Generating sound molecular cages

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

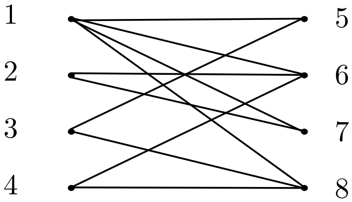
Université de Versailles St-Quentin-en-Yvelines
Laboratoire PRiSM

Novembre 2014, Séminaire de l'équipe Bamboo

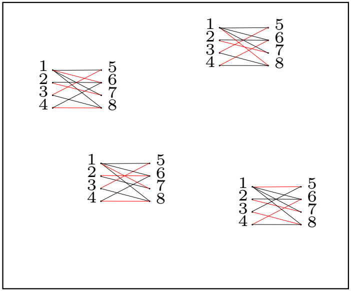
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

Perfect matchings:



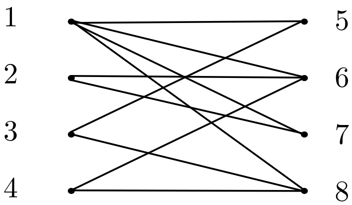
Solutions :



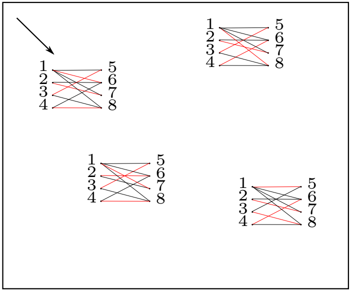
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

Perfect matchings:



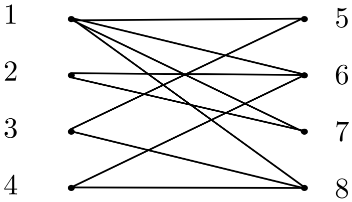
Solutions :



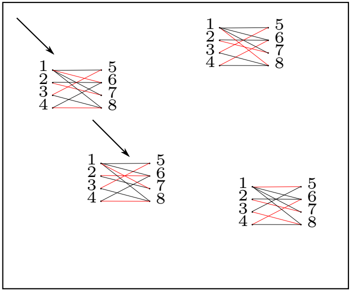
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

Perfect matchings:



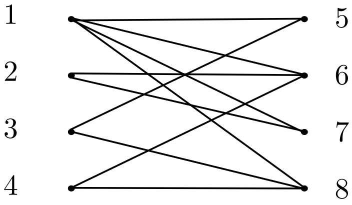
Solutions :



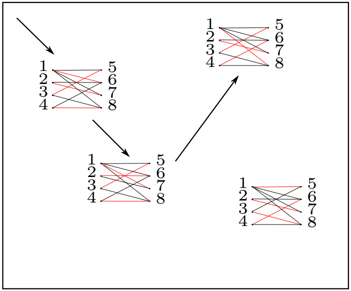
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

Perfect matchings:



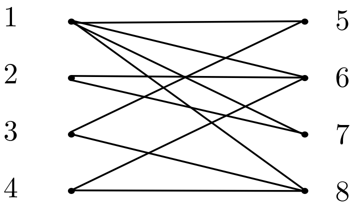
Solutions :



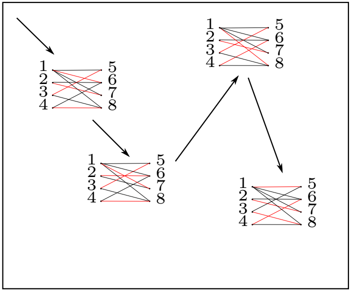
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.
- Complexity measures: total time and delay between solutions.

Perfect matchings:



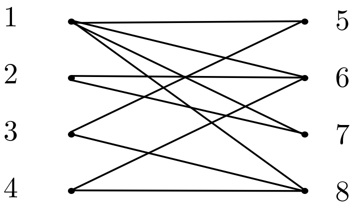
Solutions :



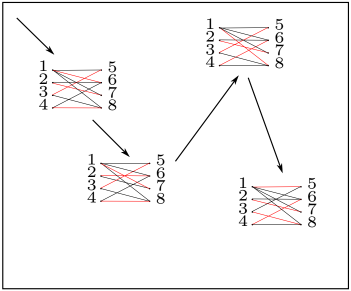
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.
- Complexity measures: total time and delay between solutions.
- Motivations: database queries, optimization, building molecular libraries.

Perfect matchings:



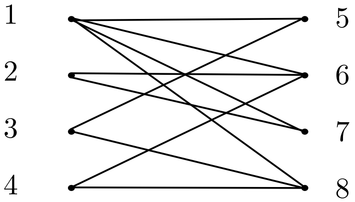
Solutions :



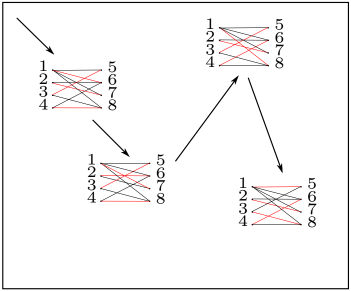
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.
- Complexity measures: total time and delay between solutions.
- Motivations: database queries, optimization, building molecular libraries.

