Module 4: Listing Chemical Isomers of a Given 2-Lean Cyclic Chemical Graph

`mol-infer/Cyclic`

March 19, 2021

Contents

1 Introduction

This text explains how to use the program for listing chemical isomers of a given 2-lean cyclic chemical graph [1].

The structure of files and folders of this module are as follows

```
./Module_4
├── Manual_Module_4_Cyclic_en.pdf
├── Manual_Module_4_Cyclic_en.tex
├── Manual_Module_4_Cyclic_jp.pdf
├── Manual_Module_4_Cyclic_jp.tex
├── Pseudocode_Graph_Generation.pdf
└── files
    ├── Makefile
    ├── executables
    │   ├── linux
    │   │   ├── generate_isomers
    │   │   └── generate_partition
    │   ├── osx
    │   │   ├── generate_isomers
    │   │   └── generate_partition
    │   └── windows
    │       ├── generate_partition.exe
    │       └── generate_isomers.exe
    ├── instances
    │   ├── sample_1_partition.txt
    │   ├── sample_1.png
    │   └── sample_1.sdf
    ├── main
    │   ├── readme.txt
    │   ├── generate_partition.cpp
    │   ├── output.sdf
    │   └── generate_isomers.cpp
    └── include
        ├── chemical_graph.hpp
        ├── data_structures.hpp
        ├── tools.hpp
        ├── cross_timer.h
        └── fringe_tree.hpp
```

The contents of each of the files is as follows.

- `Manual_Module_4_Cyclic_en.pdf`
  This file.
- `Manual_Module_4_Cyclic_en.tex`
  A LaTeX source file of this manual.
- `Manual_Module_4_Cyclic_jp.pdf`
  The Japanese version of this manual.

- `Manual_Module_4_Cyclic_jp.tex`
  A LaTeX source file of the Japanese version of the manual.
- `Pseudocode_Graph_Generation.pdf`
  A pdf file with Pseudo-codes for the graph search algorithm.
- Folder `files`
  - `Makefile`
    Compilation directives for the C++ programs of this module.
  - Folder `executables`
    A folder that contains compiled executable files for each of the architectures: linux, osx, and windows.
  - Folder `instances`
    A folder containing sample input instances.
    * `sample_1.sdf`
      A cyclic chemical graph with 20 vertices (non-Hydrogen atoms), core size 18 and core height 1.
    * `sample_1_partition.txt`
      A file containing partition information into acyclic subgraphs of the cyclic chemical graph given in `sample_1.sdf`.
    * `sample_2.sdf`
      A cyclic chemical graph with 50 vertices, core size 24 and core height 6.
    * `sample_2_partition.txt`
      A file containing partition information into acyclic subgraphs of the cyclic chemical graph given in `sample_2.sdf`.
    * `sample_3.sdf`
      A cyclic chemical graph with 60 vertices, core size 31 and core height 4.
    * `sample_3_partition.txt`
      A file containing partition information into acyclic subgraphs of the cyclic chemical graph given in `sample_3.sdf`.
    * `sample_4.sdf`
      A cyclic chemical graph with 120 vertices, core size 60 and core height 4.
    * `sample_4_partition.txt`
      A file containing partition information into acyclic subgraphs of the cyclic chemical graph given in `sample_4.sdf`.
  - Folder `main`
    Folder containing source files written in the C++ programming language.
    * `generate_partition.cpp`
      Functions for calculating a partition of a cyclic chemical graph into acyclic subgraphs.
    * `generate_isomers.cpp`
      Implements an algorithm for listing chemical isomers of a 2-lean cyclic chemical graph.

* Folder `include`
  A folder that contains related header files, written in C++.
  - chemical_graph.hpp
    A header file that contains functions for manipulating chemical graphs.
  - cross_timer.h
    A header file that contains functions for measuring execution time.
  - data_structures.hpp
    Data structures implemented for storing chemical graphs.
  - debug.h
    Used for debugging purposes.
  - fringe_tree.hpp
    Header file with functions for enumerating 2-fringe trees [1].
  - tools.hpp
    Various functions used in the implementation.

The remainder of this text is organized as follows. Section 2 explains some of the terminology used throughout the text. Section 3 gives an explanation of the input and the output of the program for calculating a partition of a cyclic chemical graph into acyclic subgraphs, and provides a computational example. Section 4 gives an explanation of the program for generating chemical isomers of a given 2-lean cyclic chemical graph, explaining the input, output, and presents a computational example.

2 Terminology

This section gives an overview of the terminology used in the text.

- Chemical Graph
  A graph-theoretical description of a chemical compound, where the graph's vertices correspond to atoms, and its (multi) edges to chemical bonds. Each vertex is colored with the chemical element of the atom it corresponds to, and edges have multiplicity according to the corresponding bond order. We deal with "hydrogen-suppressed" graphs, where none of the graph's vertices is colored as hydrogen. This can be done without loss of generality, since there is a unique way to saturate a hydrogen-suppressed chemical graph with hydrogen atoms subject to a fixed valence of each chemical element.

- Feature vector
  A numerical vector giving information such as the count of each chemical element in a chemical graph. For a complete information on the descriptors used in feature vectors for this project, please see [1].

