MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **62** (2009) 209-220

ISSN 0340 - 6253

An efficient algorithm for the generation of planar polycyclic hydrocarbons with a given boundary

G. Brinkmann[*]
Applied Mathematics
and Computer Science
Krijgslaan 281 - S9
Ghent University
B 9000 Ghent, Belgium

B. Coppens[†]
Electronics and Information
Systems Department
Sint-Pietersnieuwstraat 41
Ghent University
B 9000 Ghent, Belgium

(Received September 3, 2008)

Abstract

In this article we give an algorithm for the constructive enumeration of planar polycyclic hydrocarbons with hexagon rings and at most 5 pentagon rings that have a prescribed cyclic sequence of valencies of the boundary vertices.

Introduction

In this article we will discuss planar polycyclic hydrocarbons only in the form of their mathematical models. So a planar polycyclic hydrocarbon is a bridgeless plane graph with a distinguished outer (or unbounded) face O. All faces except O are pentagons or hexagons, all vertices not in the boundary of O have degree 3 and all vertices in the boundary of O have degree 2 or 3. The cyclic sequence of vertex degrees as they occur in the boundary of O is called its *boundary sequence*. Since we do not distinguish between mirror images, we consider the sequence obtained in clockwise order around O and the one obtained in anti-clockwise order around O as identical. In this article we will denote planar polycyclic hydrocarbons shortly as *patches* and patches with p bounded pentagons

[*]Gunnar.Brinkmann@ugent.be
[†]Bart.Coppens@elis.ugent.be

and the remaining bounded faces hexagons as p-patches. We will always assume that a given boundary sequence consists of 2's and 3's only.

Boundary sequences of patches have been useful tools in several applications. In [7] boundary sequences are used to encode fusenes and benzenoids, boundary sequences are the general method to classify nanotube caps (see e.g. [9],[6]) and different sub-patches of fullerenes having the same boundary sequence were proposed as a mechanism for fullerene growth [10] or isomerisation [13]. In [2],[11] a catalogue of possible growth and isomerism patches is given, but since at that time no efficient generator for these patches was available, the generator for planar polycyclic hydrocarbons contained in CaGe (see [3]) was used and the output was sorted and filtered. If one looked for the fillings of some particular boundary, this would be a very inefficient way to do it.

In [12],[8] an algorithm to fill given boundaries with vertices of a given degree and faces of a given size is described. The overlap with the algorithm described here (also in the *principal choice where to add a face*) is the case with all faces hexagons and all interior vertices of degree 3. Nevertheless, in [12] there is no proof given that the algorithm actually terminates. This proof will be given here. Furthermore we describe several optimisations that increase the speed of the program enough to make it sufficiently fast to be able to enumerate all fillings for boundaries that may occur in the context of chemistry.

Basics

We will restrict ourselves to p-patches with at most $p \leq 5$ pentagons. For $p \geq 6$ a complete enumeration is impossible because there may be infinitely many patches with the given boundary. This can be seen by looking e.g. at nanotube caps to which an arbitrary number of hexagon rings can be added without changing the boundary structure. For $p < 6$ an upper bound for the number of vertices of the patch has been proven (see [1]), so it follows immediately that there is just a finite number of patches with a given boundary – and therefore complete enumeration is (in principle) possible.

A well known formula that follows directly from the Euler formula is stated in the following lemma:

Lemma 1 *In a p-patch with d vertices of degree two and t vertices of degree three in the boundary we have $d - t = 6 - p$.*

This implies that for $p < 6$ there are more vertices with degree 2 in the boundary than vertices with degree 3. Therefore the longest subsequence of 2s has a length of at least 2.

A pre-patch (G, U) is a 2-connected plane graph – possibly with dangling edges – together with a set U of bounded faces, which we will call *unfinished* faces and a distinguished outer (or unbounded) face. All dangling edges belong to faces in U and all bounded faces that are not in U are pentagons or hexagons. All vertices have degree 2 or 3 and vertices that are not in the boundary of the outer face have degree 3 (counting dangling edges). Note that with Lemma 1 it follows that for a given unfinished face the number of pentagons of every possible filling is the same and can easily be computed from the boundary.