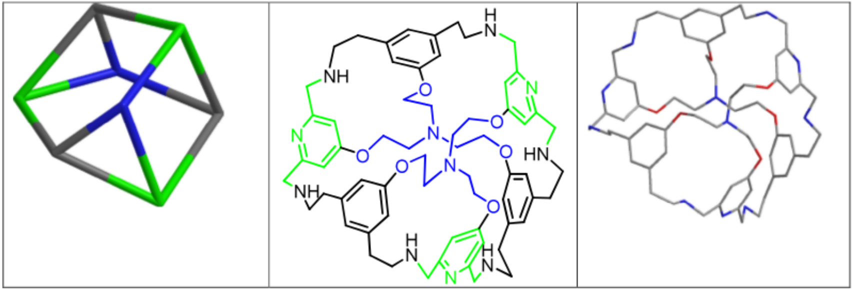
Perfect matchings:



Solutions :



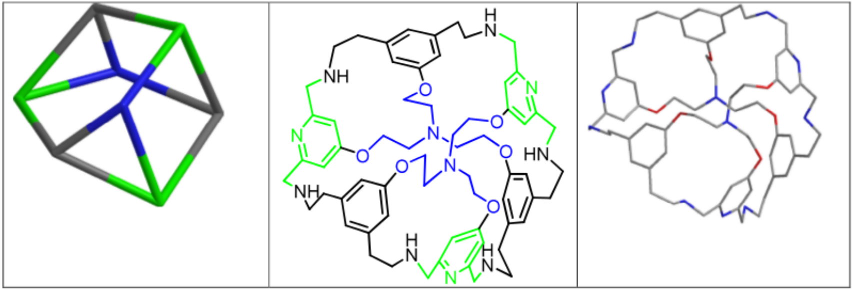
Introduction



Motivation: chemists (Olivier David) wants to build molecular cages.

But what kind of *nice cages* can be built from basic components ?

Introduction



Motivation: chemists (Olivier David) wants to build molecular cages.

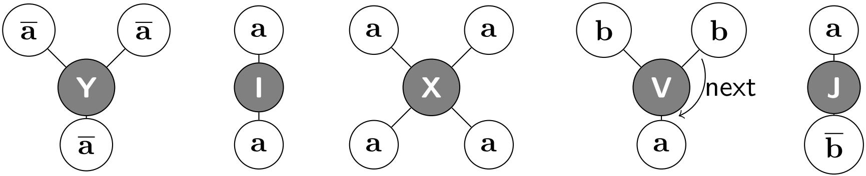
But what kind of *nice cages* can be built from basic components ?

The motifs

Definition

A map $G = (V_c, V, E, \text{next})$ 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 $\mathcal{A}$ a fixed alphabet
3. $E = \{(c, u),\ u \in V\}$
4. next gives an order on the edges of c



Map of motifs

Definition

A connected planar map $G = (V_c, V, E, \text{next})$ is a **map of motifs** based on $\mathcal{M}$ if,

1. each vertex in V is connected to at most one vertex in V, which is of the complementary colour.
2. when all edges between vertices in V are removed, the remaining connected components must all be motifs of $\mathcal{M}$

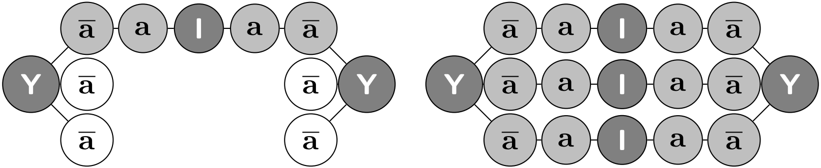


Figure: Example of two maps of motifs based on $\mathcal{M} = \{\mathbf{Y}, \mathbf{I}\}$, the first map is unsaturated while the second map is saturated.

Molecular map

Definition

Let $G = (V_\mathsf{c}, V, E_G, \mathsf{next}_G)$ be a saturated map of motifs based on $\mathcal{M}$, we define the **molecular map** $M = (V, E_M, \mathsf{next}_M)$:

1. $V = V_\mathsf{c}$
2. $(c_1, c_2) \in E_M$ if it exists a path (c_1, u, v, c_2) in G
3. $\mathsf{next}_M((c, c_1)) = (c, c_2)$ if it exists two paths (c, u_1, v_1, c_1) and (c, u_2, v_2, c_2) in G and $\mathsf{next}_G((c, u_1)) = (c, u_2)$

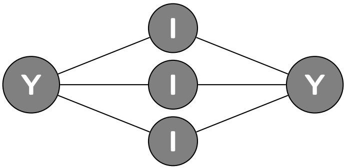


Figure: The molecular map corresponding to the saturated map of motifs in Fig. 1

The indices

Why is a molecular map a good representation of a molecula ?

1. Constraint on the edges: possible chemical connections
2. 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.

$$sparsity(S) = \frac{size(S)}{\min(|S_1|, |S_2|)}$$

Sound molecula have high minimum sparsity.

The indices

Why is a molecular map a good representation of a molecula ?

1. Constraint on the edges: possible chemical connections
2. 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.

$$sparsity(S) = \frac{size(S)}{\min(|S_1|, |S_2|)}$$

Sound molecula have high minimum sparsity.

The indices

Why is a molecular map a good representation of a cage ?

1. Constraint on the edges: possible chemical connections
2. 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.

$$sparsity(S) = \frac{size(S)}{\min(|S_1|, |S_2|)}$$

   Sound molecula have high minimum sparsity.

3. Planar graphs and large automorphism groups $\equiv$ spherical shape.

The indices

Why is a molecular map a good representation of a cage ?

