Efficient Generation of Stable Planar Cages for Chemistry

Dominique Barth, Olivier David, Franck Quessette, Vincent Reinhard, Yann Strozecki, Sandrine Vial*

Université de Versailles Saint-Quentin

Abstract. In this paper we describe an algorithm which generates all possible planar maps with a good minimum sparsity from simple motifs and rules to connect them. A real implementation of this algorithm is available and is used by chemists who want to quickly generate all sound molecules they can obtain by mixing some basic components.

Bien insister sur le fait que c'est un papier pratique carl aconf est plus générale

1 Introduction

Carbon dioxide, as well as methane can be absorbed by large organic cages [1]. These cages are formed by spontaneous assembly of small organic molecules, called motifs, bearing different reacting centres. The prediction of the overall shape of the cage that will be obtained by mixing the starting motifs is rather difficult, especially because a given set of reacting partners can in principle leads to very different cages. It is hence crucial for chemists to have an operating tool that is capable of generating the many shapes of cages accessible from predetermined molecular motifs. The algorithm we have implemented generates molecules that are much larger and less regular that what the chemists usually design by hand.

We modelize molecules by maps i.e. planar embeddings of planar graphs, as explained in Sec.2. The use of maps may seem unsuitable since they do not represent spatial positions. Though, planar maps are a good model for spherical topologies and the embedding capture the rigidity of the motifs. We must also be able to select the most relevant molecules among the huge number we generate. In Sec.3.4, we characterize what a "good" molecule is through graph parameters which are then used to filter the best molecules. The relevance of our modeling and of our parameters is validated by the results we obtain: All small molecules (5-10 motifs) we generate and consider to be good according to our parameters have been studied before by chemists. Some of the very regular molecules of medium size (10-15 motifs) we generate correspond to the largest cages chemists have ever produced. We also have produced cages of shape unknown to chemists that they now try to synthesize (see 4).

TODO: quote something and put examples in the paper

* Authors thank the French Labex CHARMMMAT for the financial support of this work and David Auger for fruitful discussions about the algorithms to do the folding step.

The aim of this paper is the **generation of all colored planar maps up to isomorphism** representing possible molecules obtained from a set of elementary starting motifs. As with all listing problems, one difficulty is to avoid to produce a solution several times. Moreover the number of solutions to list may grow exponentially with their size, it is here the case for all bases of motifs but the most contrived. The complexity of such enumeration problems must then take into account the number of produced solutions (see [2] for more details on enumeration).

We say that an algorithm is in *polynomial total time* if its complexity is polynomial in the number of solutions and polynomial in the size of the produced solutions. In our context, where the number of solutions is always exponential in their size, we are interested in *linear total time* algorithms. The best algorithms are in constant amortized time (CAT): the algorithm uses on average a constant time to generate each solution. This kind of efficient algorithm exist for simple enumeration problems such as listing all trees [3]. We may also want to bound the delay that is the time between the production of two consecutive solutions. Good algorithms have a delay polynomial, linear or even constant in the size of the generated solutions.

There exist numerous works on enumeration and generation of planar maps [4], but none of them deals with the generation of planar maps built with a set of starting motifs and color constraints. Moreover, most of the literature deal with non-constructive tools[5] or yields algorithms which are not in polynomial total time. There are a few programs such as plantri [6] and CaGes [7] which generate efficiently some particular class of planar graphs such as cubic graphs or graphs with bounded size of face but they are not general enough for our purposes.

The algorithm we present in Sec. 3 is far to be in polynomial total time since we are not able to bound the number of isomorphic solutions we generate. However, we will present several subroutines used in our algorithm which are either CAT, for instance the generation of paths and almost foldable paths in Sec. 3.1, or in linear delay such as the folding of unsaturated maps of motifs in Sec. 3.2. Note that we express the complexity of our problem with regards to the size of the maps we want to generate, while other parameters (detailed in the modeling section) such as the size of the alphabet $\mathcal{A}$, the size of the base of motifs $\mathcal{M}$, the degree of the motifs are assumed to be small constants. Sec. 4 presents numerical results on the sizes obtained in a reasonable computation time. A comparison between our different algorithms is given.

2 Modeling of the problem

In this section, we propose the modeling of our problem by graphs. The graph of a molecular cage is constructed from smaller graphs that model the basic elements. All the graphs used in this paper are *vertex-colored maps*. A map is a connected planar graph drawn on the sphere considered up to continuous deformation. Note that by Steinitz's theorem, when a planar graph is 3-connected, there is only one corresponding map, but otherwise there may be several of them.The

representation of a map is a graph and a cyclic order of the neighbors around each vertex.

We first model the basic chemical elements with maps we call **motifs**. Then the motifs are assembled to form a **map of motifs** and from this map we derive a **molecular map** that is a more faithful model of the molecular cages we try to design.