So for a given boundary sequence it is easy to construct a pre-patch with this boundary sequence: one just has to construct a cycle of the given length, fix the unbounded face and add a dangling edge into the bounded face for every vertex with degree 3. Then the set U is the set with the bounded face only. We call this the minimal pre-patch (of the boundary sequence). To each unfinished face we can assign a boundary sequence by interpreting vertices in the boundary of the face that have dangling edges in the inside as vertices with valency 3 and the others as vertices with valency 2. For a given pre-patch with boundary C we write $d(C)$, resp. $t(C)$ for the number of vertices with valency 2, resp. 3 in C.

The basic algorithm

We will first describe the basic structure of the algorithm without optimisations:

We start with the minimal pre-patch and try to extend it to a pre-patch with U the empty set – that is: a patch. We try to fill all the unfinished faces with pentagons and hexagons. Each time an unfinished face is filled with pentagons and hexagons and there are no dangling edges left, it is removed from U and we start to fill the next unfinished face. In case there is no unfinished face left, we output the patch.

The filling of the unfinished faces is done one pentagon or hexagon at a time. A new face is always attached to the boundary and starts and ends at dangling edges that then become real edges. If possible the dangling edge after which the new face is attached, is chosen in a way that three or more vertices of degree 2 (so vertices without dangling edges) follow. Only in the case where no such places are found, a place with only 2 vertices of degree 2 following is chosen. In fact the algorithm starts at a position so that the boundary sequence starting there is lexicographically minimal and afterwards always uses that place with the properties described above that is closest to the last added edge in counterclockwise direction. The choice for the first edge will later be used for isomorphism rejection, but another (fixed) choice for the other edges would not affect the correctness of the algorithm as long as the operations described can be applied.

We distinguish between 4 possibilities for the number t of vertices of degree two following each other. The operations used to extend the pre-patch in these cases are also sketched in Figure 1.

$t > 4$ If $t \in \{5, 6\}$ and the length of the boundary is equal to t, then this face can be removed from U – it is completely filled. Otherwise more than 6 different vertices of the boundary would have to be part of the face at this position – so the unfinished face cannot be filled.

$t = 4$ In this case every filling must have a hexagon at this position consisting of exactly the 4 vertices in the boundary with valency 2 and the two neighbouring vertices with dangling edges. So we connect the dangling edges of the two vertices neighbouring the sequence of 2s to form a hexagon and take this as the new pre-patch, considering the face on the other side of the newly built hexagon as unfinished.

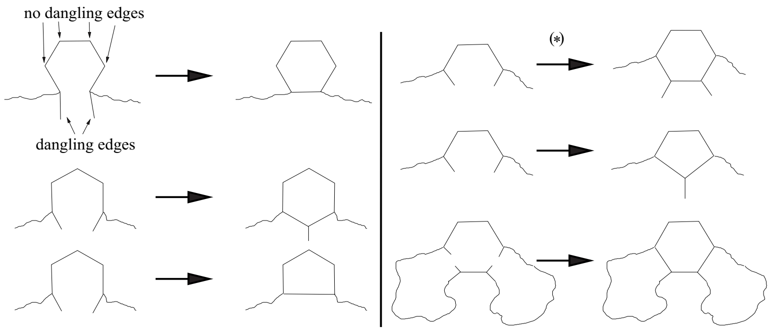

Figure 1: The possible operations to extend a given pre-patch.

$t = 3$ In this case every filling must have a pentagon or a hexagon at this position consisting of exactly the 3 vertices in the boundary with valency 2, the two neighbouring vertices with dangling edges and – in case of a hexagon – one new vertex, which cannot lie on the already constructed boundary. We construct two pre-patches from this. In case the number of 2's and 3's in the boundary sequence of this unfinished face also allows a pentagon, we proceed like in the previous case, connecting two dangling edges with each other. When adding a hexagon we also add a new vertex with a dangling edge to the inside.

