple and intuitive geometrical
interpretation, well suited for use by both computers and humans. Based on the algorithm, we
suggest a unique, unambiguous, and simple notation for canonical naming of polyhedral graphs, up
to automorphism, from which the graph is easily reconstructed. From this, we propose a practical
nomenclature for all polyhedral molecules, and an especially compact form for the special class
of fullerenes. A unique numbering of vertices is obtained as a byproduct of the spiral algorithm.
This is required to denote modifications of the parent cage in IUPAC naming schemes. Similarly,
the symmetry group of the molecule can be found together with the canonical general spiral at
negligible cost. The algorithm is fully compatible with the classical spiral algorithm developed by
Manolopoulos for fullerenes, i.e., classical spirals are accepted as input, and spiralable graphs lead
to identical output. We prove that the algorithm is correct and complete.
The worst case runtime complexity is $O(N(N + J))$ for general $N$-vertex polyhedral graphs, with
$J$ the sum of all jump lengths. When the number of faces of any particular size is bounded by a
constant, such as the case for fullerenes, this reduces to $O(N + J)$. We have calculated canonical
general spirals for all 1,943,623,681 fullerene isomers from $C_{20}$ to $C_{200}$, as well as for all fullerene
graphs that require jumps up to $C_{400}$. Further, we have calculated canonical general spirals for large
fullerenes with few or no classical spirals: the non-spiralable chiral $T$-$C_{380}$, $D_{33}$-$C_{484}$, $D_{33}$-$C_{440}$, and
$D_{33}$-$C_{572}$ fullerenes, and all their Goldberg Coxeter transforms up to $C_{50,000}$, and GC transforms
of assorted fullerenes with no pentagon spiral starts. We verify exhaustively that the algorithm is
linear for all the 2.7 × 10^{12} fullerene isomers up to $C_{400}$, and show that this holds also for 11,413
large GC-transform fullerenes up to $C_{50,000}$. On the used hardware, each single general spiral took
about $N \times 200\mu s$ to produce for a $C_N$ fullerene, and the canonical general spiral was found in
$N \times 22\mu s$–32$\mu s$. Hence, we claim the algorithm to be efficient even for very large polyhedra.
The algorithm is implemented in our program package Fullerene. In addition, the source code for a
reference implementation of our proposed nomenclature for polyhedral molecules can be downloaded
from http://erda.ku.dk/vgrid/Polyhedra/spirals/.

* Dedicated to Sir Harry Kroto in Memoriam
†Electronic address: l.n.wirz@kjemi.uio.no
‡Electronic address: p.a.schwoedtfeger@massey.ac.nz
§Electronic address: avery@nbi.ku.dk
We devise an extension of the classical face spiral algorithm and rigorously prove its correctness for all polyhedral graphs, including the molecular graphs of fullerenes and fulleroids. Based on this algorithm, we suggest a unique, short, and constructive notation for polyhedral graphs, as well as a practical nomenclature for fullerenes and other molecular cages including a numbering scheme for vertices.
I. INTRODUCTION

Classification of polyhedra has a long history.\cite{1, 2} For regular polyhedra, the Schläfli symbol (or extensions to it) is commonly used,\cite{3} but this covers only a tiny subclass of all polyhedra. By Steinitz theorem, the vertices and edges of polyhedra form 3-connected planar graphs, which are exactly those planar graphs that have well-defined faces. Conversely, every 3-connected planar graph forms the vertices and edges of a unique polyhedron. Hence, the 3-connected planar graphs are exactly the graphs of the polyhedra, and are named the polyhedral graphs as a class.

If one is not interested in the three-dimensional embedding but only in the connectivity, a polyhedron is uniquely described by its polyhedral graph, and vice versa: fully, if the graph is oriented, and up to chirality if orientation is neglected.

In 1991, Manolopoulos, May, and Down introduced an ingenious way to characterize fullerenes up to automorphism, i.e., up to renumbering the atoms.\cite{4} Fullerene graphs are cubic polyhedral graphs (i.e., each vertex has 3 neighbors) with only hexagon and pentagon faces. They found that unwinding the faces in a tight spiral, while writing down whether each face was a pentagon or a hexagon, yielded a representation of the fullerene graph up to automorphism. This could be made unique by unwinding the spiral in each of $6N$ possible ways, and canonically choosing the lexicographically smallest one. Conversely, given a valid spiral string represented as a sequence $\{n_i\}$ of face sizes, a simple algorithm reconstructed the graph.

Manolopoulos' spiral representations soon became very popular. Firstly, because they gave unique, canonical representations of fullerenes of any size using only 12 numbers, the pentagon indices in the spiral; and secondly because of their simple and intuitive geometrical interpretation: One merely unravels the polyhedron face-by-face like peeling an orange while writing down the face sizes; and to construct the polyhedron from the spiral, one rolls up the "orange peel" of hexagons and pentagons, as seen in Figure 1. These intuitive operations can be formulated as simple constructions on oriented graphs, yielding fast, and robust computer algorithms. The lexicographically smallest spiral found in this way for the $6N$ possible spiral-starts is a unique, canonical representation of the oriented fullerene graph up to graph automorphism. Hence, due to the compact representation, the ease of finding it, and the intuitive geometrical interpretation, it provided an excellent way of naming fullerenes, and it was soon extended for use on other polyhedral molecules.

However, Manolopoulos' (tempting) conjecture that every fullerene graph can be unwound into a face spiral\cite{5} turned out to be incorrect, the first counterexample being $T_d-C_{380}$-fullerene shown in Figure 2c.\cite{5, 6} Indeed, all of the important classes of convex polyhedral graphs contain ones that do not admit any face spirals, i.e., for which all $6N$ spiral starts fail. It is conjectured that the class of non-face-spiral fullerene graphs is infinite, but exceedingly small compared to the spiralable fullerene graphs.\cite{7} However, non-spiralable polyhedra are expected to be more common outside this class (in the remainder of this paper, the terms ‘spirable’ and ‘non-spirable’ are to be understood with respect to the classical face spiral algorithm\cite{5}).

A. Existing approaches for encoding graphs up to automorphism

In order to address the problem of non-spiralable graphs, both Brinkmann\cite{9} and Fowler et al.\cite{10} suggested a general face-spiral scheme, believed but not formally proved to work for all cubic polyhedral graphs. However, there has not been any improvement for the past decades, and generalized spirals have mostly fallen out of use. Brinkmann, McKay, Goedgebeur, and coworkers now employ a scheme for canonical labeling of connected planar graphs based on
Figure 2: The smallest polyhedral graphs without face spirals for (a) cubic planar graphs in general (omni-truncated prism \(G_3\) [3, 6, 8], number of vertices \(N = 18\), the different \(n\)-gons found in the polyhedron are indicated by the numbers in square brackets); (b) cubic planar graphs with \(n\)-gons up to \(n \leq 6\) (\(G_3\) [3, 6], number of vertices \(N = 36\)); (c) for fullerene graphs (\(G_5\) [5, 6], \(N = 380\)). See ref.[8] for details.

vertex coloring [11], which has better computational properties: it works for all connected planar graphs; a canonical labeling of a general planar graph \((V, E)\) can always be found in time \(O(|V| + |E|^2)\); and canonical labeling of several special graph types such as fullerenes can be done in time proportional to \((|V| + |E|)\). However, while this method is very computationally efficient (and excellent for its purpose of automorphism elimination in the \texttt{plantri} and \texttt{buckygen} programs for generating cubic and fullerene graphs), the representation is not designed for naming polyhedra or polyhedral molecules, nor is it appropriate. The result is a long list of \(|V| + |E|\) numbers that have no easily seen meaning: It is neither compact, human readable, nor has an intuitive geometric interpretation. For a large fullerene it becomes impractical, for example it encodes a \(C_{20,000}\) by a string of about 50,000 integers, while a spiral pentagon-index string, if it exists, still comprises only 12 numbers indicating the pentagons positions.

At the opposite end with respect to weighing computation against clarity, the Lederberg-Coxeter-Frucht (LCF) notation is a concise and frequently used notation for Hamiltonian 3-regular graphs. Given a Hamiltonian cycle \(\gamma\), for each vertex \(v\), two of its neighbors are connected along \(\gamma\). The LCF notation consists of a length-\(N\) integer list \(k = [k_1, \ldots, k_N]\), where \(N\) is the number of vertices in the graph \(G\). \(k_i\) denotes the remaining neighbor not connected to \(v_i\) along the Hamiltonian cycle backbone, and the value of \(k_i\) is the distance along the cycle of this neighbor \(k_i > 0\) for clockwise, and \(k_i < 0\) for anti- (or counter-) clockwise. The LCF notation unambiguously defines a cubic graph, and algorithms are available to construct a graph drawing directly from the list.[12] However, the representation is not unique (the same graph can be encoded by many different LCF representations depending on which Hamilton cycle is chosen and depending on the vertex numbering). Figure 3 shows such a Hamilton cycle for the fullerene \(C_{20}\) and the vertices along the Hamilton cycle arranged on a circle to illustrate the LCF notation.

The LCF notation has the advantage that the notation is simple, and applicable to both planar and non-planar graphs that are Hamiltonian. Moreover, it is straightforward to determine planarity by placing the edges into two disjoint edge sets \(A\) (edges that are inside the circle) and \(B\) (edges that are outside the circle) as shown in Figure 3c. This subdivision is possible for every cubic planar graph with a Hamilton cycle.[14] The LCF notation has, however, the disadvantage that the graph needs to be Hamiltonian. Determining whether a cubic planar graph has a Hamiltonian cycle at all is an NP-complete problem; and to find a canonical representation, all Hamiltonian cycles must be considered, a much harder problem still.

## B. Existing approaches for naming fullerenes

Currently, there are no official systematic methods for naming general polyhedral molecular cages, but a number of schemes have been in use for fullerene nomenclature. The general IUPAC naming scheme for organic molecules is to name the parent hydride, define a numbering scheme for all atoms, and to specify the location (locant) of any substitution using the number of the atom at which the substitution takes place (substitutive nomenclature). If the parent hydride is unsaturated and does not contain hydrogen atoms at desired locants as is the case for all fullerenes, the substitutive nomenclature is not directly applicable. Instead, the recommended name is formed by nominally adding a hydrogen atom and substituting it by the desired group.[15] Any general nomenclature scheme for fullerenes

URL: http://mc.manuscriptcentral.com/lfnn Email: fncneditorialoffice@gmail.com
must address two tasks: the parent hydride must be defined uniquely, and a numbering scheme for all carbon atoms must be provided.

Fullerene cages could be named as bridged and fused ring systems or as polycyclic parent hydrides, two strategies that are used for organic molecules with multiple rings. The latter requires to identify the longest ring in a structure such that the bridges have the lowest possible indices. Translated to graph theory this means that all Hamilton cycles in the molecular graph have to be found (if any exist) and compared, a problem that is at least as hard as counting such that the bridges have the lowest possible indices. Furthermore, both naming strategies lead to impractically long names (\(I_{150}\)-fullerene named as a polycyclic parent hydride takes several lines to print) as already pointed out by Goodson et al. To solve this problem, both the International Union for Pure and Applied Chemistry (IUPAC) and the chemical abstract service (CAS) have devised schemes that are tailored to fullerene cages. The preferred IUPAC nomenclature contains the number of carbon atoms and the ideal point group of the cage. In a preliminary survey, ambiguity between isomers with equal point group symmetry is lifted by the index in the list of isomers with a given point group as tabulated in the Atlas of Fullerenes. This strategy requires a list of all isomers with a given point group symmetry, and must fail for structures that do not admit a spiral. In a newer version of the recommendations for preferred names, additionally the set of face sizes and the ‘relative arrangement of rings’ is contained in the parent hydride name, but it is not clear how the ambiguity between the large numbers of fullerene isomers with low symmetry should be avoided: For example, there are 14,029,812 \(C_{1-C_{150}}\) fullerences, and 20,597 \(C_{2-C_{150}}\) and for larger structures the numbers become so large due to the \(O(N^3)\) scaling, that even computing the lists and storing them on a computer becomes infeasible. CAS uses a comparable naming scheme with the same characteristics as the IUPAC scheme, but in a different format. They keep a catalogue of individual molecules that have been discussed in the literature at some point, and assign a unique CAS registry number to each structure.

For the first experimentally relevant fullerenes, atom numbering schemes were tabulated that either took into account chemical reactivity, symmetry elements, or that followed vertex spirals. Numbering the atoms following a vertex spiral is a poor naming scheme, because many fullerenes (let alone other molecular cages) do not admit vertex spirals. In order to handle any fullerene cage, a highly intricate set of rules was devised. While these rules define a unique numbering, they seem to be unnecessarily complicated and error prone.