We use an even set of colors $\mathcal{A} = \{a, \overline{a}, b, \overline{b}, c, \overline{c}, \dots\}$ where each positive color a in $\mathcal{A}$ has a unique complementary negative color denoted by $\overline{a}$ and a is the complementary color of $\overline{a}$. Each color represents a different kind of reacting center. Let us give the definition of motifs.

Definition 1. *A map $G = (V_c \sqcup V, E, next)$ is a **motif** if, (1) V_c contains only one vertex c called the center, (2) each vertex in V is colored with a color in $\mathcal{A}$, (3) $E = \{(c, u),\ u \in V\}$, and (4) next gives an order on the edges of c: $next((c, u)) = (c, v)$ means that the edge (c, v) is "following" the edge (c, u) in a clockwise drawing of G. For all $k < |V|$, $next^k((c, u)) \neq (c, u)$ and $next^{|V|}((c, u)) = (c, u)$.*

Note that a motif is a star graph. We assume as input $\mathcal{M}$ a finite set of motifs all different, each motif is identified by a distinct color from an alphabet $\mathcal{A}_M$ disjoint from $\mathcal{A}$. Fig. 1 gives examples of motifs.

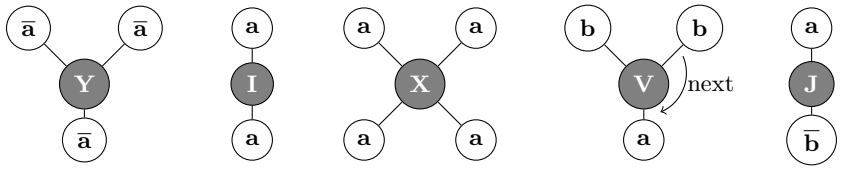


Fig. 1. Example of motifs on $\mathcal{A}_M = \{\mathbf{Y}, \mathbf{I}, \mathbf{X}, \mathbf{V}, \mathbf{J}\}$ and $\mathcal{A} = \{a, \overline{a}, b, \overline{b}\}$. In this example, $\text{next}((\mathbf{V}, b)) = (\mathbf{V}, a)$.

Definition 2. *A connected planar map $G = (V_c \sqcup V, E, next)$ is a **map of motifs** based on $\mathcal{M}$ if, (1) each vertex in V is connected to exactly one vertex in V_c, (2) each vertex in V is connected to at most one vertex in V. If u and v in V are connected, the color of u and v must be complementary and (3) next gives an order of the edges of each vertex in V_c. When all edges between vertices in V are removed, the remaining connected components must all be motifs of $\mathcal{M}$.*

*The number of connected components is called the **size** of G.*

Note that each motif of $\mathcal{M}$ may appear several times in a map of motifs, it may also be not present. Note also that a motif is a map of motifs of size 1. In a map of motif, a vertex of degree 1 in V is called a **free vertex**. A map of motifs with no free vertex is called **saturated** otherwise it is called unsaturated.

In all the paper, we assume when needed that an ordering of the edges around the elements of V_c consistent with next has been fixed.

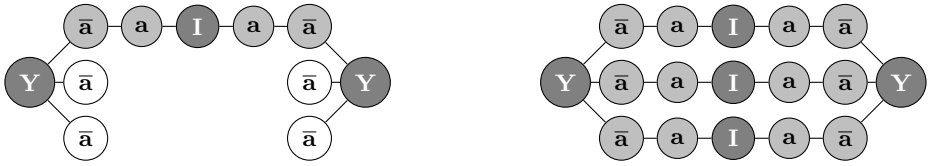


Fig. 2. Example of two maps of motifs based on $\mathcal{M} = \{\mathbf{Y}, \mathbf{I}\}$, the first map is unsaturated while the second map is saturated.

Based on a saturated map of motifs we construct the molecular map that is the graph model of the cages.

Definition 3. *Let $G = (V_c, V, E_G, next_G)$ be a saturated map of motifs based on $\mathcal{M}$, we define the* ***molecular map*** *M as the map G where all paths of size two between vertices of V_c are replaced by an edge.*

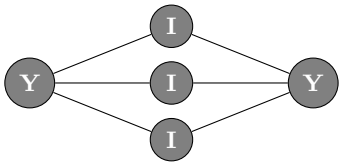


Fig. 3. The molecular map corresponding to the saturated map of motifs in Fig. 2

3 Description of the algorithm

In this part we describe the important steps of the algorithm and we provide several variants that we compare in terms of complexity, real performance and number of relevant solutions generated on instances of medium size.

We mainly use two basic operations. The first one, *the concatenation*, consists in adding edges between complementary vertices of two maps of motifs in such a way the result is still a map of motifs. In this paper, we always concatenate a single motif to a map of motifs. The second operation, *the fold or folding*, consists in adding an edge between two complementary vertices of a map of motifs, in such a way the result is a map of motifs. Sec. 3.1 presents different strategies of concatenation. Sec. 3.2 presents an efficient approach to folding. In Sec. 3.3 we explain how we detect and discard isomorphic copies of the same graph. Finally in Sec. 3.4, we introduce the indices which characterize a good molecular map and explain how we compute it.