$t = 2$ In this case it is possible that the intersection of the face at this position with the already constructed boundary is not connected and a large number of pre-patches can be constructed from this one. In case of a connected boundary, we can proceed as in the previous cases inserting one or two new vertices with dangling edges pointing to the inside. In case of a disconnected intersection with the already known boundary, it is easy to see that the face must be a hexagon. In order to cover these cases we connect the two dangling edges starting at vertices bordering the two vertices of degree 2 with two (other) dangling edges starting at neighbouring vertices to form a hexagon as shown in Figure 1. This gives us two new and smaller unfinished faces that are added to U. This step is only perfomed for counterparts where the boundaries of the resulting new unfinished faces both induce between 0 and 5 pentagons.

Concerning efficiency, the most problematic operation is the case when for $t = 2$ a hexagon is formed in a way connecting two parts of the boundary. There may be a large number of places that can be used as counterparts and maybe only few of them finally lead to patches.

It is obvious that with these operations all patches with the given boundaries can be constructed and at the moment we may even get isomorphic copies. Because all possibilities for how a face can be connected to the already constructed pre-patch are covered, one can e.g. prove this by induction on the number of faces.

What is less obvious is that the algorithm actually terminates. Except for the operation marked (*) in Figure 1, all operations decrease the sum of all lengths of the boundaries of unfinished faces. So if the algorithm does not terminate, there can only be a finite number of these operations involved and therefore there must be an infinite sequence of the operation marked (*).

Lemma 2 *Given a pre-patch (G, U) and a face $f \in U$ for which the number d of vertices of degree 2 in the boundary and the number t of vertices of degree 3 in the boundary fulfills $d > t$. Then it is not possible to apply an infinite sequence of operations of the kind (*) to f.*

Proof: We assume that the boundary B of the pre-patch has length l in the beginning and will prove that after a finite number of operations the length of the boundary of the unfinished face will be smaller than l. Applying this result inductively gives that an infinite sequence of operations cannot exist.

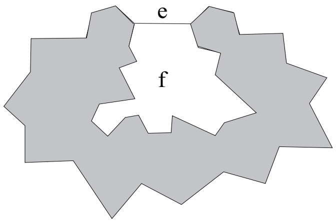

Figure 2: The situation when operation (*) is applied in a way that the same edge e is in the boundary of the unfinished face in every step.

(a) We will first prove that no such sequence exists that has the property that an edge e of B lies in the boundary of the unfinished face after each operation. Assume the contrary – that is that such an edge e existed for an arbitrary number s of steps. Then we would get the situation in Figure 2. The grey part represents the graph without the edge e. It contains only hexagons (filled in by operation (*)) and has a boundary length of $2l - 2$ because the outer boundary and the inner boundary have length l and the edge e is counted twice. We have to take into account that this part does not have to be a patch but can possibly consist of several patches connected by edges, but we can easily conclude that it contains a 0-patch (that is a part without bridges) with more than $2s/l$ faces and boundary length smaller than $2l$. These are very weak bounds, but sufficient to enable us to choose s in a way that leads to a contradiction to e.g. [1] where an explicit formula for the maximum number of faces in a 0-patch with a given boundary length is given.

So case (a) cannot occur and after a finite number of steps each edge in B is adjacent to a hexagon added by operation (*).

(b) We will now look at the length of the boundary we get when removing the faces sharing an edge with the boundary. We can assign every vertex in B that has degree 3 to the hexagon following it in clockwise direction around B. Because every hexagon sharing an edge with the boundary is assigned at least one vertex (maybe more), we get for the number h of these hexagons that $h \leq t$. Because each of these hexagons is adjacent to at least two other hexagons in this set, the number l' of edges that are adjacent to these hexagons but are neither in the outer boundary nor between two such hexagons can now be estimated as $l' \leq 6 * h - (d + t) - 2h = 4h - (d + t) \leq 4t - (d + t) < 2t < l$. These edges form cycles and one of them – say C – must contain the unfinished face f in its interior. If this procedure could be repeated an arbitrary number of times, it would finally lead to a pre-patch with a negative boundary length proving that such a situation is impossible.