- Partition Information
  Information necessary to specify the base vertices and edges, as well as the vertex and edge components of a chemical graph. For more details, please check [1].

3 Program for Calculating a Partition Into Acyclic Subgraphs

3.1 Input and Output

This section explains the input and output information of the program that is used to calculate a partition of a cyclic chemical graph into acyclic subgraphs. We call this program `Partition`. Following, Sec. 3.1.1 explains the input format, and Sec. 3.1.2 gives an explanation the output information of the program.

3.1.1 The Program's Input

The program `Partition` takes two items as its input.

First, is a cyclic chemical graph, given as a "structured data file," in SDF format. This is a standard format for representing chemical graphs. For more details, please check the documentation at

`http://help.accelrysonline.com/ulm/onelab/1.0/content/ulm_pdfs/direct/reference/`
`/ctfileformats2016.pdf`

Next, is a filename of a text file where the information on a partition of the graph given in the SDF into acyclic subgraphs calculated by the program will be saved.

3.1.2 The Program's Output

The output of this program is a partition into acyclic subgraphs of the cyclic chemical graph given in the input. The partition information is stored as a text file with filename as provided in the input parameters.

Output Format Example

```
4
7 # C
0 0 0
15 # C
0 0 0
10 # C
0 0 0
16 # C
0 0 0
5
7 4 3 5 6 9 2 15 # C1C1N1C1C1C1O1C
0 1 0
7 10 # C2C
0 0 1
10 12 14 13 11 7 # C1C2C1C2C1C
0 0 1
```

```
15 16 # C2C
0 0 1
16 18 20 19 17 15 # C1C2C1C2C1C
0 0 1
```

Following, Table 1 gives a row-by-row explanation of the numerical information in the above output file.

Table 1: Structure of an Output File with Partition Information

Value in the file	Explanation
4	Number of base vertices
7 # C 0 0 0 15 # C 0 0 0 10 # C 0 0 0 16 # C 0 0 0	The index of a base vertex in the input SDF and its element Core height lower and upper bound; the vertex component can be created or not (0/1)
5	Number of base edges
7 4 3 5 6 9 2 15 # C1C1N1C1C1C1O1C 0 1 0 7 10 # C2C 0 0 1 10 12 14 13 11 7 # C1C2C1C2C1C 0 0 1 15 16 # C2C 0 0 1 16 18 20 19 17 15 # C1C2C1C2C1C 0 0 1	Indices of vertices in the base edge from the input SDF, their elements and bond multiplicities Core height lower and upper bound, the edge component can be created or not (0/1)

3.2 Program Execution and Computation Example

This section gives an explanation on how to compile and run the program, as well as a concrete computational example of the program's execution.

3.2.1 Compiling and Executing the Program for Generating a Decomposition of a Cyclic Graph into Acyclic Graphs

A compiled executable that has been tested on
- linux

- osx
- windows (cygwin)
is included in the set of files. In addition, we describe how the accompanying source files can be compiled.

- *Computation environment*
  There should not be any problems when using an ISO C++11 compatible compiler.

- *Compiling the program*
  In the terminal, navigate to the `files` subfolder. Then, if the `make` command is available on the system, the program can be simply compiled by typing
  `$ make generate_partition`
  In case the `make` command is not available, then the program can be compiled as
  `$ g++ -o generate_partition ./main/generate_partition.cpp -O3 -std=c++11`

- *Executing the program*
  `$ ./generate_partition instance.sdf instance_partition.txt`
  `instance.sdf` is an input SDF, and a partition information as the output of the program is stored in the file `instance_partition.txt` .

3.2.2 Computational Example

This section illustrates a concrete computational example of running the `Partition` program. We assume the following:

- Input file: `sample_1.sdf` from the folder `instances` (see Sec. 1)
- Output file to store the partition information: `partition.txt`

Run the following command in the terminal to execute the program.

```
./generate_partition ./instances/sample_1.sdf partition.txt
```

After successfully executing the program, the contents of the file `partition.txt` should be as follows.

Contents of the file `partition.txt`

```
4
7 # C
0 0 0
15 # C
0 0 0
10 # C
0 0 0
```

```
16 # C
0 0 0
5
7 4 3 5 6 9 2 15 # C1C1N1C1C1C1O1C
0 1 0
7 10 # C2C
0 0 0
10 12 14 13 11 7 # C1C2C1C2C1C
0 0 0
15 16 # C2C
0 0 0
16 18 20 19 17 15 # C1C2C1C2C1C
0 0 0
```

4 Program for Generating Chemical Isomers

4.1 Input and Output of the Program

This section gives an explanation of the input and output of the program that generates chemical isomers of a given 2-lean chemical graph. We call the program `generate_isomers`. Section 4.1.1 gives an explanation of the program's input, and Sec. 4.1.2 of the program's output.