3.1 Backbone generation

The first step is to generate all *backbones*, that is unsaturated maps of motifs of a given size n which are of a very simple shape. The aim is that, by folding these

backbones in a second step, we will recover all saturated maps of motifs. Since every map of motifs have a spanning tree, we can choose trees as backbones and be sure to recover all saturated maps. But for performance reason, we will also use path and cycles as backbones. We will also try to generate only the backbones which can be folded into some saturated maps. In our implementation, generating trees and even paths may be the bottleneck thus it is crucial to choose a set of backbones which is as small as possible and which can be generated with as little redundancies as possible.

Spanning tree In a first version of our algorithm [8], the set of non isomorphic trees of size n was explicitly stored. To produce the set of trees of size $n+1$, a single motif of every possible color is concatenated to any free vertex of the tree. This clearly generates all trees of size $n+1$, but the drawback is that some trees are generated several times. This problem is important since we then have to do an isomorphism test on every generated tree and because it makes the generation itself to be not in linear total time. We have slightly improved this part so that we generate trees on the fly with linear delay between two outputs.

One problem is that we generate a tree as many times as its number of edges: one for each choice of a vertex as root and for each choice of first edge of this root. A way to improve this would be to use ideas from CAT algorithms [9, 3] which are able to generate all unrooted trees. The main idea is to choose as root the centroid of the tree. However we have to deal with a second and harder problem: we generate maps of motifs and their vertex are colored. We can easily generate all maps of motifs sharing the same underlying tree but they may turn out to be be isomorphic.

donner un exemple ? IV1V2 et IV2V1

Hamiltonian paths Since generating trees is not easy, we propose to use simpler objects as backbones, here maps of motifs such that all vertices of V_c are on a path. These maps are caterpillar trees, but since the element of V_c on the central path entirely determine the elements at distance one, we will consider them as paths and call them so. There are two advantages to generating paths instead of trees: they are easier to generate and their number is smaller. The drawback is that not any planar graph has an Hamiltonian path, therefore we could miss some planar maps in our enumeration. However, most small planar graphs have an Hamiltonian path, for instance all planar cubic 3-connected graphs of size less than 38 [10] and, if Barnette's conjecture holds, all fullerene graphs.

The uniformity of the vertices, that is the fact that they have all the same degree, crucially matters in the existence of an Hamiltonian path. Consider for instance the base of motifs $\mathcal{M} = \{\mathbf{I}, \mathbf{Y}\}$ from Fig. 1. All molecular maps based on $\mathcal{M}$ are bipartite graphs: the $\mathbf{I}$'s in one set of the bipartition and the $\mathbf{Y}$'s in the other. But in saturated maps of motifs, we have twice the number of $\mathbf{Y}$ equal three times the number of $\mathbf{I}$ because all vertices in V must be connected, therefore there are no Hamiltonian path except in graphs with exactly three $\mathbf{I}$ and two $\mathbf{Y}$. This problem can be easily solved by building from $\mathcal{M}$ a new base of motifs which in the end generates the same molecular maps. The general method is to concatenate the motifs of degree 2 in every possible way to the other motifs

and to delete it. From our example $\{\mathbf{I}, \mathbf{Y}\}$, we obtain a base $\{\mathbf{Y}_0, \mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3\}$ where the $\mathbf{Y}_i$ are of degree 3 and have i vertices of V labeled by $\bar{a}$ and the others by a. If we now generate all molecular maps of size n based on $\{\mathbf{Y}_0, \mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3\}$ it is easy to convert them into molecular maps based on $\mathcal{M}$. The converted maps are of size exactly $\frac{5}{2}n$ since there are $\frac{3}{2}n$ $\mathbf{I}$ for each $\mathbf{Y}$.

Let us now explain how we generate all paths based on a set of motifs $\mathcal{M}$. We first build for each letter $a \in \mathcal{A}$ a list L_a of all non isomorphic motifs whose first edge is incident to a vertex of label $\bar{a}$. This data structure allows us to have a complexity independent of the size of $\mathcal{M}$ and of $\mathcal{A}$. Since they are considered to be of constant size, it is mostly a practical optimization. Then to build all possible paths of size $n+1$ from a path of size n, we consider its last vertex $c \in V_c$ and for each of the free vertex v connected to c and of color a, we attach every motif of L_a. Remark that beginning by the empty path, we generate all possible paths of a given size by applying recursively the algorithm. If we consider the paths as rooted at the first vertex, every path generated is clearly different. However, we can also consider the last vertex as the beginning of the path, which means we generate every path but the palindromes twice. To avoid that, we put an ordering on the colors and we consider the sequence of colors in a path. If the sequence of colors from the beginning to the end is lexicographically larger than the sequence from the end to beginning we output the path otherwise we do not. This can be implemented by a test in contant time in our algorithm.