1. Constraint on the edges: possible chemical connections
2. 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.

$$sparsity(S) = \frac{size(S)}{\min(|S_1|, |S_2|)}$$

   Sound molecula have high minimum sparsity.

3. Planar graphs and large automorphism groups $\equiv$ spherical shape.
4. A large face in the graph $\equiv$ an entrance in the cage

The indices

Why is a molecular map a good representation of a cage ?

1. Constraint on the edges: possible chemical connections
2. 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.

$$sparsity(S) = \frac{size(S)}{\min(|S_1|, |S_2|)}$$

   Sound molecula have high minimum sparsity.

3. Planar graphs and large automorphism groups $\equiv$ spherical shape.
4. A large face in the graph $\equiv$ an entrance in the cage

The problem

Enumeration problem

We want to generate, given a set of motifs $\mathcal{M}$ and a size n, all molecular maps based on $\mathcal{M}$ and of size n.

The number of maps is exponential in n. We would like to design an algorithm with a small delay or at least whose total complexity is linear in the number of outputs.

The problem

Enumeration problem

We want to generate, given a set of motifs $\mathcal{M}$ and a size n, all molecular maps based on $\mathcal{M}$ and of size n.

The number of maps is exponential in n. We would like to design an algorithm with a small delay or at least whose total complexity is linear in the number of outputs.

Is it possible to restrict the solutions generated to the ones with a large face? with a good minimum sparsity? a large automorphism group?

The problem

Enumeration problem

We want to generate, given a set of motifs $\mathcal{M}$ and a size n, all molecular maps based on $\mathcal{M}$ and of size n.

The number of maps is exponential in n. We would like to design an algorithm with a small delay or at least whose total complexity is linear in the number of outputs.

Is it possible to restrict the solutions generated to the ones with a large face? with a good minimum sparsity? a large automorphism group?

What is the meaning of my previous question?

The problem

Enumeration problem

We want to generate, given a set of motifs $\mathcal{M}$ and a size n, all molecular maps based on $\mathcal{M}$ and of size n.

The number of maps is exponential in n. We would like to design an algorithm with a small delay or at least whose total complexity is linear in the number of outputs.

Is it possible to restrict the solutions generated to the ones with a large face? with a good minimum sparsity? a large automorphism group?

What is the meaning of my previous question?

A three steps approach

1. Generate the backbones which are simple maps of motifs
2. From each backbone we compute all saturated maps of motifs we can obtain

A three steps approach

1. Generate the backbones which are simple maps of motifs
2. From each backbone we compute all saturated maps of motifs we can obtain
3. Compute the indices of the solutions generated

A three steps approach

1. Generate the backbones which are simple maps of motifs
2. From each backbone we compute all saturated maps of motifs we can obtain
3. Compute the indices of the solutions generated

Issue: a solution can be obtained several times. No guarantee on this number.

A three steps approach

1. Generate the backbones which are simple maps of motifs
2. From each backbone we compute all saturated maps of motifs we can obtain
3. Compute the indices of the solutions generated

Issue: a solution can be obtained several times. No guarantee on this number.

Our (bad) method: Store all solutions in a good datastructure (self balanced tree) and for each new solution test whether it has already been produced (isomorphism test).

A three steps approach

1. Generate the backbones which are simple maps of motifs
2. From each backbone we compute all saturated maps of motifs we can obtain
3. Compute the indices of the solutions generated

Issue: a solution can be obtained several times. No guarantee on this number.

Our (bad) method: Store all solutions in a good datastructure (self balanced tree) and for each new solution test whether it has already been produced (isomorphism test).

The less the steps, the better the algorithm!

A three steps approach

1. Generate the backbones which are simple maps of motifs
2. From each backbone we compute all saturated maps of motifs we can obtain
3. Compute the indices of the solutions generated

Issue: a solution can be obtained several times. No guarantee on this number.

Our (bad) method: Store all solutions in a good datastructure (self balanced tree) and for each new solution test whether it has already been produced (isomorphism test).

The less the steps, the better the algorithm!

The backbones

We generate different families of backbones. Their free vertices (of degree 1) will be folded to get a saturated map.

Different kind of backbones:

1. Trees

The backbones

We generate different families of backbones. Their free vertices (of degree 1) will be folded to get a saturated map.

Different kind of backbones:

1. Trees
2. Paths

The backbones

We generate different families of backbones. Their free vertices (of degree 1) will be folded to get a saturated map.

Different kind of backbones:

1. Trees
2. Paths
3. Cycles

The backbones

We generate different families of backbones. Their free vertices (of degree 1) will be folded to get a saturated map.

Different kind of backbones:

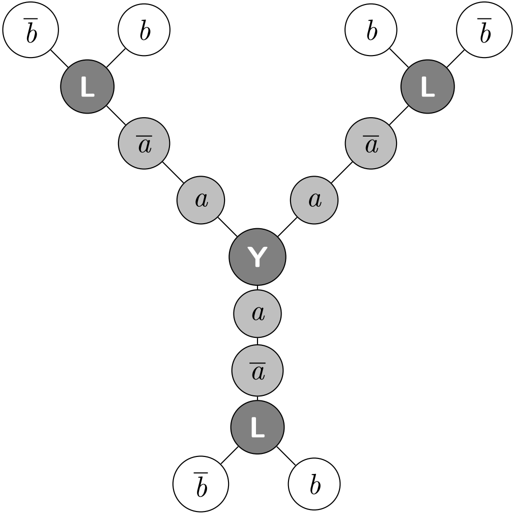
1. Trees
2. Paths
3. Cycles
4. All previous structures with restriction on their free vertices

The backbones

We generate different families of backbones. Their free vertices (of degree 1) will be folded to get a saturated map.