### C. The present work

In Section II, we introduce a new generalized spiral algorithm, and provide a mathematical proof that it works on all cubic polyhedral graphs (Appendix VI). The algorithm is compatible with the classical face spiral algorithm of Manolopoulos and Fowler in that it yields the same result for graphs that admit a classical spiral. In addition, the spiral algorithm is extended to all polyhedra by applying an injective leapfrog mapping from the polyhedral graphs...
into the cubic polyhedral graphs. By ensuring that we can uniquely invert the leapfrog transformation, we show that we can reconstruct any polyhedral graph \( G \) from the spiral of its leapfrog \( LF(G) \). This allows us to extend the general spiral scheme to the set of all polyhedral graphs. Since the leapfrog always yields a graph three times the size, the runtime complexity is unchanged. We also show how the automorphism group of a polyhedral graph, cubic or not, can be found at the same time as computing the canonical general spiral at essentially no extra cost.

Using our spiral algorithm, we introduce in Section III a unique way to name any fullerene in linear time, without the exponential blowup from the requirement of finding all Hamilton cycles in the IUPAC scheme for polycyclic molecules. We believe this fills an important gap in the currently used nomenclature: Contrary to existing schemes, the naming scheme is complete (all fullerenes can be named consistently and unambiguously), unique (the bond graph is easily reconstructed from the name), and compact (regardless of molecule size, the name consists of around 12 numbers). A similar scheme is introduced for naming polyhedral molecules in general, which uses the leapfrogged spiral representation of general polyhedral graphs described above. We believe a systematic method for naming polyhedral molecules is currently entirely missing, and hope that the method we present will be adopted. The method scales quadratically in the worst case, but in most cases becomes linear due to at least one face size occurring only a bounded number of times. The name of a polyhedral molecule is in general linear in size in the number of faces, but a compactification scheme using run-lengths usually reduces the length of the name dramatically.

In Section IV we show examples of finding canonical general spirals and naming a number of non-spiralable polyhedra and fullerenes, as well as several of their Goldberg-Coxeter transforms, going through the examples in detail. In addition, we show general spiral names for a series of recently experimentally realized autoassembled non-cubic polyhedral molecules \([23]\) by their leapfrogged spirals.

Finally, in Section V, we analyze the runtime complexity of the general face spiral algorithm, as well as two canonical labeling schemes: A compatibility canonical spiral, computed in time \( O(N(N + J)) \) (where \( J \) is the total jump length needed), which always yields the same result as the classical canonical spiral when a classical spiral exists, and the canonical general spiral, which is \( O(N + J) \) for fullerenes and all other classes of polyhedra that have at least one face size that appears a bounded number of times. While we do not yet have a proof, we conjecture that \( O(N + J) = O(N) \) for fullerenes and similar classes. This is verified for all of the 2,653,606,256,199 fullerene graphs up to \( C_{400} \), as the largest number of required jumps is 4, and the largest value of \( J \) is 35. We benchmark our implementation by finding canonical spirals for all fullerene graphs up to \( C_{200} \), all graphs that require a jump up to \( C_{400} \), and for 11,413 large Goldberg Coxeter transforms (up to \( C_{50,000} \)) of fullerenes that admit either few or no spirals. The linear behavior was found to hold also for the large fullerenes, scaling as \( t_{\text{spiral}} \approx N \times 22\mu s \sim N \times 32\mu s \), and as an example, we generate a \( C_{95,000} \) isomer through the GC transform and find its canonical spiral, all in under three seconds.

II. THE GENERAL FACE-SPiral ALGORITHM

The classical face-spiral algorithm by Manolopoulos offers a concise description of a fullerene graph.[5, 19] It is, however, not applicable to describing every cubic polyhedral graph, nor even fullerene graphs as correctly pointed out by Manolopoulos and Fowler.[5, 6, 19] For example, in fullerenes that do not admit any face spiral, such as \( T-C_{380} \)-fullerenes shown in Figure 2c, all spiral attempts end in a cul-de-sac of faces.[5] We call these non-spiralable, or "T"-fullerenes. In the present section we discuss a general vertex-spiral algorithm similar to that of Brinkmann [9] and Fowler et al.[10] that operates on triangulations of the sphere, which are exactly the duals of cubic polyhedral graphs. Through the dual operation, this yields a generalized face spiral for any cubic polyhedral graph as well as any triangulation of the sphere. Rather than restarting after spirals get stuck, the algorithm maintains a connectedness-invariant (checked in constant-time for fullerenes and similar classes) during the spiral unwinding, and inserts jumps whenever the invariant is broken. The connectedness-invariant breaks exactly when the spiral will eventually end in a cul-de-sac. The spiral winding and unwinding algorithms are simple, intuitive, easily visualized, and can be performed by hand with pen and paper.

We define two types of canonical spirals, i.e. uniquely defined representations, associated with a polyhedral graph: the canonical general spiral, which can be computed very efficiently but reduces to the Manolopoulos spiral in the case of fullerenes only when a spiral starting in a pentagon exists; and the compatibility canonical spiral, which takes longer time to compute, but always reduces to the classical spiral when any such spiral exists. These are defined for cubic polyhedral graphs and their duals, the triangulations of the sphere, and extended to all polyhedral graphs in Subsection II G. This gives us a rapid way to compute a unique representation of any polyhedral graph up to isomorphism, in the same way that the Manolopoulos canonical spiral is a unique representation of a spiralable fullerene. For fullerenes and similar classes of polyhedra, the canonical general spiral is computed in time \( O(N) \) in the number of vertices, \( N \), an order faster than the classical method. It provides a classification of polyhedral graphs up to isomorphism (and also for checking whether two such graphs are isomorphic); and as we describe further down,
a rapid way to compute a compact permutation-representation of the symmetry group.

There are other fast canonical representations; the spirals have the advantage of being simple to understand visually (“like peeling an orange”), and of being easy to implement. For fullerenes, they have approximately constant size (regardless of graph size), and the general spirals with jumps approximate this. A completeness proof, which proves the generalized spiral algorithm to work for all 3-connected cubic graphs, is given in the appendix. We believe we are the first to provide a correctness proof to go with the algorithm.

Before detailing the algorithm, we briefly review the Manolopoulos face-spiral algorithm, and introduce the **boundary code** used in our implementation of both methods.

### A. The boundary code

In the following, let $G$ be a planar cubic graph and $T = G^*$ its dual, i.e., the triangulation that has a vertex of degree $n$ for each $n$-gon, connected if they are adjacent in $G$. For a vertex-induced subgraph $P$ of $T$, which includes all the edges from $T$ connecting the subset of vertices in $P$, we define the **boundary** $\partial P$ of $P$ to be the set of vertices that are connected to a vertex in $T$ that is not contained in $P$, i.e., those that have lower degree in $P$ than in $T$. If $P$ and the graph induced by the vertex set $V_T \setminus V_P$ are both connected, we can traverse all vertices on the boundary along a path.

Given such a path, we define its **boundary code** as the string of the differences between the $T$-degree and $P$-degree of traversed vertices on that path, i.e., the string $[c_i]$ with $c_i = \deg_T(v_i) - \deg_P(v_i)$, see Figure 4. A given subgraph can in general be described by different boundary codes, corresponding to the different possible starting points of the path around its boundary. Our definition of the boundary code is purely graph theoretical, but it leads to the same boundary code as defined geometrically by Fowler et al. [10]

As an example, consider a subgraph $H$ of $T$ with three mutually connected vertices, and where each vertex has degree 6 in $T$, see Figure 4a. It will then be described by the boundary code $[4, 4, 4]$, because all three vertices have four free valencies. Adding another vertex to the subgraph as shown in Figure 4b creates a subgraph that is described by either the code $[3, 4, 3, 4]$ or $[4, 3, 4, 3]$ depending on the starting point, i.e., corresponding to the (directed) paths $(v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4)$ and $(v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_1)$ respectively.

### B. The Manolopoulos face-spiral algorithm

Before introducing the general spiral algorithm for all cubic polyhedral graphs, we describe a simple spiral algorithm equivalent to Fowler and Manolopoulos' classical algorithm,[19] which works only for polyhedra that can be unwound into spirals without jumps. This algorithm forms the basis of the general one for cubic polyhedra described in the next section.

Let us define two subgraphs $P$ and $R$ ("Processed" and "Remaining") of $T = G^*$, where the vertex set $V_P$ of $P$ is a subset of $V_T$, and $V_R = V_T \setminus V_P$. $P$ and $R$ are the subgraphs of $T$ induced by the described vertex sets. We maintain a path $w = (v_1 \rightarrow v_2 \rightarrow \ldots v_n)$ around the vertices $v_i$ of the boundary $\partial P$ and its corresponding boundary code $c = [c_1, \ldots, c_n]$ defined by the vertices $v_i$ as illustrated in Figure 4. Note that $\partial P$ only contains vertices with free valencies. The resulting spiral $\sigma$ is a string consisting of the degrees in $T$ (and finally in $P$ when the algorithm ends) of the vertices in the order in which they are placed from the start to the end.

We begin by choosing a path around a triangle, e.g., $w = (v_1 \rightarrow v_2 \rightarrow v_3)$, and the corresponding set $V_P = \{v_1, v_2, v_3\}$ with $V_R$ being the set of the remaining vertices in $V_T$. Then the following procedure is repeated until all vertices have been placed. For each consecutive step, choose the next vertex to be placed: the common neighbor...
failing spirals, whereas the very rare
some face to face 15, and as a result the last (outer) face cannot be reached. It is common for fullerene graphs to have face 15, both common neighbors of face 15 and face 10 have been placed already, i.e., there is no vacant neighboring -gons (n

The above described spiral algorithm naturally leads to the question of whether it is applicable to all planar graphs or at least to all fullerene graphs as conjectured at an early stage.[5] For general cubic polyhedral graphs, the first counterexample is a polyhedron derived from the trigonal prism by truncation at all vertices, thus containing 6 triangles, 2 hexagons and 3 octagons shown in Figure 2a.[8, 24] Fowler et al. conjectured that every infinite class of cubic polyhedra characterized by its n-gons contains non-face-spiral elements.[8] Furthermore, truncation at all vertices (omni-truncation) of cubic polyhedra with more than 4 vertices become non-spiralable,[38] making this class of cubic polyhedra infinite in size, as has been proven by Brinkmann and Fowler.[25]

To get a concrete understanding of how face spirals may fail, Figure 5 depicts two attempts to unwind D2-C28-fullerene into a spiral.[19] In Figure 5a the resulting face-spiral pentagon indices [19] are [1, 2, 3, 4, 5, 7, 10, 12, 13, 14, 15, 16], or equivalently, the spiral code is [5, 5, 5, 5, 6, 6, 5, 5, 5, 5, 5, 5] listing the n-gons (n = 5 or 6) along the spiral. The attempt to construct a spiral in Figure 5b fails: After the addition of face 15, both common neighbors of face 15 and face 10 have been placed already, i.e., there is no vacant neighboring face to face 15, and as a result the last (outer) face cannot be reached. It is common for fullerene graphs to have some failing spirals, whereas the very rare NS-fullerenes, such as T-C380-fullerene as shown in Figure 2c, have only failing spirals. We are not aware of investigations into how common non-spiralable polyhedra are in general outside the class of fullerenes.

C. Why do classical face-spirals fail?

URL: http://mc.manuscriptcentral.com/lfnn  Email: fncneditorialoffice@gmail.com

Figure 5: Two attempts to construct a spiral in D2-C28-fullerene: (a) succeeds while (b) misses face 16. In case (a) the spiral code is [5, 5, 5, 5, 6, 6, 5, 6, 5, 5, 5, 5, 5, 5] (see ref.[19] for details).
More generally, in each step of the construction of a spiral, one face that has not been added to the spiral yet and which is adjacent to the face that has been added in the previous step, is appended to the spiral. For this procedure to reach the last face, it is necessary for all remaining faces to form one connected patch. However, if the remaining faces are disconnected into two or more face patches, the spiral may only cover the remaining faces in one of them and finally gets stuck before reaching the last face.

It is important to note, that the fatal step is not the one in which the spiral gets stuck (e.g., step 15 in Figure 5b), but the step in which the patch of remaining faces gets disconnected (step 14 in the same diagram). The important insight into designing a fast general face-spiral algorithm that works for all cubic planar graphs is to efficiently detect this before it happens and altogether avoid going down any path that leads to a cul-de-sac and a failed spiral. This is achieved by detecting cut-vertices, i.e., when a step would disconnect the remaining graph, and including jumps such that it remains connected throughout. Section II F describes how this can be done in time $O(F)$ with $F$ being the size of the largest face, and hence $O(1)$ for fullerenes and most other graph classes that we will encounter.