Proposition 1. *The previous algorithm produces all maps of motifs which are paths without redundancies and in constant amortized time, when in the base of motifs no two motifs of degree* 2 *can be concatenated.*

Proof. The tree of recursive calls of our algorithm can always be seen as of degree at least 3 by merging nodes of degree 2 to nodes of degree larger. Therefore it has as many internal nodes as leaves which correspond to outputted solutions. Since the algorithm needs only a constant time to go from one node to another, the generation of all paths can be done in constant amortized time. □

In our practical examples, there are never motifs of degree two which can be concatenated.

Hamiltonian cycles If we want to further restrict the backbones we generate, a simple idea is to consider cycles instead of paths. Again it is a good choice if all motifs have the same degree or can be made so, since for instance all planar cubic 3-connected graphs of size less than 23 have an Hamiltonian cycle [11]. Moreover, we will only generate 2-connected graphs and not the ones which are only 1-connected. It is a desirable side-effect, since those graphs have a bridge they are always the worse for the two main indices we are interested with, i.e. the minimum sparsity and the size of the largest cycle (see Sec. 3.4). In our implementation, we obtain the cycles by generating every path and by connecting their beginning to their end when possible. This is not optimal since the same cycle can be obtained from several different paths (at most as much as its number of vertices).

Almost foldable backbones In each backbone we build, all free vertices will eventually be folded to get a saturated map of motifs. A simple necessary condition on the colors of a saturated map of motifs is that for each letter $a \in \mathcal{A}$, there are as many vertices in V labeled by a and $\bar{a}$. A backbone which satisfies this condition is said to be *almost foldable*. Let G be a map of motifs and let $a_1, \ldots, a_k$ be the positive letters of the alphabet $\mathcal{A}$. We denote by C_G the characteristic vector of$G$, it is of size k and its i^{th} component is the number of elements in V labeled by a_i minus the number of elements labeled by $\bar{a}_i$. Note that a map G is almost foldable if and only if C_G is the zero vector.

We propose here a method to generate in constant amortized time only the almost foldable paths. We introduce a function $F : \mathbb{N} \times \mathcal{A} \times \mathbb{Z}^k \to 2^{\mathcal{A}}$ which has the following semantic : $a' \in F(n, a, (c_1, \ldots, c_k))$ if and only if there is a path P of size n with a free vertex in the first motif labeled by a, such that $C_P = (c_1, \ldots, c_k)$ and such that a vertex of the last motif is labeled by a'.

We can easily use this function in our path generation algorithms to generate only the almost foldable paths. Assume we have generated a path P of size n' and we want to add a node at the end by connecting it to a node of label a. Assume we have already computed C_P. To decide whether P can be extended into a path of size n which is almost foldable, we only have to check if $F(n-n', \bar{a}, -C_P) \neq \emptyset$ otherwise, the extension will necessarily lead to a non almost foldable path of size n.

Proposition 2. *The algorithm we have described enumerates all almost foldable paths in constant amortized time plus a precomputation in $O(n^{k+1})$.*

Proof. First remark that the proposed method only add a single test at each step of the algorithm to generate all paths and thus do not increase its complexity.

We now explain how to generate all needed values of the function F in time $O(n^{k+1})$ by dynamic programming. Denote by f the maximal number of vertices in a motif labeled by the same color. For a path P of size n, it is clear that the coefficients in C_P are all in the interval $[-nf, nf]$. Therefore, to generate paths of size n, since f and the size of $\mathcal{A}$ are constants, we need to store $O(n^{k+1})$ values of F only.

F is easy to compute for $n = 1$: we consider each motif $M \in \mathcal{M}$ and each v of label a in M, and set $F(1, a, C_M)$ to be the set of labels of all vertices of M but v. Assume we have generated the value of F for n, we generate the values for $n + 1$ in the following way. For each a, C and each $a' \in F(n, a, C)$, we consider all motifs $M \in \mathcal{M}$ such that one of their vertex is labeled by $\bar{a}$. We add all the labels of the other vertices to the set $F(n, a, C + C_M)$. This algorithm only does a constant number of operations for each value of F it computes, therefore its complexity is $O(n^{k+1})$. □

The complexity of the precomputation may seem to be large but k must be seen as a small constant (less than 4). It is negligible with regard to the generation of paths, which is exponential in n because of the number of non isomorphic paths. In practice, the precomputation takes only a few milliseconds for size of

graphs up to 40. Moreover this optimization makes the time to computes all the backbones very small with regards to the folding step.