What is left to be shown is that C also fulfills $d(C) > t(C)$ when interpreted as the boundary of the pre-patch in its interior. But inside B and outside C are only hexagons, because operation (*) makes sure that there is always just one unfinished face and only hexagons are added. It is easy to show that B can be reduced to C by just removing hexagons that have a connected intersection with the outer boundary. But then again it is an easy exercise to show that such an operation doesn't alter the value of $d(C') - t(C')$ with C' the boundary of the pre-patch.

■

Optimizations

As mentioned before, the bottleneck concerning efficiency is the case where a hexagon is glued to two parts of the boundary. *In principle* a segment of the form $3, 2, 2, 3$ in the boundary can be connected to any other segment in the boundary that has the form $3, 3$ and fulfills the conditions for the number of pentagons on the parts. In case of a connected intersection of the new face with the boundary, there are at most two possibilities – either a hexagon is attached or a pentagon (which can happen in at most 5 cases). This way the boundary is either filled up very fast or contradictions are detected. On the other hand the number of segments of the form $3, 3$ that can be used as counterparts for a given segment of the form $3, 2, 2, 3$ can be linear in the length of the boundary. So it is crucial for the efficiency to efficiently detect as many identification operations as possible that do not lead to fillings of the patch. Because the tests are performed every time that no segment with three or more 2s exists in the boundary, it is important that the test whether a given segment of the form $3, 3$ is a possible counterpart is very fast – in the ideal case constant or at most linear in the length of the boundary.

For each possible counterpart we compute the boundary sequences of the two resulting unfinished faces and use the following criteria to detect parts that cannot be filled:

(1) The boundary sequence of a part with $p = 0$ must encode a closed path in the hexagonal lattice when 2 is interpreted as *take next edge to the right* and 3 is interpreted as *take next edge to the left*.

(2) If the boundary of a part has a length of at most a certain value *MAXLENGTH*, e.g. *MAXLENGTH*$= 30$, the boundary must be listed as being fillable in a given datafile.

For the last criterion the program reads some datafile. For every boundary sequence of length at most *MAXLENGTH* that begins with two 2's and ends with a 3, this file encodes the information whether such a boundary can be filled or not. This datafile is produced in advance by the same program – just that it is run without this criterion for optimisation. For the example running times in the end we used *MAXLENGTH*=30. For this value the size of the file is 33.554.430 bytes. The generation of this data took 8.5 minutes on a 2.6 GHZ Pentium 4 computer running GNU/Linux.

Each time a hexagon that connects two parts of the boundary is attached, the number of unfinished faces is increased by one. As soon as it is detected that one of the unfinished faces cannot be filled, the generation backtracks to the point when the hexagon was added that produced this unfinished face and all changes to the pre-patch performed afterwards are made undone. So if a hexagon splits an unfinished face in two parts and one of the parts can be filled and the other not, then at most one filling of the fillable part will be constructed before it is detected that the program can backtrack.

Isomorphism rejection

Assume first that the boundary sequence has no symmetries. Then it is obvious that an isomorphism between two structures must be the identity on the boundary and then one can easily show by induction that the isomorphism is in fact the identity on the set of vertices.

Different steps in the construction always lead to graphs where the identity on the set of vertices does **not** induce an isomorphism. This means that in case of a trivial symmetry of the boundary all structures are automatically non-isomorphic and no checks are necessary.

Isomorphic patches can only be generated in case the boundary has non-trivial symmetries as e.g. in Figure 3. In this case the isomorphism between the two different fillings induces a non-trivial automorphism of the boundary of the patch. For $p \leq 5$ there is a constant upper bound for the number of isomorphic copies of a p-patch that can be generated while for $p = 6$ the upper bound is linear in the length of the boundary.