Although there is no reliable indicator available for predicting whether a fullerene graph admits any spirals, the known counterexamples suggest (and it is graphically understandable) that large fullerene graphs with high curvature at three or four centers are good candidates for being non-spiralable. The first examples of non-spiralable fullerene graphs have high symmetry (e.g., $T$ or $D_3$), however, this may be an artefact of the smallest examples and needs not be a rule.

D. The general face-spiral algorithm: Encoding

The general vertex spiral algorithm for triangulations we propose is an extension of the previously described spiral algorithm. As in the previous section, we start from a planar cubic graph $G$ and work on its dual, the triangulation $T$. The vertex-induced subgraphs $P$ and $R$, as well as the path $w$ on the boundary $\partial P$ and the boundary code $c$, are all defined as described above. In addition, however, we introduce *jumps* in order to deal with graphs with failing spirals only, i.e., we define a *jump of length $k$* as a cyclic shift of the path $w$ such that $w = (v_1, \ldots, v_n)$ becomes $(v_{k+1}, \ldots, v_n, v_1, \ldots, v_k)$ (see Fig. 6). Each step of the cyclic shift implies that the first element of $w$ is moved from the first to the last position in $w$; this procedure is repeated $k$ times. As we will see, in cases where a spiral would otherwise get stuck, we can always choose a jump that prevents this. Appendix VI provides a formal proof that the algorithm always succeeds, i.e. every spiral start yields a general spiral representation, for all 3-connected planar graphs.

As a concrete illustration of this, consider the failing face spiral of the $D_2C_{28}$-fullerene shown in Figure 7. If we go all the way to the cul-de-sac where the spiral gets stuck, the boundary is $\partial P = [v_{10}, v_{11}, v_{12}, v_{13}, v_{15}]$, while $v_{14}$ has been added to the spiral last. A jump of length 1 leads to the same boundary, but in this case $v_{15}$ has been added last and, therefore, the spiral can be completed by adding the last vertex $v_{16}$ which is adjacent to $v_{15}$ but not to $v_{14}$. From this, we define a **general spiral code** as a pair consisting of a *jump list* and a spiral code. The jump list is a possibly empty list of pairs $(n, k)$ of jump positions $n$, specifying at which step in the algorithm the jump is taken, and jump lengths $k$. If the jump list is empty, this specifies a classical spiral. Completely analogously to the case for classical spirals, we define for fullerene graphs the **general pentagon indices** as the general spiral code, but where the spiral code is replaced by the twelve pentagon indices.

The general spiral algorithm determines a general spiral code for any planar cubic graph in the following manner. Like in the previously introduced algorithm, the spiral generation is initiated by choosing a path $w = (v_1, v_2, v_3)$

![Diagram](diagram.png)

Figure 6: (a) Default candidate for $v$ as neighbor of $v_1$ and $v_n$: $w = (v_1, \ldots, v_n)$; (b) Jump of length 1: $w = (v_2, \ldots, v_n, v_1)$; (c) Jump of length $k$: $w = (v_{k+1}, \ldots, v_n, v_1, \ldots, v_k)$
around a triangle in $T$, initializing the boundary code, setting $V_P = \{v_1, v_2, v_3\}$, and $V_R$ to all the remaining vertices.

We then proceed to place the remaining vertices in $V_P$ one by one following essentially the same rules as before, with one exception: The next vertex $v$ to be placed is chosen as before, but prior to placing it, we first check whether removing $v$ from $R$ disconnects the vertex-induced subgraph $R$. If, and only if, this is the case, then placing $v$ will lead to the spiral eventually reaching a dead end. In order to complete the spiral, we skip $v$ and perform a jump to $v'$ such that removing (the newly determined) $v'$ from $R$ leaves $R$ connected. The jump length $k$ is chosen to be the smallest that does not disconnect $R$. The test ensuring that $R$ never becomes disconnected can be performed in constant time given certain conditions, and is described in a subsection below. The complete algorithm is given in Algorithm 1.

E. The general face-spiral algorithm: Decoding

Next we describe how to construct a cubic graph from a general spiral code, i.e., the inverse operation to the encoding described above. The windup of the spiral code yields an oriented triangulation of the sphere (also called a deltahedron); the cubic polyhedral graph is then obtained by taking its dual. The general procedure is as follows.

Given is the general spiral code, i.e., a possibly empty list of jumps $(n, k)$ together with a spiral string $\sigma$ of length $N$. During the construction of the triangulation, edges are added to a subgraph $P$. $P$ is represented by the neighbor lists of each vertex, sorted in clockwise order. As each new edge is added, this order is kept. An edge $u \rightarrow v$ is added to $P$ by inserting $u$ and $v$ into each other’s neighbor lists at the appropriate position to preserve the graph orientation as detailed in Algorithm 2. We maintain a path $w$ on the boundary of $P$ its boundary code $c$. In more detail:

The first edge is $v_1 \rightarrow v_2$, and the boundary path is initialized to $w = [v_1, v_2]$. The free valencies of $v_1, v_2$ are obtained from $\sigma$ and stored in $c$, after subtracting 1 from each of them as they are now part of one edge.

Each subsequent vertex $v_3, \ldots, v_{N-1}$ is placed in the following way: First, we check whether the first element $(n, k)$ in the jump list applies to this vertex. If $v_i = n$, a cyclic shift by $k$ is performed on $w$ and $c$, and $(n, k)$ is removed from the jump list. An edge between $v_i$ and the first vertex in $w$, as well as between $v_i$ and the last vertex in $w$, is added to $P$. The first and last element of $c$ are decremented to account for the new edges.

If the first element of $c$ becomes zero, the first element is removed from $w$ and $c$, and we connect $v_i$ and the new first neighbor to $w$. This process is repeated until the number of free valencies of the first vertex in $w$ remains non-zero after decrementing. Analogously, elements are removed from the end of $w$ and $c$, and edges between $v_i$ and the last vertex in $w$ are added to $P$. After connecting $v_i$ to vertices in $P$, $v_i$ is appended to $w$. The valency of $v_i$, which is stored in $\sigma$, is appended to $c$ after subtracting the number of newly created edges.

This procedure is repeated until all vertices except the last have been added to $P$. The final vertex, $v_N$, is connected to all the remaining vertices of $w$. Now, $P$ is a complete triangulation of the sphere, $w$ and $c$ are empty. The cubic graph may finally be obtained by taking the dual of the triangulation $P$. The detailed algorithm is given in Algorithm 2.
Algorithm 1: Algorithm for encoding a planar triangulation as a general spiral string. For simplicity it was assumed that \( v_1, v_2 \) and \( v_3 \) are in clockwise order in \( T \); for counter-clockwise orientation the algorithm works analogously.

```
: Input: triangulation (T) with a planar layout, first three vertices \( v_1, v_2 \) and \( v_3 \)
initialize triangulation (TW) as working copy of T
initialize spiral (S) with valencies of \( v_1, v_2 \) and \( v_3 \) in T
initialize empty jump list (J)
for \( v_4 \leq v_N \leq v_N - 1 \) do
  JS ← 0 // jump state
  while \( TW \setminus \{v[\text{last}], w[\text{first}]\} \) is disconnected do // jump required
    OV ← OV[1+1, ... , last, first]
    w ← w[1+1, ... , last, first]
    JS ← JS+1
  end
  if JS > 0 then // jump was performed
    append \((v, JS)\) to J
  end
  remove \( v_1 \) from TW
  append \( v_i \) to w
  append degree of \( v_i \) in T to OV
  append degree of \( v_i \) in T to S
  OV[1] ← OV[1]-1
  OV[last] ← OV[last]-1
  OV[last-1] ← OV[last-1]-1
  while OV[1] = 0 do // first vertex has become interior
    pop OV[1]
    pop w[1]
    OV[1] ← OV[1]-1
    OV[last] ← OV[last]-1
  end
  while OV[last-1] = 0 do // second to last vertex has become interior
    pop OV[last-1]
    pop w[last-1]
    OV[last-1] ← OV[last-1]-1
  end
append degree of \( v_N \) in T to S
return S, J
```

F. Maintaining connectedness: Cut-vertex detection in constant or \( O(D) \) time

In the algorithm for encoding a graph into a general spiral string, there is a test if the \text{remainder-graph} \( R \) remains connected after removing a vertex \( v \) from it. Naively, testing the connectedness of a graph scales as \( O(N) \). However, in the special case of a connected planar graph in which at most one face is larger than a triangle and where the degree of each vertex is bounded (both conditions are always fulfilled by \( R \)) this test can be performed in the following way.

The bounded vertex degree in \( R \) corresponds to a bounded face size in the underlying cubic polyhedral graph. Consider a connected planar graph \( T \) which has at most one face larger than a triangle and a bounded vertex degree. \( v \) is a vertex in \( T \) (not required to be adjacent to the large face) which will be removed. \( u_1, \ldots, u_n \) is the list of all neighbors of \( v \) in \( T \). In order to test whether \( T \) is still connected after removing \( v \) it is necessary and sufficient to test if the graph \( C \) induced by \( u_1, \ldots, u_n \) is connected.

If \( C \) is connected, then \( T \setminus \{v\} \) is connected: Trivially, \( v \) was connecting its neighbors, and if all the neighbors are still connected, \( v \) was not a cut vertex in \( T \).

If \( C \) is disconnected, then \( T \setminus \{v\} \) is disconnected: Here we make use of the condition, that at most one face is larger than a triangle. If \( u_1, v \), and \( \text{next}_{\text{CW}}(u_1, v) \) form a triangle, then \( u_1 \) and \( \text{next}_{\text{CW}}(u_1, v) \) share an edge. Conversely, if \( u_1 \) and \( \text{next}_{\text{CW}}(u_1, v) \) do not share an edge, then \( u_1, v, \text{next}_{\text{CW}}(u_1, v) \) is not a triangle. As there is only one face larger than a triangle in \( T \), all pairs of neighbors \( u_i, \text{next}_{\text{CW}}(u_i, v) \) which do not share an edge are adjacent to the single large face in \( T \). Hence, if the number of pairs of neighbors \( u_i, \text{next}_{\text{CW}}(u_i, v) \), which do not share an edge is \( k \), then removing \( v \) cuts \( C \) as well as \( T \) in \( k-1 \) disconnected parts.
Algorithm 2: Algorithm for constructing a triangulation from a general spiral string. All neighbor lists are oriented consistently.

```
Input: spiral (S) = [s₁, …, sₙ], jump list (J) = [[vᵢ, kᵢ], …, [vⱼ, kⱼ]]
initialize empty boundary (w), open valencies (OV), neighbor list (NL) of every vertex
append s₁ to OV
append s₂ to OV
append v₁ to w
append v₂ to w
for vᵢ ≤ vⱼ ≤ vₙ do
    if J[first].v = vᵢ then
        k ← J[first].k
        w ← w[first+k, …, last, first, …, first+k-1]
        OV ← OV[first+k, …, last, first, …, first+k-1]
        pop J[first]
        p ← 0
        insert vᵢ into NL[w[first]] directly before w[last]
        insert w[first] into NL[vᵢ]
        p ← p+1
        OV[last] ← OV[last]-1
        while OV[first] = 0 do // first vertex has become interior
            n ← w[first]
            pop OV[first]
            pop w[first]
            insert vᵢ into NL[w[first]] directly before n
            insert w[first] into NL[vᵢ] directly before w[last]
            p ← p+1
            OV[first] ← OV[first]-1
        end
        while OV[last] = 0 do // last vertex has become interior
            n ← w[last]
            pop OV[last]
            pop w[last]
            insert vᵢ into NL[w[last]] directly before w[first-1]
            insert w[last] into NL[vᵢ] directly before w[first]
            p ← p+1
            OV[last] ← OV[last]-1
        end
        append sᵢ – p to OV
        append vᵢ to w
    end
for 1 ≤ i ≤ size of w do // connect last vertex
    append w[i] to NL[vᵢ]
    insert vₙ into NL[w[i]] directly before w[i-1]
end
return triangulation defined by NL
```

For all graphs produced by Algorithm 1, i.e. patches of triangulations, we can determine whether a vertex is a cut-vertex in time \(O(D)\), where \(D\) is the maximum vertex degree in the triangulation, i.e. the cost is independent of the total size of the graph. For fullerenes, and most other classes of polyhedra that we are interested in, \(D\) is bounded by a constant, so that the cut-vertex test takes \(O(1)\) time.