The same kind of method has been implemented for trees. Although when we extend a tree by a concatenation, it can be through any vertex. To keep the same dynamic algorithm as for path we should track all free vertices, which would make the algorithm exponential time. There are two solutions to this problem, the first and the one we have implemented is to compute a multidimensional array M such that $M(n, C) = 1$ if there is a *forest* F of size n such that $C_F = C$ and $M(n, C) = 0$ otherwise. We can thus test in our algorithm generating trees, whether any partial tree can be extended to a structure of the right size by a forest. Since we generate trees and not forests, we will sometimes expand a partial tree and obtain no almost foldable backbone in the end.

The second solution is to change the characteristic vector of a backbone so that each of its component is the number of free vertices of some color positive or negative. In this way it is easy to compute an array M such that $M(n, C) = 1$ if there is a *tree* T of size n such that $C_T = C$ and $M(n, C) = 0$ otherwise. Indeed, for each motif M with a free vertex of color a, if for some C $M(n, C) = 1$ and C has a non-zero component $\bar{a}$ then there is a tree of size $n+1$ with vector $C+C_M$ that is $M(n, C+C_M) = 1$. The only drawback is that the size of M and thus the complexity of the precomputation is $O(n^{2k+1}$, where k is the number of positive colors while the size of M in the solution we have implemented is $O(n^{k+1}$.

3.2 Folding of the backbones

Let G be a map of motifs, the *fold* operation on the vertices u and v is adding the edge (u, v) to G. The operation is valid if u and v are free, of complementary colors and in the same face of G. Therefore, the graph obtained after the fold is still a map of motifs. In this part we want to generate from a backbone, by sequences of folds, all possible saturated maps of motifs.

The *outline* of a face is the list in order of traversal of the free vertices. An outline is a circular sequence of vertices $(v_1, \ldots, v_n) \in V^n$. Sequence means that the order is significant and circular means that the starting point is not. For instance, (v_1, v_2, v_3) and (v_3, v_1, v_2) are the same circular sequence but are different from (v_3, v_2, v_1). The color of an outline $(v_1, \ldots, v_n)$ is the word $w_1 \ldots w_n$ with w_i the color of v_i. Folding two vertices v_i and v_j is equivalent to transforming the word $W_1 w_i W_2 w_j W_3$ into the two words $W_1 W_3$ and W_2. This operation is called a reduction on the words. Remark that the foldable words form a language which is a variant of the classical Dyck language of balanced string parentheses. As in the case of parentheses languages, we can restrict the reduction to consecutive complementary letters which transforms $W_1 a\bar{a} W_2$ into the single word $W_1 W_2$.

Remark that a backbone (a tree or a path) has a single outline and that a saturated maps has an empty color word. Therefore applying a sequence of fold to a backbone to get a saturated map is the same as applying a sequence of reductions to the colors of an outline so that we obtain only empty words. We work from now only on the words $w_1 \ldots w_n$ and on reduction sequences.

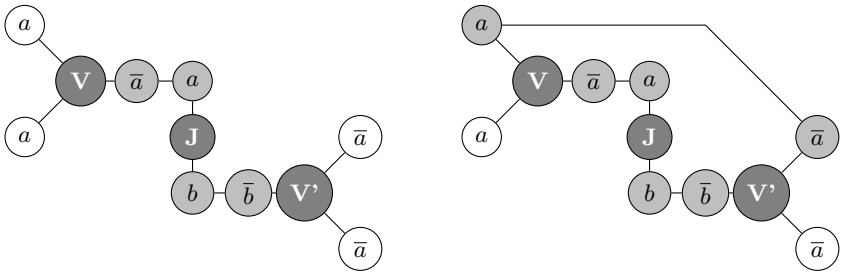


Fig. 4. A map of three motifs on $\mathcal{A}_M = \{\mathbf{V}, \mathbf{V}', \mathbf{J}\}$ and its outline before and after a fold operation.

Let us call a word (or a multiset of words) which reduces to a multiset of empty words a *foldable word*. It is clear that when a word is foldable, it can be reduced to an empty word using the reduction of consecutive letters only by reordering the different reductions.

Lemma 1 (Folklore). *The restricted reduction on words is confluent i.e. each sequences of restricted reduction starting from a foldable word can be extended so that we get an empty word.*

Proof. To prove our lemma, it is enough to prove that if S is the sequence which reduces a word $W = W_1 w_i w_{i+1} W_2$ with $w_i = \overline{w_{i+1}}$, then $W_1 W_2$ is foldable. If S pairs w_i and w_{i+1}, then $W_1 W_2$ can be reduced to the empty word by S. We now assume that S pairs w_i with w_k and w_{i+1} with w_l, where $W = W_1^1 w_k W_1^2 w_i w_{i+1} W_2^1 w_l W_2^2$. Inside the sequence S, we can find subsequences S_i^j which reduce W_i^j since we are allowed to reduce consecutive letters only. Therefore we can reduce $W_1 W_2 = W_1^1 w_k W_1^2 W_2^1 w_l W_2^2$, by first using the sequences S_i^j to obtain the word $w_k w_l$ which can be reduced in one step. □

As a consequence of this lemma, we get a simple algorithm for testing whether a word is foldable: reduce the word as long as it is possible and if an empty word is obtained, the word is foldable. This algorithm can be implemented in linear time. We use this algorithm each time we produce a backbone to test whether it can be folded into a saturated map of motifs.