Different kind of backbones:

1. Trees
2. Paths
3. Cycles
4. All previous structures with restriction on their free vertices



$\overline{b}$
b
b
$\overline{b}$
L
L
$\overline{a}$
$\overline{a}$
a
a
Y
a
$\overline{a}$
L
$\overline{b}$
b

Trees

It's a good idea: Every connected map has a spanning tree, it will make the generation exhaustive.

It's a bad idea : A graph has many spanning trees.

Trees

It's a good idea: Every connected map has a spanning tree, it will make the generation exhaustive.

It's a bad idea : A graph has many spanning trees.

How we do it : a bruteforce method to generate rooted trees and an isomorphism test. Since we generate *planar trees*, it is quite efficient.

Trees

It's a good idea: Every connected map has a spanning tree, it will make the generation exhaustive.

It's a bad idea : A graph has many spanning trees.

How we do it : a bruteforce method to generate rooted trees and an isomorphism test. Since we generate *planar trees*, it is quite efficient.

Open question : Can we adapt optimal CAT algoritms to generate unrooted trees to those colored planar trees ?

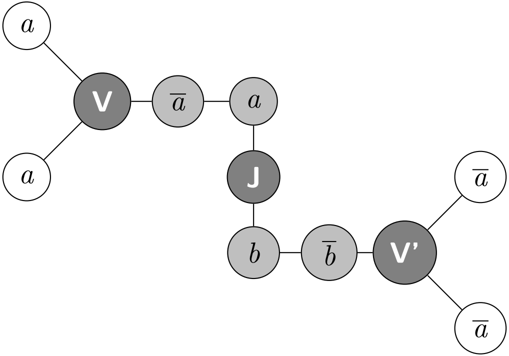
Trees

It's a good idea: Every connected map has a spanning tree, it will make the generation exhaustive.

It's a bad idea : A graph has many spanning trees.

How we do it : a bruteforce method to generate rooted trees and an isomorphism test. Since we generate *planar trees*, it is quite efficient.

Open question : Can we adapt optimal CAT algoritms to generate unrooted trees to those colored planar trees ?



a
a
V
$\overline{a}$
a
J
b
$\overline{b}$
V'
$\overline{a}$
$\overline{a}$

Paths

It's a bad idea : not every planar map has an Hamiltonian path. But all planar cubic 3-connected graphs of size less than 38 are Hamiltonian.

It's a good idea: paths are simpler than trees (smaller number).

Paths

It's a bad idea : not every planar map has an Hamiltonian path. But all planar cubic 3-connected graphs of size less than 38 are Hamiltonian.

It's a good idea: paths are simpler than trees (smaller number).

How we do it: bruteforce method, add at the end of a path any possible motif until the path is of the right size.

Paths

It's a bad idea : not every planar map has an Hamiltonian path. But all planar cubic 3-connected graphs of size less than 38 are Hamiltonian.

It's a good idea: paths are simpler than trees (smaller number).

How we do it: bruteforce method, add at the end of a path any possible motif until the path is of the right size.

The only isomorphic paths are obtained by reversing a path. We get a CAT algorithm by discarding non canonical paths.

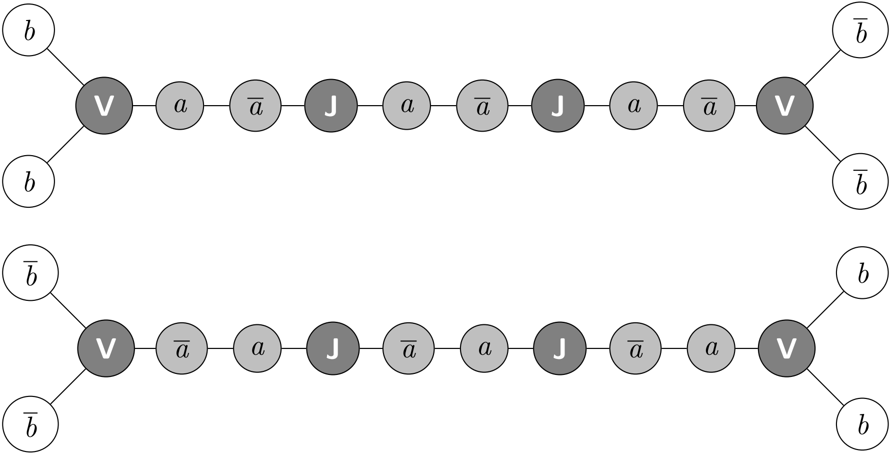
Paths

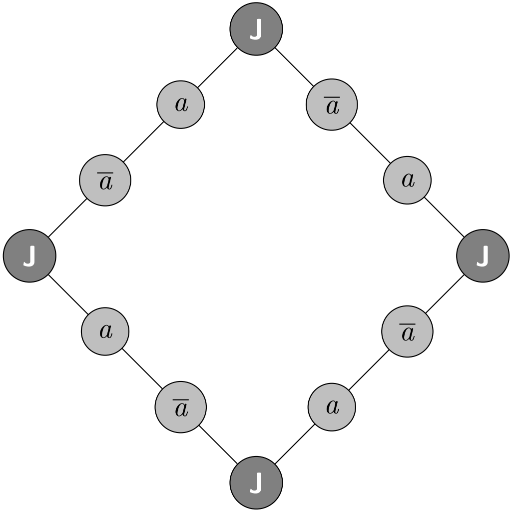
It's a bad idea : not every planar map has an Hamiltonian path. But all planar cubic 3-connected graphs of size less than 38 are Hamiltonian.