In general planar graphs, the cost of determining whether a vertex is a cut-vertex additionally depends on the size of the second largest face \(F₂\) (which previously was a triangle). Then, not only the graph induced by all neighbors of \(v\) needs to be checked for connectedness, but the graph induced by all vertices that have a distance of \(≤ |F₂| - 2\) from \(v\) (first and second neighbors for a quadrilateral, etc).
In order to define a unique inverse of the leapfrog operation, we need to identify all face-faces in $G$. Because polyhedral graphs are 3-vertex-connected, every vertex has at least degree 3, as a vertex can be disconnected by removing its neighbors. Consequently, vertex-faces in $C$ are of size 6 or larger. Furthermore, there are guaranteed to be faces smaller than hexagons in a polyhedral graph, because otherwise the cage could not be closed. Therefore, every face in $C$ that has size 5 or smaller is a face-face, and there are guaranteed to be at least four of them (in the case of four triangles). Identifying one of them defines the full set of face-faces, because each vertex in $G$ corresponds to a face in $C$ with a size of twice the vertex’ degree. The spiral—referring to $G$—of which the start is drawn is denoted as [LF: 8, 4, 6, 5, 6, 6, 5, 6, 6, ...]. The vertices of $G$ are labeled under the assumption that the spiral is the canonical spiral, in the order in which the faces are traversed by the spiral.

G. Extending the method to all polyhedral graphs

The general spiral method can be extended, such that every polyhedral graph can be represented by a general face spiral, regardless of whether the graph is cubic or not.

The leapfrog transformation maps any polyhedral graph $G$ (with $N$ vertices and $F$ faces) onto a cubic polyhedral graph $C = LF(G)$. Visually this can be understood as the omnitruncation of the dual of $G$; equivalently, one produces the dual triangulation of the leapfrog (on which we apply Algorithm 1) directly by adding a vertex at the center of each face and connecting it to each vertex of the face. $C$ has $N + F$ faces, and by drawing $C$ overlayed onto $G$, a mapping is obtained from all faces in $C$ to either a face or a vertex in $G$. Every face that corresponds to a face in $G$ (face-face) inherits its size, whereas faces that correspond to a vertex in $G$ (vertex-face) have a size of twice the degree of that vertex, as is visualized in Figure 8. Having mapped an arbitrary polyhedral graph into its (cubic and 3-connected) leapfrog graph, the proof in Appendix VI guarantees that every spiral start yields a general face spiral representation.

In order to define a unique inverse of the leapfrog operation, we need to identify all face-faces in $G$. Because polyhedral graphs are 3-vertex-connected, every vertex has at least degree 3, as a vertex can be disconnected by removing its neighbors. Consequently, vertex-faces in $C$ are of size 6 or larger. Furthermore, there are guaranteed to be faces smaller than hexagons in a polyhedral graph, because otherwise the cage could not be closed. Therefore, every face in $C$ that has size 5 or smaller is a face-face, and there are guaranteed to be at least four of them (in the case of four triangles). Identifying one of them defines the full set of face-faces, because each vertex in $C$ is adjacent to exactly two vertex-faces and one face-face. $G$ is composed of exactly these faces, any two faces are neighbors in $G$ if they are connected by an edge in $C$, which concludes the inverse leapfrog transformation $LF^{-1}$. We have thus found a bijective mapping between the set of polyhedral graphs and their leapfrog transforms, which are cubic polyhedral graphs. Visually, the inverse leapfrog operation can be understood as shrinking each vertex-face to a single point.

Making use of this bijective mapping, and the previously established general spiral algorithm, we can encode every non-cubic polyhedral graph by the canonical general spiral of its leapfrog transform. Conversely, the initial graph can be constructed by applying $LF^{-1}$. We note that this bijective mapping does not always exist for planar graphs which are not 3-connected or for infinite 3-connected graphs.

We note here that we also obtain a general vertex spiral algorithm that works for all polyhedral graphs by this method. This follows immediately from the fact that the leapfrog maps any polyhedral graph into a cubic polyhedral graph, and from the trivial fact that the dual of a polyhedral graph is also polyhedral. Since the (leapfrogged) general face spiral algorithm succeeds for all polyhedral graphs, applying it to the dual yields general vertex spiral algorithm that works for all polyhedral graphs. In fact, our algorithms work in the triangulation representation, so computing a general vertex spiral requires one less dualization step than a general face spiral.
The fast computation of the canonical general spiral for a cubic polyhedral graph at the same time computes a compact representation of the graph’s automorphism group.

Assume that we are given the fullerene graph dual $G^*$, constructed from a general spiral $S = (d_1, d_2, \ldots, d_F)$, and wish to compute the automorphism group of $G^*$. Since $G^*$ is constructed from $S$, the entries in $S$ correspond to the degrees of vertex number $1, 2, \ldots, F$ in $G^*$. For every vertex $v$ of degree $d_v$, we have $2d_v$ different spiral starts: $d_v$ for clockwise and $d_v$ for counter-clockwise traversal. If a spiral start $(f_1, f_2, f_3)$ unwinds $G^*$ to the input spiral $S$, there is an automorphism of $G^*$ that maps $(1, 2, 3) \mapsto (f_1, f_2, f_3)$. These are all the automorphisms, and the number of starts that unwind to $S$ is the order of the automorphism group (see Ref.[27] for details). For fullerenes, we can use the pentagon index representations of the general spirals to obtain a representation of the automorphism group from only $\approx 12$ integers per group generator, i.e., small near-constant size, and similarly for more general fulleroids by using the representation of non-hexagon indices in the spiral.

This information can be obtained while computing the canonical general spiral: The face sequence corresponding to each spiral unwinding is temporarily stored together with the spiral, and the ones corresponding to the canonical spiral (the lexicographically smallest one) define the group elements by the following permutation representation:

$$\pi_F(g) = (f_1, f_2, f_3, \ldots, f_F)$$  \hspace{1cm} (1)

This is a faithful representation of the group, and if we wish, we can easily build the multiplication table by composing all pairs of the permutations, or calculate characters, irreducible representations, and all other properties; and we can identify the group.

For the method defined by the Manolopoulos spirals, it was necessary to compute spirals for all $6N$ spiral starts in order to compute the symmetry group, since many graphs have no regular spiral starting in a non-hexagon face and would otherwise fail. Since the general spirals always succeed, this is no longer necessary. For fullerenes and other polyhedra with bounded number of negative curvature faces, this reduces the complexity by an order, yielding the symmetry group in linear time (and obtained for free when computing the canonical general spiral), and a near-constant size representation of the group. And, more importantly, it always works.

Because the leapfrog transform preserves symmetry, this yields a rapid method for computing the symmetry group for any polyhedral graph, not just for cubic ones.

We note that there are several algorithms already available to obtain the group automorphism of a general graph.[28–30] However, these are significantly more difficult to understand and implement than the general spirals, and all yield representations of size $O(N)$ or larger, even in cases such as fullerenes, where our general spiral scheme usually produces group representations of size $O(1)$. An advantage is that we can get everything at once: A canonical graph representation, easy to understand nomenclature, canonical vertex numbering, and the symmetry group.

In addition to the abstract symmetry group (fully defined by the multiplication table of the automorphisms), we are often interested in the symmetry 3D point group. Such a (molecular) point group has not just the group multiplication rules, but is the largest set of isometries that leave the polyhedron invariant. The point group has additional structure compared to the abstract symmetry group of graph automorphisms. For example, the point groups $D_2$, $S_4$, $C_{2h}$, and $C_{2v}$ all have the same group structure, namely the Klein 4-group $\mathbb{Z}_2 \times \mathbb{Z}_2$; but they represent different geometrical symmetries.

The point group corresponds to the rotations, reflections, roto-inversions, and inversions that leave the ideal polyhedron invariant. A strong theorem by Mani [31] states that any 3-connected cubic graph can be embedded in space as a convex polyhedron, the unique point group of which realizes the full automorphism group of the graph. That is: every graph automorphism of a cubic polyhedral graph is also a rotation or reflection of its ideal polyhedral shape.

In Schwerdtfeger et al. [27], we detail a scheme for computing the point groups for fullerenes by way of orbits on faces, edges, and vertices, and a decision tree.[39] This is specific for the 28 fullerene point groups, but similar schemes can be derived by hand for larger classes of polyhedra that have a finite set of point groups. A complete and fully automatic method for identifying the 3D point groups of general polyhedral graphs is to our knowledge still an open problem.

I. Exhaustive generation of classes of cubic polyhedra

While a particular polyhedron can be reconstructed in linear time from the generalized spiral representation, it is not recommended to attempt to exhaustively generate (for example) all $C_N$ fullerene isomers by going through all the possible spirals: both because there are many spirals that encode the same polyhedron, and because most syntactically well-formed spiral strings do not encode a polyhedron. For example, it follows from a result by Thurston
that there are $O(N^9)$ fullerene $C_N$ isomers,[32] but it is easy to see that when restricting to 12 pentagon and $N/2-10$

hexagons, there are $\sim \binom{N}{12}$ (i.e. $O(N^{12})$) regular spiral strings, and even higher for general spiral strings.

For the case of fullerenes, an exhaustive list of $C_N$ isomers can be generated extremely efficiently using the Buckygen
algorithm by Brinkmann, Goedgebeur, and McKay,[33] based on the structural induction definition of fullerenes by
Hasheminezhad et al.[34] The authors of Buckygen have graciously allowed the software to be incorporated into the
Fulleren program[35], where it can be used to exhaustively search through many different classes of fullerenes on-
the-fly, for example to find all isomers that have some specified properties. Other classes of cubic polyhedral graphs
can be generated exhaustively using more general methods, using for example the PlanTri program[11].

III. NOMENCLATURE

In the following we introduce a nomenclature for polyhedral graphs based on the spiral algorithms described above,
and derive a nomenclature for polyhedral molecules which we suggest to be adopted by IUPAC. A particularly compact
representation is given for fullerenes and fullerene duals, as well as for fulleroids. Contrary to currently used schemes,
this nomenclature yields unique, unambiguous names from which the graphs can be easily constructed.

a. Cubic polyhedral graphs  Any cubic polyhedral graph with $F$ faces can be denoted by a comma separated list
of jumps consisting of jump positions $n_i$ and jump lengths $k_i$, followed by face sizes $f_i$ in the order in which they
are traversed by the spiral. Jumps and face sizes are separated by a semicolon, and the whole general spiral string
is enclosed by square brackets. Counting of jump positions starts at 1 (not 0). If no jumps are required in a spiral,
the semicolon should be omitted as well. The spiral including the jumps is obtained by the algorithm detailed in
Algorithm 1.

$$[n_1,k_1,\ldots,n_m,k_m;f_1,f_2,\ldots,f_F]$$

We define a general spiral $\sigma_1$ to be smaller than another general spiral $\sigma_2$ if $\sigma_1$ is shorter, that is it requires fewer
jumps. If $\sigma_1$ and $\sigma_2$ are of equal length the order is determined lexicographically. Lexicographical ordering implies
that, like ordering words in a dictionary, the strings are ordered by consecutively comparing their entries starting
from the first, such that the first differing entry determines the relative order of two strings. As jump positions and
lengths are in the front of the string they are compared first, followed by the face sizes.

The canonical general spiral string is defined as the smallest spiral of a given cubic graph (for example a fullerene
isomer) which starts at a face with the rarest non-hexagon face-size (at a pentagon in case of a fullerene graph). In
case of ambiguity, smaller faces have preference. The compatibility canonical spiral string in contrast is defined as
the smallest spiral of a given cubic graph starting at any face. The two types of spiral string are prefixed ‘GS’ for the
canonical general spiral string or ‘CS’ for the compatibility canonical spiral string. Comparing by length first ensures
that spirals with fewest possible jumps are chosen. This ensures that the compatibility canonical spiral is identical to
the classical spiral whenever a classical spiral exists, and makes the compatibility canonical face spiral fully compatible
with the classical spiral algorithm. Restricting the starting face of the canonical general spiral reduces the complexity
of finding it which will be explained in more detail in the complexity analysis of Section V. As a side effect, however,
it will sometimes require more jumps, making it a less compact identifier. If the compatibility canonical spiral string
starts at a non-hexagon face of the rarest type, both canonical spiral types are identical. In this case (for example,
for all except about one in a million fullerenes), the CS/GS-prefix may be omitted.

The general spiral string can be written in a compressed form similar to Fowler et al. [10]. Any repetition of
single faces or sequences of faces may be abbreviated by runlength-encoding: writing the sequence in parentheses and
superscribing the number of repetitions. This short form is unambiguous but not unique as there may be different
options to choose repeating subsequences. As an additional (but not necessary) information the vertex count could
be provided as usually done for polyhedra.

b. Triangulations: Because triangulations and cubic graphs are each others’ duals, a general spiral encodes a
triangulation of the sphere as well as a cubic polyhedral graph.