We call *result* of a sequence of reductions the set of pairs (i, j) such that the sequence has paired i and j. Our aim is to generate all different results of sequences of reductions on foldable words without redundancies. For a given word W of size n, we first build the lists L_i which contain the set of index $j > i$ such that w_i can be folded with w_j and obtain a set of words still foldable.

The lists L_i are built from a boolean matrix M such that $M_{i,j}$ is true if and only if the word $w_i \dots w_j$ is foldable. The matrix is computed by dynamic programming: $M_{i,i+1}$ is true if and only if w_i and w_{i+1} are complementary. We compute $M_{i,j}$ once we have computed all $M_{i',j'}$ such that $(j' - i') < (j - i)$ by

using the fact that $w_i \dots w_j$ is foldable if and only if $w_i \dots w_k$ and $w_{k+1} \dots w_j$ are foldable for some k in $[i+1, j]$. This algorithm build the matrix M in time cubic in the size of the word.

Remark that a sequence of reduction applied to a word W yields a set of subwords which are consecutive letters of W. Therefore we can represent the results of several reductions by a set of pairs $\{(l_1, r_1), \dots, (l_k, r_k)\}$ with (l_i, r_i) representing the word $w_{l_i} \dots w_{r_i}$ and $r_i < l_{i+1}$. We now build the results of sequences of reductions in a recursive way. Assume we have already built a result R through the use of a sequence of reductions to W, which have produced a set $\{(l_1, r_1), \dots, (l_k, r_k)\}$. We consider l_1, the index of the first letter which has not been reduced and we do the reduction with every possible letter of index $i \in [l_1, r_1] \cap L_{l_1}$ which produces the set $\{(l_1, i), (i+1, r_1) \dots, (l_k, r_k)\}$ and the result $R \cup \{(l_1, i)\}$. By using recursively this algorithm starting on W, we obtain all possible results R which yield a multiset of empty words. It is not possible to generate twice a result since at any point of the algorithm we make recursive calls on $R \cup \{(l_1, i)\}$ for different values of i which makes the results produced by each call disjoints. Between two recursive calls we do only a constant number of operations, therefore the delay is bounded by the depth of the tree of recursive calls, that is the size of the word W.

Proposition 3. *The algorithm we have described enumerates all distinct results of sequences of reduction on a foldable word, with a linear delay and a cubic precomputation.*

3.3 Dealing with isomorphic copies

Since the construction process does not guaranty uniqueness of the generated maps, we need to detect isomorphic copies of already generated maps to discard them. To this aim, we sketch here an efficient quadratic isomorphism test between two saturated maps of motifs in the spirit of [12].

From a theoretical point of view, planar isomorphism is well understood since it has been proved to be solvable in linear time [13] and logarithmic space [14]. However those algorithms are not practical and hard to implement as observed in [15]. This is especially true for our small graphs of size about 20, which is the reason why we rely on a simpler algorithm of quadratic complexity.

Let us define a quadratic isomorphism algorithm as follows. All the signatures are numbers in a base B with $B = n + |\mathcal{A}| + |\mathcal{A}_M|$. The first step that is common to all the maps of motifs of the same size n is to assign to each color in $\mathcal{A}$ and $\mathcal{A}_M$ a different digit in $[n, n + |\mathcal{A}| + |\mathcal{A}_M|)$ in base B. In a map of motifs $G = (V_c, V, E, \text{next})$ of size n and for any edge $(c, u) \in E$ with $c \in V_c$ we perform a deterministic depth first search that will define the signature of G starting at (c, u). Since signatures are numbers, they can be easily compared and *the signature* of G will be the minimum number over all starting points.

For computing a signature starting at (c, u), at first visit of each vertex in V_c assign an index number that is a digit in the range $[0, n)$ in the base B. From c visit its neighbor u: since the map is saturated u is connected to a vertex $v \in V$

and v is connected to a vertex $c' \in V_c$. Construct the signature by concatenating the index number of c, the digits of the colors c, u, v and c' and the index number of c'. If c' is already visited backtrack else continue the visit starting at (c', v') with $(c', v') = \text{next}((c', v))$ and so on until all vertices are visited once. At the end, we obtain a signature in base B for the starting point (c, u). Note that the signature itself is of size linear in n.