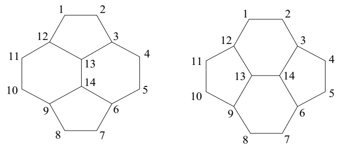

Figure 3: two isomorphic patches that are constructed by the algorithm.

An *automorphism of the boundary* is an automorphism of the cycle subgraph formed by the boundary that respects the vertex degrees.

Lemma 3 *The automorphism group of the boundary C of a p-patch with $p < 6$ has at most 12 elements.*

Proof: The orientation preserving subgroup of the automorphism group has at least half as many elements as the whole group and is a cyclic group. So it is sufficient to show that this cyclic subgroup has at most 6 elements. Assume that the group has order n. Then the boundary can be written as s^n with s a string consisting of 2's and 3's. Writing $d(s)$, resp. $t(s)$ for the number of 2's and 3's in s, we get for the number p of pentagons

$d(C) - t(C) = n * (d(s) - t(s)) = 6 - p$

and therefore (note that $d(C) > t(C)$ and therefore $d(s) - t(s) \geq 1$)

$n = \frac{6-p}{d(s)-t(s)} \leq 6.$

■

We compute the automorphism group of the boundary by computing the positions and corresponding directions (described by a directed edge in the boundary) from which the string of 2's and 3's obtained from that position in that direction is lexicographically minimal. For a cyclic sequence of length n this takes time $O(n)$ for each comparison and there are $2 * n$ possible starts. This gives a worst case complexity of $O(n^2)$. We use binary representations of the boundary – representing a 2 by a 0 bit and a 3 by a 1 bit. Depending on the machine this allows to perform the comparison of up to 32 or up to 64 symbols in the boundary in a single step and leads to a practically very fast comparison of the strings. Note also that this computation has to be done only once in the beginning and not for every structure found. In the beginning we compute all directed edges in the boundary that give a minimal boundary representation. Let us call these edges *starting edges*. In fact the edge that the construction started with will always be one of the starting edges.

In order to detect isomorphic copies we assign a unique representation to every starting edge like in [5]. This representation is obtained from a breadth-first numbering of the vertices numbering the vertices in clockwise or counterclockwise order depending on the direction of the starting edge. The patch is accepted and outputted if and only if the representation obtained from the edge that the construction started with is a smallest representation. In case from another directed starting edge a smaller representation is found, the patch is rejected – knowing that a patch with the smaller respresentation will also be generated or has been generated already. This *BFS*-numbering can easily be computed in time linear in the number of vertices of the patch. Because also comparing the representation can be done in linear time and due to Lemma 3 there is a constant upper bound on the number of starting edges, we have that

Lemma 4 *For a p-patch with $0 \leq p < 6$ and n vertices and given automorphism group of the boundary, the canonicity test can be performed in time $O(n)$.*

Testing

Like for all programs it is important to not only check the algorithm, but also the implementation. To this end two independent implementations were developed and for the second implementation even a slightly different algorithm was developed.

This alternative algorithm does e.g. not use a file with existence information for all boundary sequences starting with two 2's and ending with a 3. Instead it only stores the lexicographically minimal form of the boundary sequence and checks this against the encountered boundary sequences. Furthermore it does not use the additional bounding criteria (1) to detect impossible counterparts for operation (*). Furthermore isomorphism rejection is implemented by simply keeping a list of all generated graphs and comparing newly generated structures to those in the list.

We compared the numbers of structures generated by the two programs for all boundary sequences that would correspond to between 0 and 5 pentagons up to length 30. There was complete agreement.