Given a triangulation, we proceed as described above for cubic graphs. The spiral code is the same as for its cubic
dual, but the name is prefixed with ‘T’:

$$[T:n_1,k_1,\ldots,n_m,k_m;f_1,f_2,\ldots,f_F]$$

In the implementation, we directly apply Algorithm 1 which operates on a triangulation, rather than dualizing twice.

c. Fullerene graphs and their duals: For the special case of fullerene graphs, i.e., cubic polyhedral graphs with
only pentagons and hexagons, a much shorter representation can be given. As there are always exactly 12 pentagons,
a particularly compact representation of the general spiral is given by the positions of the pentagons in the spiral
string. This yields a constant-size representation of those fullerenes that admit classical spirals, the vast majority.[19]
Leaving the jump notation unmodified, we obtain the \textit{general face spiral pentagon indices} from the general spiral string.

\[ [n_1, k_1, \ldots, n_m, k_m; p_1, p_2, \ldots, p_{12}] \]

This notation has the advantage of having almost constant size (assuming that only a small number of jumps are required), while the \textit{general spiral string} grows linearly with the number of vertices in the graph. That is, whether the fullerene comprises 20 or 20,000 vertices, it is still uniquely representable by the 12 pentagon indices, together with any needed jumps. Up to \(C_{600}\), only two of the \(2.7 \times 10^{12}\) fullerenes require a jump, and even when requiring spirals to start in pentagons, jumps are exceedingly rare for fullerenes.

Here, the \textit{canonical general face spiral pentagon indices} and the \textit{compatibility canonical face spiral indices} are defined analogously as above, i.e., as the smallest spiral of a given graph starting either at any face or only at the non-hexagon face of which there are the fewest. The smallest spiral is determined by sorting first by length, and lexicographically in case of equal lengths. Sorting the general spiral string and the derived pentagon indices gives the same order, and hence the same canonical spiral.

For triangulations that are fullerene duals, the pentagon indices are simply used together with the ‘T’-modifier, as described above.

d. Fullerenoids: An analogous short form can be defined, for example, for cubic polyhedral graphs with face sizes four or six or three and six only, which comprises a list of the positions of all four triangles or all eight quadrilaterals in the general spiral string. For example, for a fulleroid comprised of triangles, quadrilaterals, and hexagons, we can write its spiral as

\[ [n_1, k_1, \ldots, n_m, k_m; t_1, \ldots, t_n; q_1, \ldots, q_{m_2}] = (3, 4)_6 \]

using between 4 spiral indices (the case of only triangles) and 6 (only quads); and a negative-curvature fulleroid with pentagons, hexagons, and heptagons, can be written

\[ [n_1, k_1, \ldots, n_m, k_m; p_1, \ldots, p_{m_h}; h_1, \ldots, h_{n_h}] = (5, 7)_6 \]

In the latter case, the number of pentagon indices is determined by the number of heptagons. The general form of the compact naming scheme for fulleroids is

\[ [n_1, k_1, \ldots, n_m, k_m; i_1^1, \ldots, i_m^1; \ldots, i_1^p, \ldots, o_m^p] = (f_1, \ldots, f_t)_6 \]

where \(i_k^j\) denotes the index of the \(k\)th \(f_j\)-gon in the spiral string. Note that this is only a compact naming scheme for fulleroid classes that consist primarily of hexagon faces, i.e., that have limited curvature. For non-cubic graphs, the subscripted “6” can be replaced by a more general base face-size \(F\): For example, the zero-curvature face-size for 4-regular graphs is 4 (360° = 4 \times 90°), and for 6-regular graphs, it is 3.

e. Non-cubic polyhedral graphs: Any polyhedral graph can be characterized through a face spiral of its leapfrog transform, as shown in Section II G. This allows us to lift the nomenclature scheme to the full class of \textit{all} polyhedral graphs. To form the name for a general polyhedral graph, we use the spiral of the leapfrogged graph, from which the graph can be reconstructed up to isomorphism. This operation is denoted by an ‘LF’ prefix:

\[ [\text{LF}; n_1, k_1, \ldots, n_m, k_m; f_1, f_2, \ldots, f_F] \]

Following the same rules that were defined for cubic polyhedral graphs, the canonical general spiral string and the compatibility canonical spiral string are determined. Either uniquely defines the non-cubic polyhedral graph.

\[ [\text{LF}, \text{GS}; n_1, k_1, \ldots, n_m, k_m; f_1, f_2, \ldots, f_F] \]

\[ [\text{LF}, \text{CS}; n_1, k_1, \ldots, n_m, k_m; f_1, f_2, \ldots, f_F] \]

f. Molecular cages with underlying cubic polyhedral graphs: Based on the above described nomenclature for cubic polyhedral graphs, we suggest the following scheme for naming molecular cages that can be represented by a cubic polyhedral graph: The canonical general spiral string (or compatibility spiral string) is given as described above, followed by the chemical formula \((C_N)\) for all-carbon allotropes, \((Au_N)\) for all-gold allotropes, etc.). If the atoms of the molecule are not specified, only the number of atoms in the unmodified cage, \(N\), is given. To denote that the polyhedral cage naming scheme is used, ‘cage’ is specified as the parent hydride name. The ideal point-group of the polyhedral bond graph can be optionally prefixed. While the point-group information is redundant, it is often
convenient for readability. The fully qualified name (unique and unambiguous, from which the bond molecule can be readily constructed) is then:

$$\text{PG-[GS/CS: } n_1, k_1, \ldots, n_m, k_m; f_1, f_2, \ldots, f_T]-X_N\text{-cage}$$

In case of cages with mixed atom types, it is suggested to choose $X$ as the most frequent atom type—substitution of atoms at the remaining positions is denoted using the numbering scheme that is detailed below. ‘Cage’ is the parent hydride name for a closed cage without any added atoms (as for example hydrogen), even if a ‘naked’ cage does not physically exist. The name does not specify the bond orders within the cage.

We recommend using the general spiral naming scheme, which for fullerenes always starts in a pentagon, unless compatibility with Manolopoulos’ original naming scheme is explicitly required. This allows efficient automatic computation, several orders of magnitude faster than the original canonical spiral scheme.

$g$. Deltahe dedral molecular cages: Molecular cages that are triangulations, also called deltahedra, are named identically to their cubic graph duals, with the simple change of adding the $T$-modifier. i.e., a general deltahedral molecule would be written as

$$\text{PG-[T,GS/CS: } n_1, k_1, \ldots, n_m, k_m; p_1, p_2, \ldots, p_{12}]-X_N\text{-fullerene}$$

(see below).

$h$. Molecular cages with underlying non-cubic polyhedral graphs: Molecular cages that have an underlying non-cubic polyhedral graph should be named in the same way as cubic polyhedral cages, but using the general spiral for non-cubic polyhedral graphs.

$$\text{PG-[LF,GS/CS: } n_1, k_1, \ldots, n_m, k_m; f_1, f_2, \ldots, f_N]-X_N\text{-cage}$$

$i$. Fullerene molecules: As discussed above, fullerenes have particularly compact representations of their face spirals, namely by the 12 pentagon indices that denote the positions of the pentagons in the general spiral string, no matter the size of the molecule. Thus fullerenes should be named separately from general polyhedral molecules by using the 12 pentagon indices rather than the full string. This nomenclature is unique, constructive, short, and applicable to every fullerene molecule as proved in the appendix. The fully qualified name for a fullerene molecule is

$$\text{PG-[GS/CS: } n_1, k_1, \ldots, n_m, k_m; p_1, p_2, \ldots, p_{12}]-C_N\text{-fullerene}$$

(3)

where, as above, the point-group prefix is optional. If the fullerene structure is realized by a non-carbon molecule, ‘$C_N$’ is substituted by the appropriate chemical formula.

Note that it is a common practice to denote a fullerene by its point group and chemical formula only (PG-$C_N$-fullerene). This is usually ambiguous, but in some cases for isomers of small fullerenes with high symmetry (e.g., $I_{A_1}$-$C_{60}$-fullerene or $D_{5h}$-$C_{70}$-fullerene), it is not. In this case, it is a very useful shorthand. However, even in the high-symmetry cases for which this notation is unambiguous, it is still recommended to include the fully qualified name, as reconstruction from the shorthand name requires looking it up a database of structures, but the bond graph can be readily reconstructed from the spiral name.

$j$. Fulleroid molecules: General fulleroid molecules (polyhedral with regular polygon faces) are named similarly to the fullerenes, i.e., by their non-hexagon (or more generally, non-$F$) face positions in the spiral:

$$[\text{GS/CS: } n_1, k_1, \ldots, n_m, k_m; i_1^n, \ldots, i_n^n; i_1^p, \ldots, i_n^p]-(f_1, \ldots, f_T)-C_N\text{-fulleroid}$$

(4)

For example, a fulleroid with pentagons, hexagons, and heptagons, would be denoted by the pentagon and heptagon indices in the spiral:

$$[\text{GS/CS: } n_1, k_1, \ldots, n_m, k_m; p_1, \ldots, p_{12}; h_1, \ldots, h_{12}]-(5,7)_6-C_N\text{-fulleroid}$$

$k$. Atom numbering in cubic polyhedral molecules: In order to denote the location of a substituent in a fullerene derivative, or in general a derivative of a polyhedral cage, a scheme for numbering all carbon atoms is required. An obvious choice for such a scheme would be a canonical vertex spiral, but not all fullerene graphs admit a vertex spiral. Instead we suggest to number the vertices (atoms) in a manner that builds on the general face spiral introduced here, which we prove in the appendix always exist. Thus, tying the vertex labels to the face spiral
construction guarantees applicability to all fullerenes, as well to the more general class of molecular cages with underlying cubic polyhedral graphs.

The canonical general face spiral specifies a canonical order of the faces, the vertices in the dual, by the sequence in the spiral. Each vertex in the cubic graph corresponds one-to-one to a triangle in the dual representation, the three vertices of which correspond to the three faces adjacent to it in the cubic graph. Hence, each vertex in the cubic graph is uniquely characterized by this ordered triplet. The index of each vertex is now determined by lexicographically ordering this list of ordered triplets. Indices start counting at ‘1’, as is common in organic chemistry, and end with the number of vertices \( N \). Because no two vertices can be adjacent to the same three faces, this assigns a unique numbering to the vertices, i.e. the atoms; and since the canonical general spiral exists for all cubic graphs, and is unique and unambiguous, so is the canonical atom numbering. To avoid confusion, the numbering should only be based on the canonical general spiral which is much faster to calculate.

As an example, taking the face-spiral of Figure 7, we obtain the following list of 3-tuples and the corresponding vertex numbers: (1,2,3)→1, (1,2,6)→2, (1,3,4)→3, (1,4,5)→4, \ldots .

- **Atom numbering in deltahedral molecular cages:** As molecular cages with exclusively triangular faces are duals of cubic graphs, their general spiral string representations are vertex spirals, and hence trivially induce a canonical naming of the atoms by their positions in the spiral. Hence, for deltahedra, the canonical atom numbering is the vertex position given by the general spiral name. This is handled as follows: Given a graph \( G \) and a set \( M \) of vertices marked for substitution, we compute the canonical vertex numbering together with the canonical general face spiral as described above. Each spiral start may assign the vertices in \( M \) different (symmetry equivalent) numbers; the vertex numbering that yields the lexicographically smallest numbers for \( M \) is the canonical one.

This is handled as follows: Given a graph \( G \) and a set \( M \) of vertices marked for substitution, we compute the canonical vertex numbering together with the canonical general face spiral as described above. Each spiral start may assign the vertices in \( M \) different (symmetry equivalent) numbers; the vertex numbering that yields the lexicographically smallest numbers for \( M \) is the canonical one.

- **Canonical vertex numbering in derivatives** Often the atom numbers are to be used as a basis for deriving more complex molecules. In this case, a set of atom numbers are chosen as the locations at which the parent cage is substituted. This may break the symmetry of the underlying cage, and we must perform a little extra work to make sure the vertex numbering is canonical:

In cages with non-C\(_1\) point group symmetry, each vertex \( v_i \) is equivalent to a set of vertices \( V_i = \{v_{j_1}, \ldots, v_{j_n}\} \) if there is a symmetry operation that transforms \( v_i \) into \( v_{j_k} \). Once we mark a particular set of vertices, they no longer partake in the symmetry of the underlying cage, and we must choose which of the numbers in the equivalence classes to assign them in a well-defined, canonical way.

This is handled as follows: Given a graph \( G \) and a set \( M \) of vertices marked for substitution, we compute the canonical vertex numbering together with the canonical general face spiral as described above. Each spiral start may assign the vertices in \( M \) different (symmetry equivalent) numbers; the vertex numbering that yields the lexicographically smallest numbers for \( M \) is the canonical one.

- **Derivatives:** By using the above described atom numbering schemes, any derivative of a molecular cage can be named according to existing IUPAC conventions.[15, 16] These include the removal or substitution of atoms, insertion of atoms into bonds, and functionalization of the cage. Through insertion of groups into bonds, even molecular cages that are not described by polyhedral graphs but only by planar graphs are accessible to our nomenclature.