We make a simple optimization, which are crucial, since profiling our algorithm reveals that it spends more than half of its time computing signatures. We assign the lower digits to the colors of c and u such that the number of couples (c, u) with c the center of a motif is minimal and non zero. Since the signatures are constructed with the most significant bit first, during the construction of a signature, we test for each digit added if the signature is at this point greater than the minimal one. Thus we can cut very efficiently in the signature calculation process.

To be able to test whether a map has been already produced, we must store all already produced maps. Since the number of these maps generally grows exponentially with their size, they are stored in a dynamic set structure which supports logarithmic addition and research of elements. In our implementation we have used an AVL whose key is the signature. Hence each time a new map is produced, we compute its signature and if this signature is already in the AVL, it is simply not inserted. Since the molecular maps need the expertise of the chemist to be validated it is useless to generate thousands of candidates. In the implementation one can limit the number of molecular maps that are in the output of the program. In case of limitation only the maps with the best minimum sparsity (see in the next section) are output.

3.4 Indices computed on the molecular map

A molecular map is a candidate to be a "good" cage for chemistry. The definition of a "good" cage is merely topological: the 3D shape must be close to a sphere, it must be tough to deformations and cuts and it must have an "entrance". We are able to check if a molecule satisfies or not these requirements only by considering the structure of its molecular map. The resistance is given by its minimum sparsity. We compute the number of automorphisms of the molecular map to measure the spherical property and also ask for the graph to be planar and connected. The entrance is given by the size of its largest face. All these assumptions are validated by the expertise of the chemists. Let us now describe each of these indices.

Sparsest cut A cut of a graph $G = (V, E)$ is a bipartition of V. The *size* of a cut $S = (S_1, S_2)$ is the number of edges with one end in S_1 and the other in S_2. The sparsity of a cut is $sparsity(S) = \frac{size(S)}{\min(|S_1|, |S_2|)}$. The Sparsest-Cut problem is to find the minimum sparsity over all cuts. We first implemented a brute-force algorithm, which using a Gray code, enumerates all possible partitions of the set of vertices. Since we were using a Gray code, the partition changes at each

step by only one element and the cut can be computed in constant time from the previous one. Therefore we have a simple algorithm with complexity $O(2^n)$ where n is the number of vertices in our graph, which is useful for n up to twenty.

Although computing the minimum sparsity is NP-complete in general (minimum cut into bounded set in [16]), when the graph is planar there is a polynomial time algorithm [17]. Since for a few bases of motifs, the time to compute the minimum sparsity was the limiting factor of our program, we have implemented this more complicated algorithm (which has not been done before as far as we know).

The main idea is that a cut in a graph corresponds exactly to a cycle in the dual graph (see [18] for graph definitions useful in this paragraph). A weight is associated to each cycle of the dual: if the corresponding cut in the primal partitions it into S_1 and S_2, the weight is $\min(|S_1|, |S_2|)$. From a spanning tree of the dual, we build a base of its fundamental cycles (a fundamental cycle is given by any edge not in the spanning tree completed by edges of the spanning tree to form a minimal cycle). From combinations of fundamental cycles, we can generate every cycle and its weight. For each edge in the dual, we build a graph such that some paths in this graph correspond to cycles which use the edge. Moreover, the weight of the cycle can be read in the last vertex of the path, and the size of the corresponding cut is the length of the path. Therefore, computing a single source shortest-path in each of this graph yields the optimal value. While in the original article this was done by a modified Dijkstra algorithm, we use a breadth first-search. This is faster and it enables us to use a good heuristic: at any point of one of the breadth first-search, we know the current distance from the source can only increase. We can the stop the search, if this distance divided by the maximal weight is larger than the current minimum sparsity value. This implementation has very good performance that we summarize in the following table. The time are mean time given in millisecond for maps of different sizes based on two motifs of degree 3 and one of degree 2.

Size	12	15	18	21
Exponential algorithm	0.03	0.2	1.6	12.5
Polynomial algorithm	0.05	0.08	1.1	1.7

Distribution of the sizes of faces The faces size is an important parameter in the cage construction. The chemist wants a cage with an "entrance". In graph terms we seek for graphs with one large face and all the others faces of size around the mean size, which makes the molecule more spherical in practice. The distribution of the face size is straightforward to compute. As an indicator we compute the size difference between the two largest faces divided by the mean size. This indicator is zero when there is two largest faces with the same size and grows with the entrance size.

Equivalence classes of the vertices Two vertices (motifs) of a molecular map are in the same class if it exists an automorphism that send one to the other. For each motif of $\mathcal{M}$ present in the molecular map we compute the number of equivalent classes regarding the automorphisms. The less the number of classes the more the molecule has a spherical shape.

From a large set of experiments, these three indices have proved to be realistic to the chemist.