For longer boundaries, the memory consumption of the independent testing program made it in some cases impossible to compare the results for some boundary sequences. We generated random boundaries with random lengths once in the interval 50 to 100 and once in the interval 100 to 200. Boundaries that would not correspond to 0 to 5 pentagons and boundaries with more than four 2's in a row were not considered for the tests – they are not fillable for trivial reasons. In the interval 50 to 100 about 125.000 boundaries were detected to be unfillable, about 5.000 were fillable and the results for about 100 boundaries couldn't be compared. In the interval 100 to 200 about 425.000 boundaries were detected to be unfillable, about 1.200 were fillable and the results for about 220 boundaries couldn't be compared. In all cases that could be compared the numbers of structures agreed.

The program also has the option to generate only IPR structures – that is structures with no two pentagons sharing an edge. This option is implemented not as a filter but already during the construction it is made sure that no two neighbouring pentagons occur. The correct working of this option was tested by randomly generating boundaries of length 50, 75, 100,. . . 200 until 500 fillable boundaries of every length were found. These boundaries were then started once with the option IPR and once without but an external filter checking the structures for pentagons sharing an edge. The results agreed in all cases.

Timings

For the timings we repeatedly generated random sequences of 2s and 3s with the probabilities adjusted in a way that the expected number of 2s and 3s gives the value that corresponds to the given number of pentagons. Then these sequences are first filtered for those that have indeed the number of 2s and 3s necessary for the given number of pentagons. Afterwards these boundaries are filtered for those that allow at least one filling. This process was repeated until 500 fillable boundary sequences for every possible combination of boundary lengths 25, 50, 75,. . . ,250 and $0 \leq p \leq 5$ pentagons were found. Then we timed the execution time of a script starting the program once for each of the 500 sequences. All times are given for an Intel X3220 processor with 2.4GHz running GNU/Linux and are the user times given by the command *time*.

The average number of structures per boundary should only be interpreted as showing a certain tendency. The exact values will in some cases strongly depend on the random sample.

Of course these times may just be taken for what they are: not more than random samples

boundary length	0 pent.	1 pent.	2 pent.	3 pent.	4 pent.	5 pent.
25		1		5,7		129
26	1		2,24		22,2	
50	1		4,8		296	
51		1		37,4		6.972
75		1		92		61.967
76	1		7,8		1.754	
100	1		11,6		4.916	
101		1		193		316.495
125		1,01		319		560.594
126	1		14,4		12.547	
150	1		17,4		21.488	
151		1		576		2.559.607
175		1		975		2.956.581
176			21		34.222	
200			24		63.719	
201		1,01		998		7.648.245

Table 1: The (rounded) average number of structures per random fillable boundary. For the cases of 0 pentagons and length 176 or 200 the ratio of fillable boundaries is so small that the computational effort to find 500 random fillable boundaries would probably not be justified. Thus, we did not compute these numbers.

boundary length	0 pent.	1 pent.	2 pent.	3 pent.	4 pent.	5 pent.
25		2.083		9.829		214.873
26	2.500		7.000		48.239	
50	2.119		10.366		276.559	
51		3.049		62.307		487.122
75		2.155		43.095		172.169
76	2.000		11.529		104.040	
100	3.472		9.092		37.949	
101		1.927		20.457		68.149
125		1.633		10.121		35.429
126	2.976		5.623		18.268	
150	4.166		3.622		10.926	
151		1.459		5.595		14.606
175		1.715		3.529		14.291
176			2.283		7.586	
200			1.648		4.716	
201		1.352		2.324		

Table 2: The average number of structures per second for a given boundary length and a given number of pentagons.

boundary length	0 pent.	1 pent.	2 pent.	3 pent.	4 pent.	5 pent.
25		3,16		13,14		30,45
26	0,24		6,99		20,3	
50	0,015		1,85		7,35	
51		0,46		3,97		9,81
75		0,086		1,00		3,20
76	0,0014		0,37		1,67	
100	0,00024		0,08		0,46	
101		0,018		0,20		0,73
125		0,0037		0,050		0,21
126	0,000027		0,019		0,096	
150	0,0000048		0,0038		0,025	
151		0,00074		0,010		0,047
175		0,00016		0,0027		0,012
176			0,00082		0,0052	
200			0,00019		0,0015	
201		0,000028		0,00056		0,0026

Table 3: The percentage of fillable random boundary sequences corresponding to various numbers of pentagons and boundary lengths.

to give a rough impression of the performance of the algorithm. The generation rates vary **a lot** depending on the structure of the boundary and especially for long boundaries a random sample of 500 fillable boundaries can hardly be regarded as representative. But the fact that for long boundaries it is very time consuming to find a large number of fillable boundaries **at random** made it impossible to use much larger data sets. The ratio of fillable boundaries among the randomly generated ones as it occured during this search is displayed in Table 3.

