Program Manual: Listing Chemical Isomers of a Given Chemical Graph

March 3, 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].

- Folder`isomers`
  Containing source code files of the program for generating chemical isomers of a cyclic chemical graph. The program is written in the C++ programming language.
  - Folder`include`
    A folder that contains related header files.
    * 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.
    * fringe_tree.hpp
      A header file with functions for enumerating 2-fringe trees with fictitious degree [1].
    * tools.hpp
      Various functions used in the implementation.
    * tree_signature
      A header file with functions to compute and read canonical representation of fringe trees.
  - Folder`instance`
    A folder containing sample input instance.
    * `sample.sdf`
      A cyclic chemical graph with 45 vertices (non-Hydrogen atoms).
    * `sample_partition.txt`
      A file containing partition information into acyclic subgraphs of the cyclic chemical graph given in `sample.sdf`.
    * `sample_fringe_trees.sdf`
      A family of 2-fringe trees attached with the cyclic chemical graph given in `sample.sdf`.
  - Folder`main`
    Folder containing source files.
    * generate_isomers.cpp
      Implements an algorithm for listing chemical isomers of a 2-lean cyclic chemical graph.
    * Pseudocode_Graph_Generation.pdf
      A pdf files showing Pseudo-codes for Graph Search Algorithms.

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 program for generating chemical

isomers of a given 2-lean cyclic chemical graph, explaining the input, output, and presenting 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 and fringe trees for each component. For more details, please check [1].

- Fringe Tree Information
  A list of 2-fringe trees to be used to construct isomer. For more details, please check [1].

3 Program for Generating Chemical Isomers

3.1 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 3.1.1 gives an explanation of the program's input, and Sec. 3.1.2 of the program's output.

3.1.1 The Program's Input

The input to the `Generate Isomers` program consists of ten necessary items.
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 program).
Third is an upper bound on the number of *partial* feature vectors, that the program stores during its computation.
Fourth is the number of graphs that are stored per one base vertex or edge.

Fifth is a global time limit for enumeration of paths.
Sixth is a global upper bound on the number of paths enumerated from the DAGs.
Seventh is an upper limit on the number of generated output chemical graphs.
Eighth is a filename (SDF) where the output graphs will be written.
Ninth is a file that contains partition information of the chemical graph given as the first parameter.
Tenth is a file that contains list of fringe trees.

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`

The partition information is stored as a text file and an output of Stage 4.
The `sample_partition.txt` looks like:

Partition information file

```
2
14
0 45
1 2 3 4 5 6 7 8
15
0 45
1 2 3 4 5 6 7 8
2
14 1 2 3 5 6 15
0 45
1 2 3 4 5 6 7 8
14 10 15
0 45
1 2 3 4 5 6 7 8
```

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

Table 1: Structure of a Partition Information

Value in the file	Explanation
2	Number of base vertices
14	The index of a base vertex in the input SDF
0 45	Core height lower and upper bound
1 2 3 4 5 6 7 8	Indices of fringe trees in the fringe tree file
15	
0 45	
1 2 3 4 5 6 7 8	
2	Number of base edges
14 1 2 3 5 6 15	Indices of vertices in the base edge from the input SDF
0 45	Core height lower and upper bound
1 2 3 4 5 6 7 8	Indices of fringe trees in the fringe tree file
14 10 15	
0 45	
1 2 3 4 5 6 7 8	

The list of fringe trees are stored in a text file. The `sample_fringe_trees.sdf` is given below:

Fringe tree file

```
1, C 0 C 1, 1
2, C 0 O 1, 2
3, N 0 C 1, 1
4, C 0 C 1 C 2, 1 1
5, C 0 C 1 C 2 C 2, 1 1 1
6, C 0 C 1 O 2 O 2, 1 1 2
7, N 0 C 1 N 2 N 2, 2 1 1
8, C 0 C 1 C 2 C 1, 1 1 1
```

Following, Table 2 gives a row-by-row explanation of the numerical information in the above fringe tree file.

Table 2: Structure of a Fringe Tree File

Value in the file	Explanation
`1, C 0 C 1, 1`	Index of the fringe tree, (color, depth)-sequence, weight sequence For details please check Pseudocode_Graph_Generation.pdf

`2, C 0 O 1, 2`
`3, N 0 C 1, 1`
`4, C 0 C 1 C 2, 1 1`
`5, C 0 C 1 C 2 C 2, 1 1 1`
`6, C 0 C 1 O 2 O 2, 1 1 2`
`7, N 0 C 1 N 2 N 2, 2 1 1`
`8, C 0 C 1 C 2 C 1, 1 1 1`	

3.1.2 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.