4 Results

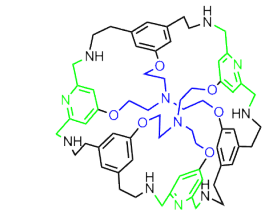
The code and the exhaustive results of our approach can be found at the following address `http://kekule.prism.uvsq.fr`. For each set of motifs, one can find the set of generated maps in different format and their indices. The following tabular shows for a set of motifs and a size of maps, the number of maps and time needed to compute them using different algorithms.

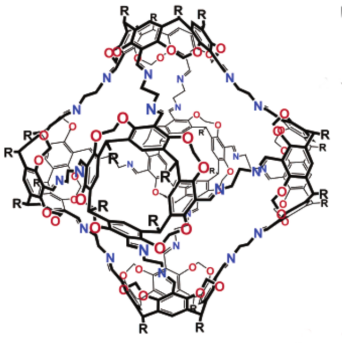
		Brute-Force		**Path**		**Cycle**	
$\mathcal{A}_M$	Size	# of maps	Time (s)	# of maps	Time (s)	# of maps	Time (s)
YVV	8	25	18	25	< 1	7	< 1
YVV	16	–	–	24281	49	4269	19
JYV	12	38	207	36	< 1	8	< 1
JYV	20	–	–	2374	7	144	< 1

The number of maps using the brute-force approach is the exact number of maps that can be generated. Path and cycle approach does not necessarily generate this number of maps as it is shown in the table above.

TODO: -comparaison des différentes générations de backbones avec commentaire -exemple sur plusieurs motifs avec les plus grandes valeurs atteintes avec commentaire sur ce qui ne peut être atteint à cause de l'explosion combinatoire ou à cause de la mauvaise qualité de notre approche -des exemples avec concaténation pour montrer l'intérêt de l'approche -un exemple de chimie

Using the **XI** set of motifs if we take for each size of set the maps with the lowest cut indices, we find the compounds obtained by Warmuth[19] in real-life experiments.





References

1. Holst, J., Trewin, A., Cooper, A.: Porous organic molecules. Nature Chem. **2** (2010) 915–920
2. Strozecki, Y.: Enumeration complexity and matroid decomposition. PhD thesis, Université Paris Diderot - Paris 7 (2010)
3. Li, G., Ruskey, F.: The advantages of forward thinking in generating rooted and free trees. In: ACM-SIAM symposium on Discrete algorithms. (1999) 939–940
4. Liskovets, V.: Enumeration of nonisomorphic planar maps. Selecta Math. Soviet. **4** (1985) 304–323
5. Cori, R., Vauquelin, B.: Planar maps are labelled trees. Canadian Journal Math. **33**(5) (1981) 1023–1042
6. Brinkmann, G., McKay, B.D.: Fast generation of planar graphs. MATCH Commun. Math. Comput. Chem **58**(2) (2007) 323–357
7. Brinkmann, G., Friedrichs, O.D., Lisken, S., Peeters, A., Van Cleemput, N.: Cage—a virtual environment for studying some special classes of plane graphs—an update. MATCH Commun. Math. Comput. Chem **63**(3) (2010) 533–552
8. Barth, D., Boudaoud, B., Couty, F., David, O., Quessette, F., Vial, S.: Map generation for CO_2 cages. In: Computer and Information Sciences III. Springer (2013) 503–510
9. Beyer, T., Hedetniemi, S.M.: Constant time generation of rooted trees. SIAM Journal on Computing **9**(4) (1980) 706–712
10. Holton, D.A., McKay, B.D.: The smallest non-hamiltonian 3-connected cubic planar graphs have 38 vertices. Journal of Combinatorial Theory, Series B **45**(3) (1988) 305–319
11. Aldred, R.E., Bau, S., Holton, D.A., McKay, B.D.: Cycles through 23 vertices in 3-connected cubic planar graphs. Graphs and Combinatorics **15**(4) (1999) 373–376
12. Weinberg, L.: A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. Circuit Theory, IEEE Transactions on **13**(2) (1966) 142–148
13. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: ACM symposium on Theory of computing. (1974) 172–184
14. Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: Planar graph isomorphism is in log-space. In: Computational Complexity. (2009) 203–214
15. Kukluk, J.P., Holder, L.B., Cook, D.J.: Algorithm and experiments in testing planar graphs for isomorphism. Journal of Graphs Algorithms and Applications **8**(3) (2004) 313–356
16. Garey, M., Johnson, D.: Computers and intractability: a guide to NP-completeness. WH Freeman and Company, San Francisco (1979)
17. Park, J.K., Phillips, C.A.: Finding minimum-quotient cuts in planar graphs. In: ACM symposium on Theory of computing. (1993) 766–775
18. Diestel, R.: Graph theory. 2005. Grad. Texts in Math (2005)
19. Liu, X., Warmuth, R.: Solvent effects in thermodynamically controlled multicomponent nanocage syntheses. Journal of the American Chemical Society **128**(43) (2006) 14120–14127