**IV. EXAMPLES AND APPLICATIONS**

In this section, we show examples of the general spirals and nomenclature for the different classes of polyhedra and molecules. We note that the examples we give are chosen to be simple for didactic reasons, but small highly symmetric non-cubic polyhedra are the worst case for the use of general spirals (as more compact names often exist). The general spirals are particularly useful mainly due to their generality and completeness, and the fact that the polyhedra can be reconstructed directly from the name. They outperform other naming scheme mostly for larger structures – and easily handle polyhedra with low- or no-symmetry. In the following figures, faces are colored by their number of edges, and the vertices by their neighbor counts: 3 is green, 4 is dark purple, 5 is blue, 6 is orange, 7 is red, 8 is pink, 9 is light purple, and 10 is grey. For example, a green vertex is trivalent, and corresponds to a light purple triangle in the dual; and the orange vertices in Figure 9(c) have six neighbors, and correspond to the hexagons of its dual polyhedron in Figure 9(b).
Figure 9: Simple examples of three cubic polyhedra, and a non-convex deltahedron. Below the polyhedra are shown the polyhedral graphs with their compatibility general spirals, which in these cases coincide with their classical face spirals for the cubic graphs, and vertex spirals for the triangulations.

Figure 10: Examples of non-spiralable cubic polyhedra. Below the polyhedra are shown the polyhedral graphs with their canonical generalized face spirals, which are listed in the text.

A. Small cubic non-spiralable graphs

Figure 9 shows fully qualified general spiral names for a selection of small cubic and triangular polyhedral graphs. The simplest cubic polyhedral graph is a tetrahedron, shown in Figure 9(a). It is a 4-vertex cubic graph with the spiral [3,3,3,3], or [3^4]. While this spiral name is enough to reconstruct the polyhedron, the full molecular
and a Tutte-shaped molecule would be named $M_{12L_{24}}$. The point group $T_d$ and the canonical spiral specifier can be included, whereby the name becomes $T_{d}[\text{GS}]:3^4\text{-}Au_1\text{-cage}$. The omni-truncated tetrahedron in Figure 9(b) is a 12-vertex cubic graph with a single jump and the compatibility spiral name for the polyhedron is $[\text{CS}]:6(3,3)^4$; the canonical general spiral which starts in a triangle requires a jump and has the spiral $[\text{GS}]:6,1,3,6^3,3,2,6,3$. A carbon molecule with this polyhedral bond graph would be named in full $T_{d}[\text{GS}]:3(6,3)^4\text{-}C_{12}\text{-cage}$ (when including its tetrahedral point group symmetry). The third example, Figure 9(c), is a non-convex deltahedron, the triakis tetrahedron, which is the dual of the truncated tetrahedron. Hence its name is the same except for the dual-modifier ‘T’ and the vertex count: $[T,\text{CS}]:6(3,3)^4$ for the polyhedron, and $T_{d}[T,\text{CS}]:6(3,3)^4\text{-}X_{12}\text{-cage}$ for a molecule with this structure, where $X$ is replaced by the base atomic symbol. The omni-truncated octahedron in Figure 9(d), consisting of squares and hexagons, has the canonical general spiral name $[4,6^3,(6,4)^3]$ (for both the CS and GS scheme), and a molecule with this structure would be named $O_{h}[4,6^3,(6,4)^3]\text{-}X_{24}\text{-cage}$.

Figure 10 shows small polyhedra that do not admit classical spirals. The smallest non-spiralable graph with faces up to hexagons (Figure 10(a)) requires a single jump: The canonical general spiral name for the polyhedron is $[\text{GS}]:18,2,3,6^1,3,6,(6,3)^2,6^2$, and a molecule would be $T_{d}[\text{GS}]:18,2,3,6^1,3,6,(6,3)^2,6^2\text{-}X_{36}\text{-cage}$. The omni-truncated trigonal prism (see Figure 10(b)) requires one jump, and the canonical general spiral name for a molecule with this structure is $D_{ih}[\text{GS}]:9,3,(8,3,6,3)^2,3,8,3\text{-}X_{18}\text{-cage}$.

The famous Tutte graph [36] (Fig. 10(c)), which is non-Hamiltonian, requires a minimum of two jumps, and has the compatibility canonical spiral string

$$[\text{CS}]:11,1,17,15,10,5,5,5,5,9,5,4,5,4,4,5,4,10,5,5,5,5,10,4,5,5,4,5]$$ (5)

and a Tutte-shaped molecule would be named $C_{5}[\text{CS}]:11,1,17,1;5,10,5^3,9,(5,4)^2,4,5,4,10,5^5,10,4,5^2,4,5]\text{-}X_{46}\text{-cage}$

### B. Non-cubic polyhedral molecules

To show how the general spiral method works for all polyhedral graphs through the leapfrog transform, we here apply it to a class of non-cubic polyhedra, described in Fujita et al. [23]. Fujita et al. were able to realize these molecules experimentally by self-assembly of $n$ palladium ions and 2n bent bidentate organic ligands (written as $M_nL_{2n}$). These cages can be abstracted by polyhedra where every vertex represents a palladium atom and every edge a bidentate ligand. Figure 11 shows these tetravalent polyhedral structures (above) and the canonical generalized face spirals (below). Our program calculates the dual of the leapfrog, on which the spiral algorithm is performed directly instead of first calculating the leapfrog and then dualizing; this operation simply consists of adding a new vertex in the center of each face, connected to each of the vertices of the face. Table I shows the full canonical names of the structures, from which the graphs and polyhedra are easily reconstructed.

### C. Fullerenes

Figure 12 shows the canonical general spiral for a $T_d$-$C_{100}$-fullerene, the smallest fullerene without a classical spiral that starts with a pentagon, and the only one among the 285,914 $C_{100}$-fullerenes to require a jump in the canonical general spiral. The canonical general spiral name, starting with a pentagon, is $[\text{GS}]:43,2,1,4,5,26,27,31,32,40,43,47,48,52$, where the jump ‘43,2’ indicates that before adding the 43rd face, a jump of length 2 is performed. The compatibility spiral, $[\text{CS}]:2,8,9,23,24,28,29,37,41,45,46,52$, contains no jumps, but requires an order longer to compute as it starts in a hexagon. However, for molecules this small, the extra order of computation for the compatibility spiral is not an issue unless one wishes to calculate canonical names for very many graphs, and the slightly shorter name may...
We now apply the general spiral naming scheme to fullerenes that do not admit classical spirals. These are too large for the spiral superimposed on the bond graph to be legible, so we only list the spiral names. The graphs and polyhedra can be constructed, and chemical information calculated, by giving the spiral strings as input to Fullerene ([35]):
The smallest non-spiralable fullerene, a $T$-$C_{380}$-fullerene:

$[GS:]$ 162, 2, 186, 3; 1, 4, 5, 126, 127, 136, 137, 155, 162, 171, 172, 186$-C_{380}$-fullerene
$[CS:]$ 110, 2; 45, 70, 71, 82, 83, 110, 119, 120, 144, 184, 185, 192$-C_{380}$-fullerene

The second smallest non-spiralable fullerene, a $D_3$-$C_{384}$-fullerene:

$[GS:]$ 171, 8, 178, 9; 1, 3, 4, 5, 168, 169, 170, 178, 190, 191, 192, 194$-C_{384}$-fullerene
$[CS:]$ 49, 1; 29, 30, 31, 49, 145, 146, 170, 171, 190, 191, 192, 194$-C_{384}$-fullerene

A non-spiralable $D_3$-$C_{440}$-fullerene:

$[GS:]$ 198, 9, 205, 10; 1, 3, 4, 5, 195, 196, 197, 205, 218, 219, 220, 222$-C_{440}$-fullerene
$[CS:]$ 62, 1; 39, 40, 41, 62, 170, 171, 197, 198, 218, 219, 220, 222$-C_{440}$-fullerene

A non-spiralable $D_3$-$C_{672}$-fullerene, which is the halma transform of $D_3$-$C_{168}$-fullerene:

$[GS:]$ 297, 8, 310, 9; 1, 10, 12, 14, 260, 262, 264, 310, 324, 326, 328, 338$-C_{672}$-fullerene
$[CS:]$ 142, 1; 51, 53, 109, 111, 220, 252, 288, 302, 304, 306, 320, 328, 338$-C_{672}$-fullerene

## D. Goldberg-Coxeter transforms of non-spiralable fullerenes

In one of our previous papers [7] we introduced the halma (GC$_{k,0}$) and leapfrog (GC$_{1,1}$) transforms of four well known non-spiralable fullerenes: the $T$-$C_{380}$, $D_3$-$C_{384}$, $D_3$-$C_{440}$ and $D_3$-$C_{672}$ named above. We will refer to them by their short-hand names; their fully qualified names are given in the previous section. At that time were not able to perform a general Goldberg-Coxeter transformation GC$_{k,l}$ on these fullerenes to obtain the corresponding general spirals with jumps. We can now fill this gap with our algorithm introduced and correct one of our conjectures.

### 1. $T$-$C_{380}$-fullerene

The general RSPIs including jumps for the Goldberg-Coxeter transforms of $T$-$C_{380}$-fullerene are listed in Table II. In the Goldberg-Coxeter transforms where simple face spirals without jumps are found, the number of such spirals are small compared to the number of all possible spirals, which is 672. For example, GC$_{2,1}(T$-$C_{380}$-fullerene) admits 38 symmetry distinct spirals out of 456 in total out of 15960 maximum possible spirals, and GC$_{3,1}(T$-$C_{380}$-fullerene) admits 77 symmetry distinct spirals out of 924 in total out of 29640 maximum possible spirals. We note that some of these transforms admit a pentagon start. We also note that large jumps occur, for example GC$_{6,6}(T$-$C_{380}$-fullerene) requires a jump over 13 faces (14 jumps from face to face).

It was previously conjectured that every halma transform of a non-spiralable fullerene is also non-spiralable.[7] Here, we found a counterexample disproving this conjecture. Because of the general relation

\[ GC_{k,l} \circ GC_{m,n} = GC_{k+m-l, n+l} \]

with $k \geq l$ and $n \geq m$, we have GC$_{2,0} \circ GC_{3,2}C_{380} = GC_{2,0}C_{7220} = GC_{6,4}C_{380} = C_{2880}$. Now, this $T$-$C_{2880}$-fullerene is a spiralable fullerene whilst the $T$-$C_{7220}$-fullerene is not. However, for our calculated cases, all combinations of halma and leapfrog transforms of the original graph $T$-$C_{380}$-fullerene do not admit a classical spiral, i.e., GC$_{k,l}(T$-$C_{380}$-fullerene) with $l = 0$ of $k = l$ may all be non-spiralable.

A comment on Goldberg-Coxeter transformations has to be added here. If a fullerene graph $G$ is chiral, then the GC$_{k,l}[G]$ and GC$_{l,k}[G]$ transformations lead to two different graphs if $k \neq l$ and $k,l \neq 0$. We therefore added a few examples of both transformations in Table II.

### 2. $D_3$-$C_{384}$-fullerene

The general RSPIs including jumps for the Goldberg-Coxeter transforms of $D_3$-$C_{384}$-fullerene are listed in Table III. Here, we see that all the investigated halma transforms of $D_3$-$C_{384}$-fullerene are non-spiralable. For the Goldberg-Coxeter transforms GC$_{k,k}(D_3$-$C_{384}$-fullerene) we obtain a different picture, here we see spiralable fullerenes except for GC$_{4,4}(D_3$-$C_{384}$-fullerene).
3. \(D_3\text{-}C_{440}\text{-fullerene}\)

The general RSPIs including jumps for the Goldberg-Coxeter transforms of \(D_3\text{-}C_{440}\text{-fullerene}\) are listed in Table IV. We see much the same picture as for \(T\text{-}C_{440}\text{-fullerene}\), except that for the first time we have spirals appearing which require more than 1 jump.

4. \(D_3\text{-}C_{672}\text{-fullerene}\)

The general RSPIs including jumps for the Goldberg-Coxeter transforms of \(D_3\text{-}C_{672}\text{-fullerene}\) are listed in Table V. As noted before \(D_3\text{-}C_{672}\text{-fullerene}\) is the halma transform of the small spiralable fullerene \(D_3\text{-}C_{168}\text{-fullerene}\).[7] Here, we see only the halma transforms of \(D_3\text{-}C_{672}\text{-fullerene}\) producing non-spiralable fullerenes. We did not investigate the Goldberg-Coxeter transforms of \(D_3\text{-}C_{168}\text{-fullerene}\) here. However, note that while \(GC_{2,0}(D_3\text{-}C_{168}\text{-fullerene}) = D_3\text{-}C_{672}\) or \(GC_{4,0}(D_3\text{-}C_{168}\text{-fullerene}) = GC_{2,0}(D_3\text{-}C_{672}\text{-fullerene})\) are non-spiralable as can be seen from Table V, both \(GC_{3,0}(D_3\text{-}C_{168}\text{-fullerene})\) and \(GC_{5,0}(D_3\text{-}C_{168}\text{-fullerene})\) are spiralable.