3.2 Executing the Program and a Computational Example

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

3.2.1 Compiling and Executing the `Generate` Program

- *Computation environment*
  There should not be any problems when using a ISO C++ compatible compiler. There program has been tested on Ubuntu Linux 20.04, with the g++ compiler ver 9.3. If the compiler is not installed on the system, it can be installed with the following command.
  `$ sudo apt install g++`

- *Compiling the program*
  Please run the following command in the terminal.
  `$ g++ -o generate_isomers 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 e f 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 on each stages of the program,
`b` upper bound on the number of stored partial feature vectors,
`c` upper bound on the number of graphs stored per base vertex or edge,
`d` upper bound (in seconds) on time for enumeration of paths,
`e` upper bound on the number of total paths stored during the computation,
`f` 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.
`instance_fringe_trees.txt` list of fringe trees.

3.2.2 Computational Example

We execute the `Generate Isomers` program with the following parameters.

- Input graph: File `sample.sdf` from the folder `instance`
- Time limit: 2 seconds
- Upper limit on the number of partial feature vectors: 10000
- Number of graphs per base vertex or edge: 5
- Global time limit for enumeration of paths: 10 seconds
- Global upper bound on number of paths: 10000
- 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_partition.txt` from the folder `instance`.
- Fringe tree information of the input and output graphs: File `sample_fringe_tree.txt` from the folder `instance`.

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

```
./generate_isomers ../instance/sample.sdf 2 10000 5 10 10000 2
                   output.sdf ../instance/sample_partition.txt
                   ../instance/sample_fringe_tree.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 = 115338281250
Number of generated graphs = 2
Total time :  9.02s.
```

Contents of the file output.sdf

```
1
BH-2L
BH-2L
42 42 0 0 0 0 0 0 0 0999 V2000
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 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 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 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 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 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 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 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 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 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
1 8 1 0 0 0 0
1 36 1 0 0 0 0
2 3 1 0 0 0 0
2 8 1 0 0 0 0
2 10 1 0 0 0 0
3 4 1 0 0 0 0
4 5 1 0 0 0 0
5 6 1 0 0 0 0
6 7 1 0 0 0 0
8 9 1 0 0 0 0
10 11 1 0 0 0 0
10 28 1 0 0 0 0
11 12 1 0 0 0 0
11 13 1 0 0 0 0
13 14 1 0 0 0 0
14 15 1 0 0 0 0
14 16 1 0 0 0 0
16 17 1 0 0 0 0
16 18 1 0 0 0 0
18 19 1 0 0 0 0
19 20 1 0 0 0 0
20 21 1 0 0 0 0
20 22 1 0 0 0 0
22 23 1 0 0 0 0
22 24 1 0 0 0 0
24 25 1 0 0 0 0
24 27 1 0 0 0 0
25 26 1 0 0 0 0
28 29 2 0 0 0 0
28 30 1 0 0 0 0
30 31 1 0 0 0 0
30 33 1 0 0 0 0
31 32 1 0 0 0 0
33 34 1 0 0 0 0
33 36 1 0 0 0 0
34 35 1 0 0 0 0

```
36 37 1 0 0 0 0
37 38 1 0 0 0 0
38 39 1 0 0 0 0
39 40 1 0 0 0 0
39 42 1 0 0 0 0
40 41 1 0 0 0 0
M END
$$$$
2
BH-2L
BH-2L
42 42 0 0 0 0 0 0 0 0999 V2000
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 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 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 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 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 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 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 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 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 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
1 8 1 0 0 0 0
1 36 1 0 0 0 0
2 3 1 0 0 0 0
2 8 1 0 0 0 0
2 10 1 0 0 0 0
3 4 1 0 0 0 0
4 5 1 0 0 0 0
5 6 1 0 0 0 0
6 7 1 0 0 0 0
8 9 1 0 0 0 0
10 11 1 0 0 0 0
10 28 1 0 0 0 0
11 12 1 0 0 0 0
11 13 1 0 0 0 0
13 14 1 0 0 0 0
14 15 1 0 0 0 0
14 16 1 0 0 0 0
16 17 1 0 0 0 0
16 18 1 0 0 0 0
18 19 1 0 0 0 0
19 20 1 0 0 0 0
20 21 1 0 0 0 0
20 22 1 0 0 0 0
22 23 1 0 0 0 0
22 24 1 0 0 0 0
24 25 1 0 0 0 0
24 27 1 0 0 0 0
25 26 1 0 0 0 0
28 29 2 0 0 0 0
28 30 1 0 0 0 0

```
30 31 1 0 0 0 0
30 33 1 0 0 0 0
31 32 1 0 0 0 0
33 34 1 0 0 0 0
33 36 1 0 0 0 0
34 35 1 0 0 0 0
36 37 1 0 0 0 0
37 38 1 0 0 0 0
38 39 1 0 0 0 0
39 40 1 0 0 0 0
39 42 1 0 0 0 0
40 41 1 0 0 0 0
M END
$$$$
```

References

[1] Y. Shi, J. Zhu, N. A. Azam, K. Haraguchi, L. Zhao, H. Nagamochi, and T. Akutsu. A two-layered model for inferring chemical compounds with integer programming, J. Mol. Sci. [submitted].