The times for detecting non-fillable boundaries have been close to zero for all random cases tested. Even for 5000 random unfillable boundaries of length 300 (not including trivially unfillable boundaries with a sequence of five or more 2s) with the number of 2s and 3s implying between 0 and 5 pentagons (in this case that means 0,2, or 4) the total time was only 4 seconds.

With techniques like in [4] one can construct examples where the running time of the program is exponential in the length of the boundary and just detects that it is not fillable or constructs only a constant number of structures. But these cases are on one hand very special and on the other hand the constants are very small, so that it would take chemically unrealistically large boundaries before the running time for boundaries with few or no fillings becomes a problem. So we are optimistic that the program is fast enough for all applications that occur in the context of chemistry.

The program can be obtained from the authors or as part of the software package CaGe (see [3]) where it comes with a user friendly interface.

References

[1] J. Bornhöft, G. Brinkmann, and J. Greinus. Pentagon-hexagon-patches with short boundaries. *Eur. J. Combin.*, 24:517–529, 2003.

[2] G. Brinkmann and P.W. Fowler. A catalogue of growth transformations of fullerene polyhedra. *J. Chem. Inf. Comput. Sci.*, 43:1837–1843, 2003.

[3] G. Brinkmann, O. Delgado Friedrichs, A. Dress, and T. Harmuth. Cage – a virtual environment for studying some special classes of large molecules. *MATCH Commun. Math. Comput. Chem.*, 36:233–237, 1997.
corresponding program at `http://www.mathematik.uni-bielefeld.de/~CaGe`.

[4] G. Brinkmann, O. Delgado Friedrichs, and U. von Nathusius. Numbers of faces and boundary encodings of patches. In *Graphs and Discovery*, volume 69 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 27–38, 2005.

[5] G. Brinkmann and B.D. McKay. Fast generation of planar graphs. *MATCH Commun. Math. Comput. Chem.*, 58:323–357, 2007.
corresponding program at http://cs.anu.edu.au/~bdm/index.html.

[6] G. Brinkmann, U. von Nathusius, and A.H.R. Palser. A constructive enumeration of nanotube caps. *Discrete Appl. Math.*, 116:55–71, 2002.

[7] G. Caporossi and P. Hansen. Enumeration of polyhex hydrocarbons to h=21. *J. Chem. Inf. Comput. Sci.*, 38:610–619, 1998.

[8] M. Deza and M. Dutour Sikirić. *Geometry of Chemical Graphs*. Cambridge University Press, 2008.

[9] M.S. Dresselhaus, G. Dresselhaus, and P.C. Eklund. *Science of Fullerenes and Carbon Nanotubes*. Academic Press, 1996.

[10] M. Endo and H.W. Kroto. Formation of carbon nanofibers. *J. Phys. Chem.*, 96:6941–6944, 1992.

[11] P.W. Fowler G. Brinkmann and C. Justus. A catalogue of isomerisation transformations of fullerene polyhedra. *J. Chem. Inf. Comput. Sci.*, 43:917–927, 2003.

[12] M. Dutour Sikirić, M. Deza, and M. Shtogrin. Filling of a given boundary by p-gons and related problems. *Discrete Appl. Math.*, 156:1518–1535, 2008.

[13] A.J. Stone and D.J. Wales. Theoretical studies of icosahedral c_{60} and some related species. *Chem. Phys. Lett.*, 128:501–503, 1986.