It's a good idea: paths are simpler than trees (smaller number).

How we do it: bruteforce method, add at the end of a path any possible motif until the path is of the right size.

The only isomorphic paths are obtained by reversing a path. We get a CAT algorithm by discarding non canonical paths.


b
b
V
a
$\overline{a}$
J
a
$\overline{a}$
J
a
$\overline{a}$
V
$\overline{b}$
$\overline{b}$
$\overline{b}$
$\overline{b}$
V
$\overline{a}$
a
J
$\overline{a}$
a
J
$\overline{a}$
a
V
b
b


J
a
$\overline{a}$
$\overline{a}$
a
J
J
a
$\overline{a}$
$\overline{a}$
a
J

Cycles

It's a good idea : There are even less circuits than paths. The maps will be 2-connected.

It's a bad idea: not every planar map has an Hamiltonian circuit. But all planar cubic 3-connected graphs of size less than 30 have one.

Cycles

It's a good idea : There are even less circuits than paths. The maps will be 2-connected.

It's a bad idea: not every planar map has an Hamiltonian circuit. But all planar cubic 3-connected graphs of size less than 30 have one.

For $\{Y, V_1, V_2\}$ and 8 motifs we have 40112 trees, 9024 paths and less than 2000 cycles.

Cycles

It's a good idea : There are even less circuits than paths. The maps will be 2-connected.

It's a bad idea: not every planar map has an Hamiltonian circuit. But all planar cubic 3-connected graphs of size less than 30 have one.

For $\{Y, V_1, V_2\}$ and 8 motifs we have 40112 trees, 9024 paths and less than 2000 cycles.

Fold and outline

The fold operation on the vertices u and v is adding the edge (u, v) to G. Valid when u and v are:

1. free
2. of complementary colors
3. in the same face of G

A map is foldable when by successive fold operations we can turn it into a saturated map.

Fold and outline

The fold operation on the vertices u and v is adding the edge (u, v) to G. Valid when u and v are:

1. free
2. of complementary colors
3. in the same face of G

A map is foldable when by successive fold operations we can turn it into a saturated map.

The outline of a face is the list in order of traversal of the free vertices. When the backbone is a tree or a path there is a single outline.

Fold and outline

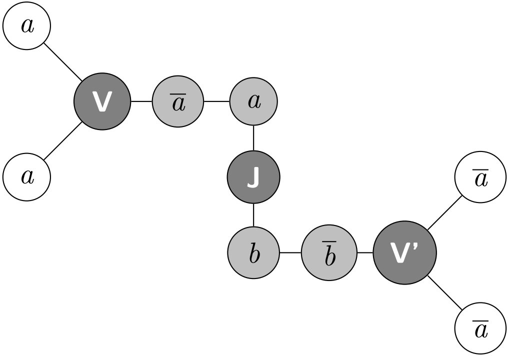
The fold operation on the vertices u and v is adding the edge (u, v) to G. Valid when u and v are:

1. free
2. of complementary colors
3. in the same face of G

A map is foldable when by successive fold operations we can turn it into a saturated map.

The outline of a face is the list in order of traversal of the free vertices. When the backbone is a tree or a path there is a single outline.

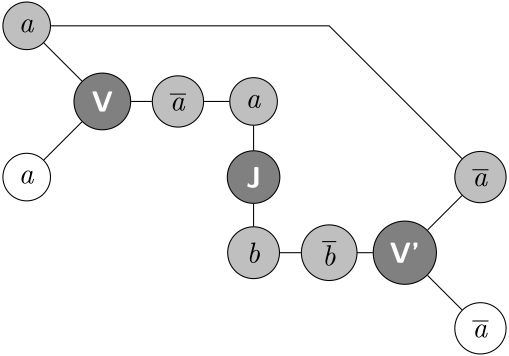
Example



outline $= \{a, \overline{a}, \overline{a}, a\}$

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

Example



$$\text{outline} = \{\overline{a}, a\}$$

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

When is a map foldable?

The outline is a circular sequence of vertices. The fold remove two vertices of compatible colours.

Enough to work with the sequence of colours of the vertices. In the previous example $a\bar{a}\bar{a}a$.

When is a map foldable?

The outline is a circular sequence of vertices. The fold remove two vertices of compatible colours.

Enough to work with the sequence of colours of the vertices. In the previous example $a\bar{a}\bar{a}a$.

Definition

A word is a Dyck word if we can reduce it to the empty word by removing consecutive complementary letters.

When is a map foldable?

The outline is a circular sequence of vertices. The fold remove two vertices of compatible colours.

Enough to work with the sequence of colours of the vertices. In the previous example $a\bar{a}\bar{a}a$.

Definition

A word is a Dyck word if we can reduce it to the empty word by removing consecutive complementary letters.

Lemma

A map is foldable if and only if the associated word is a Dyck word.

When is a map foldable?

The outline is a circular sequence of vertices. The fold remove two vertices of compatible colours.

Enough to work with the sequence of colours of the vertices. In the previous example $a\bar{a}\bar{a}a$.

Definition

A word is a Dyck word if we can reduce it to the empty word by removing consecutive complementary letters.