V. PERFORMANCE: THEORETICAL BOUNDS AND BENCHMARKS

A. Runtime Complexity of computing the compatibility canonical spiral:

To compute the compatibility canonical spiral, all \(6V\) spiral starts need to be considered in the worst case. There are \(N\) steps in each successful classical spiral unwinding, yielding a total runtime of \(O(N^2)\) for spiralable polyhedra. For non-spiralable polyhedra, each of the \(J\) jump-steps (the sum of all jump lengths) incur a constant amount of extra work, i.e. the compatibility canonical spiral is found in time \(O(N(N + J))\).

The number of jumps is bound by \(N\), and the lengths of individual jumps are bound by \(N/2\) (because the border of \(P\) cannot be longer than that) making the sum of all jump lengths trivially bounded by \(N(N/2)\). This induces a trivial bound of the total worst case runtime of \(O(N(N + J)) = O(N^3)\). However, as even the relatively rare non-spiralable graphs require very few and short jumps (as seen below), we believe that the bound on total jump length can be improved to \(J \sim O(N)\). For the special case of fullerene graphs, it is exceedingly rare that any jump is required at all, whereby the expected scaling \(O(N(N + J))\) reduces to \(O(N^3)\).

B. Runtime Complexity of computing the canonical general spiral:

To compute the canonical general spiral, we need only consider spirals starting in faces of the size that occurs fewest times. If we denote the number of such faces by \(f\) (i.e., 12 for fullerenes), then the runtime complexity is \(O((N + J)f)\), i.e., \(O(N + J)\) when this is bounded by a constant.

Hence, for any class of polyhedral graphs with a face size that appears at most a constant \(f \sim O(1)\) number of times, the canonical general spiral is computed in time \(O(N + J)\). This is true, for example, for all convex polyhedral graphs (i.e. with no faces larger than a hexagon), of which fullerenes is the largest class; and it is true as well e.g. for fulleroids that have at most a constant number of faces of size \(\geq 7\). As \(J\) has been negligible in all cases we have seen, we conjecture that \(J\) is (at most) \(O(N)\), whereby \(O(N + J) = O(N)\), and the total runtime complexity in this case is linear in the graph size.

However, the completely general case of polyhedral cubic graphs requires \(O(N)\) spiral starts in the worst case to find a canonical general spiral: It is possible to construct classes of highly non-convex polyhedra with \(O(N)\) negative-curvature faces (heptagons, octagons, ...), balanced out by \(O(N)\) positive-curvature faces (pentagons, squares, and triangles). For these special classes, the formal asymptotic runtime complexity is no better than the compatibility canonical spiral, \(O(N(N + J))\).

C. Benchmarks

We have investigated the performance of the algorithm on a large number of fullerene graphs, and have found strong indication that computing the canonical general spiral is linear for fullerenes, and that the pentagon-index notation is near to constant size (of the \(2.7 \times 10^{12}\) isomers up to \(C_{100}\), the largest representation takes \(4 \times 2 + 12 = 20\) integers). Future work will investigate the performance for other classes of polyhedral graphs; however, their numbers
Figure 13: (a) The number of jumps required for every isomer C_{20} to C_{400}. (b) Distribution of total jump lengths J for those isomers that require jumps. (c) Total jump-length distribution in logarithmic scale.

Figure 14: Linear scaling: (a) Time $t_{spiral}$ in microseconds to compute the canonical general spiral, and (b) the time divided by the number of vertices N. The line shows $27N\,\mu s$, the average time per vertex.

grow so fast that systematic benchmarking is unrealistic. The benchmark was performed on a 2.6GHz Intel Xeon E5-2650v2 CPU (released 2013).

a. All isomers up to C_{200}: We have calculated canonical general spirals for all 1,943,623,681 fullerene isomers from C_{20} to C_{200}. As a correctness check, we have verified that we reproduced the previous results of Brinkmann et al. [6, 37]: we found exactly the same 278 isomers out of the 1,943,623,681 up to C_{200} to not admit classical spirals starting in pentagons, i.e. for which the general spiral requires at least one jump. The average time to find the canonical general spirals was approximately $N \times 27\,\mu s$ without discernible dependence on the number of jumps or total jump length J. The exhaustive list of fullerene graphs was generated on-the-fly using the Buckyen filter in the development version of Fullerene [33, 35].

b. All isomers with jumps up to C_{400}: As a second benchmark, we have computed canonical general spirals for all fullerene isomers up to C_{400} that do not have a classical spiral starting in a pentagon. Out of the 2,653,606,256,199 isomers up to C_{400}, there are 2,702,266 such isomers, i.e., about one in a million isomers requires a jump. As seen in Figure 13(a), the vast majority of the few isomers that require a jump needs either one or two: only 110 (about one isomer per $2.4 \times 10^{10}$) require 3 or 4 jumps, and none required more. Similarly, the total jump length J (Figure 13(b)) was 10 or more in less than a billionth of the isomers up to C_{400} (1050 isomers in total). The longest total jump length was 35, achieved by a single D_3-C_{366}-isomer with 4 jumps: D_3-[CS: 158,6,165,8,169,10,173,11; 1,4,5,12,166,167,168,170,171,172,173,185]–C_{366}–fullerene with compatibility spiral D_3-[CS: 80,81,113,114,126,127,153,154,166,167,180,181]–C_{366}–fullerene. Hence the J-term is negligible in all 2.7 $\times$ 10^{12} isomers, and we see $O(N + J) \approx O(N)$ in practice. This linearity is quite apparent from Figure 13(c): the time required to find the canonical spiral for a C_N isomer is very nearly $N \times 27\,\mu s$. The exhaustive list of graphs with no pentagon spiral start was taken from the House of Graphs database [37].

c. Large fullerenes: We have computed the canonical general spirals of the graphs from Tables II–V and extended the series up to 50,000 vertices. We have additionally computed all the Goldberg-Coxeter transforms up to C_{50,000} for all C_{100}, C_{104}, C_{124}, C_{132}, C_{152}, C_{166}, C_{180}, C_{192}, and C_{200}-isomers that do not admit spirals starting in pentagons.

---

<table>
<thead>
<tr>
<th># jumps</th>
<th>Isomers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2,653,603,540,914</td>
</tr>
<tr>
<td>1</td>
<td>2,703,505</td>
</tr>
<tr>
<td>2</td>
<td>11,670</td>
</tr>
<tr>
<td>3</td>
<td>107</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

---

URL: http://mc.manuscriptcentral.com/lfnn Email: fncneditorialoffice@gmail.com
In total, 11,141 fullerene graphs with sizes ranging between C_{100} and C_{50,000}. These are chosen to challenge our implementation: Experimentally, we have found that Goldberg Coxeter transforms of graphs with few or no spirals are more likely than average to admit few or no spirals. The result is shown in Figure 15, and we again verify that the scaling is linear at a rate averaging about 23\(\mu\)s per vertex up to around 32\(\mu\)s per vertex for particularly difficult subseries.

Figure 16 shows the timing for the compatibility canonical spiral scheme for a subset of 722 of these graphs: the 4 series from Tables II-V extended up to C_{50,000}, together with similar series for C_{130} and C_{184}. We can clearly see the \(O(N^2)\) scaling: the quadratic rate averages around 0.63\(\mu\)s per vertex squared, and ranges from about 0.45\(\mu\)s/\(N^2\) to 1.25\(\mu\)s/\(N^2\). Notice the over three orders of magnitude difference in total computation times from the canonical general spiral: The canonical general spiral was found for a non-spiralable C_{49,056} in 1.1 s, but 1305 s was required to find the compatibility spiral.

Although we have attempted to generate pathological cases, in none of the over 2 billion fullerene graphs, for which we have calculated canonical spirals, has the sum of all jump lengths been comparable in size to \(N\). Hence, although we have not found a formal proof, we believe that \(O(N + J) = O(N)\) for fullerenes, whence our algorithm calculates canonical spiral representations in linear time. This is verified for all 2.7 trillion fullerene graphs up to C_{400}.
D. Acknowledgments

PS is indebted to the Alexander von Humboldt Foundation (Bonn) for financial support in terms of a Humboldt Research Award and to the Philipps University Marburg where the fullerene project originally started. JA was partially supported by VILLUM FONDEN through the network for Experimental Mathematics in Number Theory, Operator Algebras, and Topology, and by the Danish Council for Independent Research Sapere Aude grant “Complexity through Logic and Algebra” (COLA). LNW thanks the Niels Bohr Institute for financial support during his visit to Copenhagen. LNW was supported by the Research Council of Norway (RCN) through a Centre of Excellence Grant (Grant No. 179568/V30). We thank Gunnar Brinkmann, Jan Goedgebeur, and Brendan McKay for the inclusion of the Buckeye fullerenes generator into our Fullerene software [35], and for their kind help for using their House of Graphs database [37]; without either, the exhaustive benchmarking of our method and verification for all 200,000 fullerenes up to $C_{400}$ would not have been possible.

VI. APPENDIX

In this appendix, we will show that the general spiral algorithm is indeed general, i.e., it unwinds any dual of a planar cubic graph into a general spiral, and in cases where an ordinary spiral exists, it is found.

A. Completeness of the General Spiral Algorithm

The main difference between the ordinary spiral algorithm and the general one is, that, as we follow the spiral and add vertices from the boundary $\partial R$ of $R$ to the spiral ($P$ and $R$ are defined as before), we skip any that would disconnect $R$ when removed, and hence, lead to a dead end (cul-de-sac) for the spiral. Such a vertex is called a cut vertex. This means that at any step of the algorithm, both $P$ and $R$ are connected.

In order to prove that the method is complete for any dual of a planar cubic graph, it suffices to show that at each step, there is at least one vertex in $\partial R$ that can be added to the spiral. However, it is not sufficient that a vertex $v$ is not a cut vertex in order to be added to the spiral, because not all vertices on $\partial R$ are reached while traversing $\partial P$.

Figure 17 shows when a vertex can be placed, and when it cannot.

Figure 17: In this figure, the vertex $r_2$ has two adjacent neighbors $p_1, p_2$ in $\partial P$ and will be considered when traversing $\partial P$ in the spiral. In contrast, $r_1$ is not reached, since it has only one neighbor in $P$ and hence, does not define a triangle on the border of the spiral.

15a and 15b show that this is the case also for the 11,413 tested graphs up to $C_{50,000}$. For large fullerenes, the timing seems to improve from $\approx N \times 27 \mu s$ (upper line) to $\approx N \times 23 \mu s$ (lower line). The improvement with larger $N$ is likely due to the linear part beginning to dominate fully over lower-order overhead. The worst case outliers take $\approx N \times 32 \mu s$. Hence, we compute each general spiral in approximately $N \times 23 \mu s / 120 \approx 200 ns$, as the canonical spiral requires computing one general spiral for each of the 10 directed edges incident to each of the 12 degree-5 vertices in the dual triangulation.

The general spiral algorithm is complete for all duals of planar cubic graphs.

Theorem 1. At each step, there is at least one vertex on $\partial R$ that is accessible and that is not a cut-vertex.

Corollary 2. The general spiral algorithm is complete for all duals of planar cubic graphs.
In an oriented triangulation, a directed edge $u \rightarrow v$ uniquely defines a triangle and hence the third vertex $w$, by choosing either clockwise (CW) or counter-clockwise (CCW) traversal. Let $\text{next}_{\text{CCW}}(u, v)$ be the function that maps the directed edge $u \rightarrow v$ to the third vertex in the CCW-triangle $(u, v, w)$.

When traversing the boundary of $P$ in clockwise order while building the spiral, for each directed edge $u \rightarrow v$, we always add the vertex $w = \text{next}_{\text{CCW}}(u, v)$ corresponding to the CCW-triangle, since the clockwise vertex has always already been placed. Correspondingly, in the general spiral algorithm, the counter-clockwise vertex is always unplaced. Similarly, if we traverse $\partial R$, we have the same rules: When traversing clockwise, the counter-clockwise triangles, which are always outside $R$, are considered.

**Lemma 3.** Consider a vertex $r \in \partial R$, and the ingoing and outgoing directed edges $q \rightarrow r$ and $r \rightarrow s$ in a clockwise walk of $\partial R$. Then $r$ is accessible if $\text{next}_{\text{CW}}(q, r) \neq \text{next}_{\text{CW}}(r, s)$.