4.1.1 The Program's Input

The input to the `generate_isomers` program consists of six necessary and one optional item.
First, comes information of a 2-lean chemical graph (in SDF format).
Second, comes a time limit in seconds on each of the stages of the program (for details, check the accompanying file with pseudo-codes of the algorithm).
Third is an upper bound on the number of *partial* feature vectors that the program stores during its computation.
Fourth is the number of "sample graphs" that are stored per one feature vector.
Fifth is an upper limit on the number of generated output chemical graphs.
Sixth is a filename (SDF) where the output graphs will be written.
An optional parameter is a filename with a partition information of the chemical graph given as the first parameter, and if given, it must be last in the list of parameters.

4.1.2 The Program's Output

After executing the `generate_isomers` program, the chemical isomers of the input graph will be written in the specified SDF, and some information on the execution will be output on the terminal. The information printed on the terminal includes:
- a lower bound on the number of chemical isomers of the given input chemical graph,

- the number of graphs that the program generated under the given parameters, and
- the program's execution time.

4.2 Executing the Program and a Computational Example

This section gives a concrete computational example of the `generate_isomers` program.

4.2.1 Compiling and Executing the `generate_isomers` Program

A compiled executable that has been tested on
- linux
- osx
- windows (cygwin)

is included in the set of files. In addition, we describe how the accompanying source files can be compiled.

- *Computation environment*
  There should not be any problems when using a ISO C++ compatible compiler.

- *Compiling the program*
  If the `make` command is available on the system, then the program can be compiled in the `files` folder by typing
  `$ make generate_isomers`
  in the terminal. If the `make` command is not available, then please run the following command in the terminal.
  `$ g++ -o generate_isomers ./main/generate_isomers.cpp -O3 -std=c++11`

- *Executing the program*
  The program can be executed by running the following command in the terminal.
  `$ ./generate_isomers instance.txt a b c d output.sdf instance_partition.txt`
  Above, `generate_isomers` is the name of the program's executable file, and the remaining command-line parameters are as follows:
  `instance.txt` a text file containing a chemical specification
  `a` upper bound (in seconds) on the computation time,
  `b` upper bound on the number of stored partial feature vectors,
  `c` upper bound on the number of sample graphs stored per feature vector,
  `d` upper bound on the number of output graphs,
  `output.sdf` filename to store the output chemical graphs (SDF format),
  `instance_partition.txt` partition information of the input chemical graph.

4.2.2 Computational Example

We execute the `generate_isomers` program with the following parameters.

- Input graph: File `sample_1.sdf` from the folder `instances` (see Sec. 1)

- Time limit: 10 seconds
- Upper limit on the number of partial feature vectors: 10000000
- Number of sample graphs per feature vector: 5
- Upper limit on the number of output graphs: 2
- Filename to store the output graphs: `output.sdf`
- Partition information of the input graph: File `sample_1_partition.txt` from the folder `instances`.

Execute the program by typing the following command into the terminal (without a line break).

```
./generate_isomers ./instances/sample_1.sdf 10 10000000 5 2
                   output.sdf ./instances/sample_1_partition.txt
```

Upon successful execution of the program, the following text should appear on the terminal.

Output Written on the Terminal

```
A lower bound on the number of graphs = 72
Number of generated graphs = 2
Total time :  0.00649s.
```

Contents of the file output.sdf

```
1
BH-cyclic
BH-cyclic
20 21 0 0 0 0 0 0 0 0999 V2000
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 N 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
```

0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
1 3 2 0 0 0 0
1 5 1 0 0 0 0
1 16 1 0 0 0 0
2 4 2 0 0 0 0
2 11 1 0 0 0 0
2 20 1 0 0 0 0
3 13 1 0 0 0 0
4 17 1 0 0 0 0
5 6 1 0 0 0 0
6 7 1 0 0 0 0
7 8 1 0 0 0 0
8 9 1 0 0 0 0
8 10 1 0 0 0 0
10 11 1 0 0 0 0
11 12 1 0 0 0 0
13 14 2 0 0 0 0
14 15 1 0 0 0 0
15 16 2 0 0 0 0
17 18 2 0 0 0 0
18 19 1 0 0 0 0
19 20 2 0 0 0 0
M END
$$$$
2
BH-cyclic
BH-cyclic
20 21 0 0 0 0 0 0 0 0999 V2000
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 N 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0

```
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0 0 0 0 0 0 0 0
1 3 2 0 0 0 0
1 5 1 0 0 0 0
1 16 1 0 0 0 0
2 4 2 0 0 0 0
2 11 1 0 0 0 0
2 20 1 0 0 0 0
3 13 1 0 0 0 0
4 17 1 0 0 0 0
5 6 1 0 0 0 0
6 7 1 0 0 0 0
7 8 1 0 0 0 0
8 9 1 0 0 0 0
8 10 1 0 0 0 0
10 11 1 0 0 0 0
11 12 1 0 0 0 0
13 14 2 0 0 0 0
14 15 1 0 0 0 0
15 16 2 0 0 0 0
17 18 2 0 0 0 0
18 19 1 0 0 0 0
19 20 2 0 0 0 0
M END
$$$$
```

References

[1] T. Akutsu and H. Nagamochi. A Novel Method for Inference of Chemical Compounds with Prescribed Topological Substructures Based on Integer Programming. Arxiv preprint, arXiv:2010.09203