Lemma

A map is foldable if and only if the associated word is a Dyck word.

This yields a linear time algorithm to test whether a map is foldable.

When is a map foldable?

The outline is a circular sequence of vertices. The fold remove two vertices of compatible colours.

Enough to work with the sequence of colours of the vertices. In the previous example $a\bar{a}\bar{a}a$.

Definition

A word is a Dyck word if we can reduce it to the empty word by removing consecutive complementary letters.

Lemma

A map is foldable if and only if the associated word is a Dyck word.

This yields a linear time algorithm to test whether a map is foldable.

How to avoid non foldable maps?

Definition

A map is almost foldable if for every letter in $a \in \mathcal{A}$, there are as many vertices labeled with a and $\bar{a}$.

Since a foldable backbone is always almost foldabe, we would like to enumerate almost foldable backbones only.

How to avoid non foldable maps?

Definition

A map is almost foldable if for every letter in $a \in \mathcal{A}$, there are as many vertices labeled with a and $\bar{a}$.

Since a foldable backbone is always almost foldabe, we would like to enumerate almost foldable backbones only.

We use a dynamic programming algorithm of complexity $O(n^{k+1})$ where k is the number of letters.

How to avoid non foldable maps?

Definition

A map is almost foldable if for every letter in $a \in \mathcal{A}$, there are as many vertices labeled with a and $\bar{a}$.

Since a foldable backbone is always almost foldabe, we would like to enumerate almost foldable backbones only.

We use a dynamic programming algorithm of complexity $O(n^{k+1})$ where k is the number of letters.

Seems large, but small with regards to the C^n paths.

How to avoid non foldable maps?

Definition

A map is almost foldable if for every letter in $a \in \mathcal{A}$, there are as many vertices labeled with a and $\bar{a}$.

Since a foldable backbone is always almost foldabe, we would like to enumerate almost foldable backbones only.

We use a dynamic programming algorithm of complexity $O(n^{k+1})$ where k is the number of letters.

Seems large, but small with regards to the C^n paths.

$\{I, V1, V2\}$ of size 18:

- $179,896,320$ paths in 78.7s
- $1,277,952$ almost foldable paths in 0.63s.

How to avoid non foldable maps?

Definition

A map is almost foldable if for every letter in $a \in \mathcal{A}$, there are as many vertices labeled with a and $\bar{a}$.

Since a foldable backbone is always almost foldabe, we would like to enumerate almost foldable backbones only.

We use a dynamic programming algorithm of complexity $O(n^{k+1})$ where k is the number of letters.

Seems large, but small with regards to the C^n paths.
$\{I, V1, V2\}$ of size 18:

- $179,896,320$ paths in 78.7s
- $1,277,952$ almost foldable paths in 0.63s.

How to fold a map?

We call result of a sequence of reductions the set of pairs (i, j) such that the sequence has paired i and j.

Problem: given a word, we want to generate the results of all sequences of reductions which yield an empty word.

How to fold a map?

We call result of a sequence of reductions the set of pairs (i, j) such that the sequence has paired i and j.

Problem: given a word, we want to generate the results of all sequences of reductions which yield an empty word.

Another dynamic programming algorithm:

- Build the matrix M such that $M_{i,j}$ is true if and only if the subword $w_i \dots w_j$ is foldable.
- In the enumeration algorithm a partially folded word is a set of subwords.
- At each step reduce the first non folded letter with all possible letters given by M.
- The preprocessing is in $O(n^2)$ and the delay is linear.

How to fold a map?

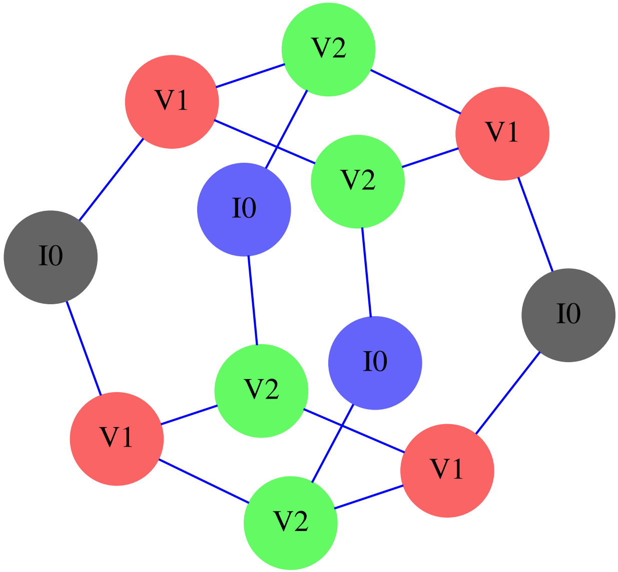
We call result of a sequence of reductions the set of pairs (i, j) such that the sequence has paired i and j.

Problem: given a word, we want to generate the results of all sequences of reductions which yield an empty word.