**Proof.** Let $r \in \partial R$ be a vertex on the boundary of $R$, and let $q \rightarrow r$ and $r \rightarrow s$ be the ingoing and outgoing arcs in a clockwise walk of $\partial R$.

Consider now the two triangles $(q, r, u)$ and $(u, r, v)$, where $u = \text{next}_{\text{CW}}(q, r)$ and $v = \text{next}_{\text{CW}}(u, r)$. There are two possible cases:

If $v = s$ (a), the edge $u - v$ is not part of $P$, and we cannot conclude that $r$ is accessible. But if $v \neq s$ (b), then the edge $u - v$ is part of $\partial P$, and both $u$ and $v$ neighbor $r$, whence $r$ is accessible. Because $\text{next}_{\text{CW}}(u, r) = v \iff \text{next}_{\text{CW}}(v, u) = r$, this concludes the proof.

**Lemma 4.** Let $C = (c_1, \ldots, c_n)$ be a simple cycle in a triangulation $T$ of the sphere. If, for any planar layout of $T$, there exists $p$ such that

$$p = \text{next}_{\text{CW}}(c_1, c_2) = \text{next}_{\text{CW}}(c_2, c_3) = \ldots = \text{next}_{\text{CW}}(c_n, c_1)$$

then $p$ is connected to no other vertices and is either the only vertex inside the cycle or the only vertex outside. The same holds for $\text{next}_{\text{CCW}}$.

**Proof.** Assume that there exists such a $p$ and a planar layout of $T$ places it inside the cycle. Since the triangles $(c_1, c_2, p), \ldots, (c_n, c_1, p)$ are faces forming a closed disk around $p$, there can be no further vertices connected to $p$, and no other vertices on the inside of $C$:

Assume now that $p$ is placed outside $C$. Since $T$ is a triangulation of the sphere, we can create a new planar layout that makes all vertices that are internal to $C$ external, and vice versa, hence placing $p$ inside. By the previous argument, it must be the only interior vertex in the new layout, and hence the only exterior vertex in the old layout.

**Lemma 5.** If $C = (c_1, \ldots, c_n)$ is a simple cycle in $\partial R$, then at least two vertices are accessible.

**Proof.** Assume that at most one vertex is accessible. Without loss of generality, we can assume that $c_2, \ldots, c_n$ are inaccessible, and $c_1$ may or may not be so. By Lemma 3, each inaccessible vertex defines two triangles as follows:

$$c_2 \text{ inaccessible } \Rightarrow \text{next}_{\text{CCW}}(c_1, c_2) = \text{next}_{\text{CCW}}(c_2, c_3)$$

$$c_3 \text{ inaccessible } \Rightarrow \text{next}_{\text{CCW}}(c_2, c_3) = \text{next}_{\text{CCW}}(c_3, c_4)$$

$$\vdots$$

$$c_n \text{ inaccessible } \Rightarrow \text{next}_{\text{CCW}}(c_{n-1}, c_n) = \text{next}_{\text{CCW}}(c_n, c_1)$$

Hence, there is a single point $p \in P$ such that $p = \text{next}_{\text{CCW}}(c_1, c_2) = \ldots = \text{next}_{\text{CCW}}(c_n, c_1)$, whereby Lemma 4 immediately tells us that either $P = \{p\}$, or $P$ is disconnected, both of which are impossible by construction. Hence, $C$ must contain at least two accessible vertices. The same holds for $\text{next}_{\text{CW}}$.  

We are now ready to prove completeness of the method.

**Proof of Theorem 1.** We can distinguish between two cases for the shape of the boundary \( \partial R \):

**Case 1:** If \( \partial R \) contains leaf vertices, i.e., vertices that have degree 1 in \( R \), all such vertices are automatically accessible and not cut-vertices.

This can be easily seen from Lemma 3: If \( r \) is a leaf vertex in \( \partial R \), then a walk \( q \to r \to s \) through it has \( q = s \).

Since \( \text{next}_{CCW}(q,r) = \text{next}_{CCW}(r,q) \), we automatically get \( \text{next}_{CCW}(q,r) \neq \text{next}_{CCW}(r,s) \), since from planarity the CW-triangle is always different from the CCW-triangle. Hence, \( r \) is accessible. In addition, a leaf by definition can not be a cut vertex.

**Case 2:** If \( \partial R \) has no leafs, then it can be decomposed into single cycles connected by (possibly length-0) linear segments. Such decompositions are not unique, but there is always at least one.

Given any such decomposition of \( \partial R \), we can derive the graph that has the simple cycles and crossing points of the linear segments as vertices, and the linear segments as edges. There are no parallel edges, since that would yield an additional simple cycle instead of the parallel segments. In addition, the graph defined in this manner is a tree, since if it had cycles, either \( P \) would be disconnected (interior of the cycle non-empty) or \( \partial R \) would not be a boundary (interior empty).

Now we can pick any leaf of the tree, which will be a simple cycle with at most one contact point (0 if \( \partial R \) is itself a simple cycle, 1 if not). By Lemma 5, such a cycle must have at least two accessible vertices, and hence at least one of these vertices is not a cut vertex.

\( \blacksquare \)

[38] This does not hold for non-cubic polyhedra as for example the omni-truncation of an icosahedron leads to the well known spiralable $I_{8}-C_{60}$-fullerene.
[39] The scheme is discussed already in [19], but we have not found a complete description in the literature outside of Ref.[27].
Table II: Compatibility canonical face spiral pentagon indices for Goldberg-Coxeter transforms \(GC_{k,l}\) of \(T\)-C\(_{380}\)-fullerene up to \(k = l = 5\) and a few more selected fullerenes. Point group symmetry is \(T\) for all Goldberg-Coxeter transforms. Goldberg-Coxeter indices \(k, l\), number of vertices \(N\), and canonical ring (face) spiral pentagon indices (RSPI) including jumps.
<table>
<thead>
<tr>
<th>k/l</th>
<th>N</th>
<th>jumps</th>
<th>RSPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1</td>
<td>384</td>
<td>29 30 31 49 146 147 171 191 192 194</td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>1152</td>
<td>1 11 13 24 453 496 498 542 572 574 577</td>
<td></td>
</tr>
<tr>
<td>1/3</td>
<td>2688</td>
<td>1 17 20 29 1025 1159 1190 1193 1196 1224 1289 1318</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>4992</td>
<td>1 36 40 44 2011 2099 2197 2285 2327 2331 2335 2408</td>
<td></td>
</tr>
<tr>
<td>2/0</td>
<td>1536</td>
<td>[298,4]</td>
<td>197 199 201 298 565 615 669 719 756 758 760 770</td>
</tr>
<tr>
<td>2/1</td>
<td>2688</td>
<td>-</td>
<td>200 306 309 430 1064 1137 1206 1210 1312 1330 1339 1343</td>
</tr>
<tr>
<td>2/2</td>
<td>4608</td>
<td>-</td>
<td>51 108 116 272 1735 1721 4905 2096 2246 2284 2287 2304</td>
</tr>
<tr>
<td>3/0</td>
<td>7296</td>
<td>-</td>
<td>1 58 63 68 2936 3156 3214 3219 3273 3543 3590 3638</td>
</tr>
<tr>
<td>3/1</td>
<td>3456</td>
<td>[580,5]</td>
<td>369 372 375 581 1315 1318 1354 1549 1549 1700 1703 1706 1730</td>
</tr>
<tr>
<td>3/2</td>
<td>1133,3</td>
<td>-</td>
<td>641 829 924 1136 2706 2932 3153 3159 3556 3611 3626 3646</td>
</tr>
<tr>
<td>3/3</td>
<td>10368</td>
<td>-</td>
<td>51 186 192 449 4906 4426 4432 4825 5051 5138 5144 5181</td>
</tr>
<tr>
<td>3/4</td>
<td>6144</td>
<td>[1183,8]</td>
<td>777 781 785 1183 2305 2405 2721 2821 3022 3026 3030 3074</td>
</tr>
<tr>
<td>4/0</td>
<td>8064</td>
<td>-</td>
<td>1346 1746 1894 2342 3167 3399 3361 3344 3494 3680 3860 3985</td>
</tr>
<tr>
<td>4/1</td>
<td>10752</td>
<td>[2129,11]</td>
<td>989 1461 1467 2000 4214 4503 4793 4929 5141 5312 5353 5361</td>
</tr>
<tr>
<td>4/2</td>
<td>14208</td>
<td>[1561,4]</td>
<td>778 1094 1201 1565 5371 5837 5868 6303 6922 7034 7055 7110</td>
</tr>
<tr>
<td>4/3</td>
<td>18432</td>
<td>[1161,2]</td>
<td>439 765 773 1370 6852 7542 8056 8237 8978 9134 9142 9214</td>
</tr>
<tr>
<td>5/0</td>
<td>9600</td>
<td>[1699,9]</td>
<td>1093 1098 1103 1099 3661 3666 4306 4311 4722 4732 4802</td>
</tr>
<tr>
<td>5/1</td>
<td>11904</td>
<td>-</td>
<td>2050 2056 2886 2892 4666 4816 5210 5360 5441 5509 5736 5950</td>
</tr>
<tr>
<td>5/2</td>
<td>14976</td>
<td>-</td>
<td>1825 2623 2804 3528 5860 6208 6557 6806 6892 7148 7397 7465</td>
</tr>
<tr>
<td>5/3</td>
<td>18816</td>
<td>-</td>
<td>1340 1747 2031 2817 7328 7885 8249 8444 9171 9302 9357 9390</td>
</tr>
<tr>
<td>5/4</td>
<td>23424</td>
<td>[3021,9]</td>
<td>1601 2200 2352 2863 8594 9381 10164 10174 11410 11599 11626 11708</td>
</tr>
<tr>
<td>5/5</td>
<td>28800</td>
<td>-</td>
<td>197 573 583 1406 11191 12276 12286 13376 14072 14272 14282 14397</td>
</tr>
<tr>
<td>5/6</td>
<td>38232</td>
<td>[2656,12]</td>
<td>1741 1747 1753 2656 5221 5371 6157 6307 6800 6806 6812 6914</td>
</tr>
<tr>
<td>6/0</td>
<td>16512</td>
<td>-</td>
<td>2731 2908 3896 4097 6375 6752 7377 7467 7553 7727 7820 8254</td>
</tr>
<tr>
<td>6/1</td>
<td>19968</td>
<td>-</td>
<td>4602 6097 6371 7004 7564 7741 7933 8321 8740 9039 9226 9873</td>
</tr>
<tr>
<td>6/2</td>
<td>24192</td>
<td>-</td>
<td>2052 3078 3087 4251 9613 10261 10900 10912 11644 11948 12044 12056</td>
</tr>
<tr>
<td>7/0</td>
<td>18816</td>
<td>[3406,13]</td>
<td>2201 2208 2215 3406 7183 7190 8450 8457 9256 9263 9270 9410</td>
</tr>
<tr>
<td>7/1</td>
<td>21888</td>
<td>-</td>
<td>5300 3484 4574 4792 8540 8740 9990 10069 10295 10393 10491 10942</td>
</tr>
<tr>
<td>7/2</td>
<td>25728</td>
<td>-</td>
<td>4370 6090 6364 8015 9745 10208 10671 11000 11457 11564 12338 12746</td>
</tr>
<tr>
<td>8/0</td>
<td>24576</td>
<td>[4717,16]</td>
<td>3089 3097 3105 4717 9313 9513 10977 11177 12090 12098 12106 12290</td>
</tr>
</tbody>
</table>

Table III: Compatibility canonical face spiral pentagon indices for Goldberg-Coxeter transforms $G_{k,l}$ of $D_3 \cdot C_{384}$-fullerene up to $k = l = 5$ and a few more selected fullerenes. Point group symmetry is $D_3$ for all Goldberg-Coxeter transforms. Goldberg-Coxeter indices $k, l$, number of vertices $N$, and canonical ring (face) spiral pentagon indices (RSPI) including jumps.
Table IV: Compatibility canonical face spiral pentagon indices for Goldberg-Coxeter transforms GC\(_{k,l}\) of \(D_3\)-\(C_{440}\)-fullerene up to \(k = l = 4\) and a few more selected fullerenes. Point group symmetry is \(D_3\) for all Goldberg-Coxeter transforms. Goldberg-Coxeter indices \(k, l\), number of vertices \(N\), and canonical ring (face) spiral pentagon indices (RSPI) including jumps.
Table V: Compatibility canonical face spiral pentagon indices for Goldberg-Coxeter transforms GC_{k,l} of D_3-C_{672}-fullerene up to $k = l = 3$ and a few more selected fullerenes. Point group symmetry is $D_3$ for all Goldberg-Coxeter transforms. Goldberg-Coxeter indices $k,l$, number of vertices $N$, and canonical ring (face) spiral pentagon indices (RSPI) including jumps.