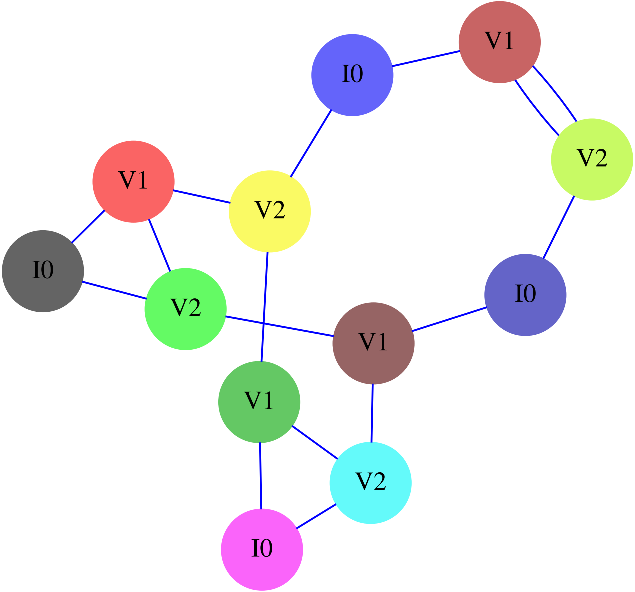
Another dynamic programming algorithm:

- Build the matrix M such that $M_{i,j}$ is true if and only if the subword $w_i \dots w_j$ is foldable.
- In the enumeration algorithm a partially folded word is a set of subwords.
- At each step reduce the first non folded letter with all possible letters given by M.
- The preprocessing is in $O(n^2)$ and the delay is linear.

Results



Results



Dealing with duplicates

The algorithm generates many duplicates.
$\{I, V1, V2\}$ of size 18: $1,658,070$ maps produced but only $23,901$ unique maps

Solution: For each planar map we compute a unique signature. We store the map with its signature in a self-balancing binary search tree.

Dealing with duplicates

The algorithm generates many duplicates.
$\{I, V1, V2\}$ of size 18: $1,658,070$ maps produced but only $23,901$ unique maps

Solution: For each planar map we compute a unique signature. We store the map with its signature in a self-balancing binary search tree.

For each edge, compute the DF traversal of the map, the smallest lexicographically is the signature. The complexity is quadratic.

Dealing with duplicates

The algorithm generates many duplicates.
$\{I, V1, V2\}$ of size 18: $1,658,070$ maps produced but only $23,901$ unique maps

Solution: For each planar map we compute a unique signature. We store the map with its signature in a self-balancing binary search tree.

For each edge, compute the DF traversal of the map, the smallest lexicographically is the signature. The complexity is quadratic.

Open question: more than 50 percent of computation time is spent in signature computation. Practical linear time algorithme ? Average linear time algorithm ?

Dealing with duplicates

The algorithm generates many duplicates.
$\{I, V1, V2\}$ of size 18: $1,658,070$ maps produced but only $23,901$ unique maps

Solution: For each planar map we compute a unique signature. We store the map with its signature in a self-balancing binary search tree.

For each edge, compute the DF traversal of the map, the smallest lexicographically is the signature. The complexity is quadratic.

Open question: more than 50 percent of computation time is spent in signature computation. Practical linear time algorithme ? Average linear time algorithm ?

Indices computations

For each non isomorphic map we must compute indices.

1. Computing all faces and their sizes: $O(n)$
2. The equivalence class of each vertex: $O(n^2)$
   Same idea as the signature : two equivalent vertices yields the same DF traversal.

Indices computations

For each non isomorphic map we must compute indices.

1. Computing all faces and their sizes: $O(n)$
2. The equivalence class of each vertex: $O(n^2)$
   Same idea as the signature : two equivalent vertices yields the same DF traversal.
3. Computing the minimum sparsity of a map. Currently 15 percent of the total time.

Indices computations

For each non isomorphic map we must compute indices.

1. Computing all faces and their sizes: $O(n)$
2. The equivalence class of each vertex: $O(n^2)$
   Same idea as the signature : two equivalent vertices yields the same DF traversal.
3. Computing the minimum sparsity of a map. Currently 15 percent of the total time.

Minimum Sparsity

Problem NP-hard in general and hard to approximate within a constant factor.

We used a Gray Code to generate all partitions: $O(2^n)$. It was already a bit slow for graphs of size 20.

Minimum Sparsity

Problem NP-hard in general and hard to approximate within a constant factor.

We used a Gray Code to generate all partitions: $O(2^n)$. It was already a bit slow for graphs of size 20.

There is a cubic algorithm for planar graphs !

Minimum Sparsity

Problem NP-hard in general and hard to approximate within a constant factor.

We used a Gray Code to generate all partitions: $O(2^n)$. It was already a bit slow for graphs of size 20.

There is a cubic algorithm for planar graphs !

Idea: To a graph G associates its dual graph D_G.

G	D_G
cut	cycles
size of the cut	size of the cycle
size of the partition	weigth of the cycle

We find a cycle of minimal size for each possible weigth.

Minimum Sparsity

Problem NP-hard in general and hard to approximate within a constant factor.

We used a Gray Code to generate all partitions: $O(2^n)$. It was already a bit slow for graphs of size 20.

There is a cubic algorithm for planar graphs !

Idea: To a graph G associates its dual graph D_G.

G	D_G
cut	cycles
size of the cut	size of the cycle
size of the partition	weigth of the cycle

We find a cycle of minimal size for each possible weigth.

Current challenges

Bottlenecks of Kékulé:

1. Generate maps of motifs which are trees (find a CAT algorithm).
2. We cannot use paths or cycles in all cases.

Current challenges

Bottlenecks of Kékulé:

1. Generate maps of motifs which are trees (find a CAT algorithm).
2. We cannot use paths or cycles in all cases.
3. Computing the signature of a map. Cannot be significantly improved.

Current challenges

Bottlenecks of Kékulé:

1. Generate maps of motifs which are trees (find a CAT algorithm).
2. We cannot use paths or cycles in all cases.
3. Computing the signature of a map. Cannot be significantly improved.
4. Combinatorial explosion. For some set of motifs, fast enough but to many generated maps.

Current challenges

Bottlenecks of Kékulé:

1. Generate maps of motifs which are trees (find a CAT algorithm).
2. We cannot use paths or cycles in all cases.
3. Computing the signature of a map. Cannot be significantly improved.
4. Combinatorial explosion. For some set of motifs, fast enough but to many generated maps.

Solutions?

1. Concatenation of base motifs to form a new base.
2. New generation technique through ear decomposition.
3. Generate only graphs satisfying additional constraints on connectivity, face size, sparsity...

Current challenges

Bottlenecks of Kékulé:

1. Generate maps of motifs which are trees (find a CAT algorithm).
2. We cannot use paths or cycles in all cases.
3. Computing the signature of a map. Cannot be significantly improved.
4. Combinatorial explosion. For some set of motifs, fast enough but to many generated maps.

Solutions?

1. Concatenation of base motifs to form a new base.
2. New generation technique through ear decomposition.
3. Generate only graphs satisfying additional constraints on connectivity, face size, sparsity…

Data extraction

A very large sequence of conformations of a molecule.
We need to detect each time a bound appears or disappears in the sequence.

Very easy algorithmically, but very *large dataset*. We need a linear complexity or even less by using the fact that two consecutive images are very similar.

Data extraction

A very large sequence of conformations of a molecule.
We need to detect each time a bound appears or disappears in the sequence.

Very easy algorithmically, but very *large dataset*. We need a linear complexity or even less by using the fact that two consecutive images are very similar.

Identify and classify all possible configurations and transitions between configuration.

Data extraction

A very large sequence of conformations of a molecule.
We need to detect each time a bound appears or disappears in the sequence.

Very easy algorithmically, but very *large dataset*. We need a linear complexity or even less by using the fact that two consecutive images are very similar.

Identify and classify all possible configurations and transitions between configuration.

Same kind of problem: identify an interesting interaction between a given ligand and any protein in the PDB.

Data extraction

A very large sequence of conformations of a molecule.
We need to detect each time a bound appears or disappears in the sequence.

Very easy algorithmically, but very *large dataset*. We need a linear complexity or even less by using the fact that two consecutive images are very similar.

Identify and classify all possible configurations and transitions between configuration.

Same kind of problem: identify an interesting interaction between a given ligand and any protein in the PDB.

Filter, so we do not test every possible protein in the PDB.

Data extraction

A very large sequence of conformations of a molecule.
We need to detect each time a bound appears or disappears in the sequence.

Very easy algorithmically, but very *large dataset*. We need a linear complexity or even less by using the fact that two consecutive images are very similar.

Identify and classify all possible configurations and transitions between configuration.

Same kind of problem: identify an interesting interaction between a given ligand and any protein in the PDB.

Filter, so we do not test every possible protein in the PDB.

Isomorphism

Most of the questions we are asked deal with:

- classifying objects:
- caracterizing object:
- comparing objects:

where object = molecule, cristal, conformation ...

Isomorphism

Most of the questions we are asked deal with:

- classifying objects: isomorphism
- caracterizing object: signature
- comparing objects: approximate isomorphism
  - isomorph up to a constant number of point or by changing a type of node by another
  - are the geometries alike
  - description at different scales

where object = molecule, cristal, conformation ...

Isomorphism

Most of the questions we are asked deal with:

- classifying objects: isomorphism
- caracterizing object: signature
- comparing objects: approximate isomorphism
  - isomorph up to a constant number of point or by changing a type of node by another
  - are the geometries alike
  - description at different scales

where object = molecule, cristal, conformation . . .

Experiment planning

We have:

- a set of products $P_1, \ldots, P_l$
- a set of experiment $E_1, E_2, \ldots, E_k$
- the result of each experiment on each product represented by an integer

Question: Given a product P choose a minimal set of experiments which characterizes P.

Real Question: Devise a strategy (decision tree) such that the average number of experiments to characterize a molecule is minimal.

Experiment planning

We have:

- a set of products $P_1, \ldots, P_l$
- a set of experiment $E_1, E_2, \ldots, E_k$
- the result of each experiment on each product represented by an integer

Question: Given a product P choose a minimal set of experiments which characterizes P.

Real Question: Devise a strategy (decision tree) such that the average number of experiments to characterize a molecule is minimal.

Both problems are NP-hard.

Experiment planning

We have:

- a set of products $P_1, \ldots, P_l$
- a set of experiment $E_1, E_2, \ldots, E_k$
- the result of each experiment on each product represented by an integer

Question: Given a product P choose a minimal set of experiments which characterizes P.

Real Question: Devise a strategy (decision tree) such that the average number of experiments to characterize a molecule is minimal.

Both problems are NP-hard.

Is there a good heuristic ? A solution when the number of experiments is small (FPT algorithm) ?

Experiment planning

We have:

- a set of products $P_1, \ldots, P_l$
- a set of experiment $E_1, E_2, \ldots, E_k$
- the result of each experiment on each product represented by an integer

Question: Given a product P choose a minimal set of experiments which characterizes P.

Real Question: Devise a strategy (decision tree) such that the average number of experiments to characterize a molecule is minimal.

Both problems are NP-hard.
Is there a good heuristic ? A solution when the number of experiments is small (FPT algorithm) ?

Thanks!