Chemical Graph Project: Inferring Chemical Graphs

`mol-infer/2LMM-LLR-monomer`

October 5, 2021

Introduction

Abstract

Our research goal is to develop a system that generates chemical graphs such that the corresponding compounds must attain a desired value for a prescribed chemical property. This system consists of four modules as follows.

Module 1: Converting a given data on chemical graphs (represented by an SDF file) into a set of feature vectors.

Module 2: Constructing a prediction function by using Lasso Linear Regression (LLR).

Module 3: Inferring a chemical graph that must attain a desired property value by using mixed integer linear programming.

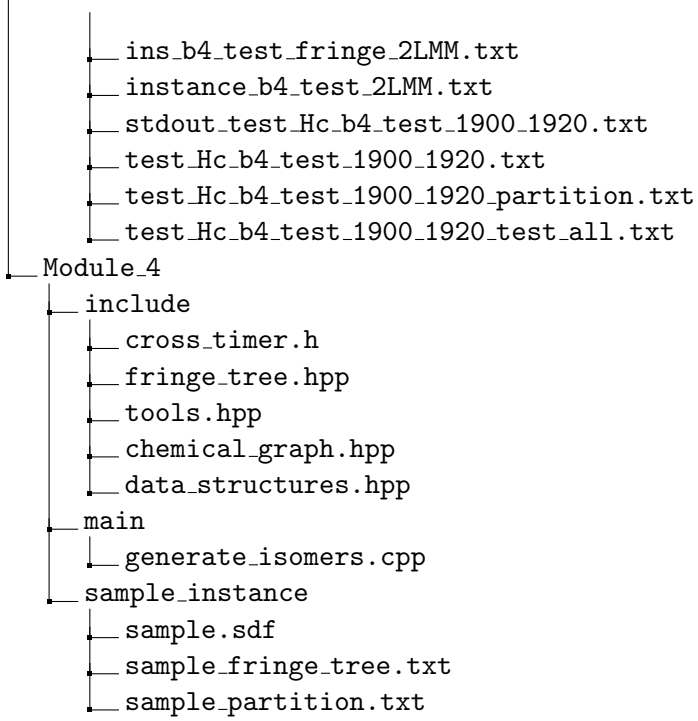
Module 4: Listing chemical isomers of the chemical graph obtained in Module 3.

This booklet explains how to obtain inferred chemical graphs by using the programs of the modules.

Structure of Files and Folders

```
├── LICENSE
├── README.md
└── bin
    ├── eliminate.py
    ├── infer_2LMM_LLR.py
    ├── read_instance_2layer_2LMM_L.py
    ├── twolayered_MILP_2LMM_L.py
    ├── lasso_eval_linreg.py
    ├── limit_atoms.py
    ├── lr_inverter.py
    ├── osx
    │   ├── generate_isomers
    │   └── FV_2LMM_V018
    └── linux
        ├── generate_isomers
        └── FV_2LMM_V018
```

```
├── cygwin
│   ├── generate_isomers
│   └── FV_2LMM_V018
├── doc
│   ├── 2LMM-LLR_flow.pdf
│   └── Manual_2LMM-LLR-monomer_en.pdf
├── instances_used_in_exp
│   ├── instances_for_FV_and_Learning.zip
│   ├── instances_for_MILP.zip
│   └── instances_for_DP.zip
└── src
    ├── Module_1
    │   ├── compute_fc.cpp
    │   ├── fv_2LMM.cpp
    │   ├── fv_common.h
    │   ├── fv_common.hpp
    │   ├── fv_def.h
    │   ├── limit_atoms.py
    │   ├── eliminate.py
    │   ├── fringe
    │   │   ├── ChemicalGraph.hpp
    │   │   ├── commonused.hpp
    │   │   ├── cross_timer.hpp
    │   │   ├── RootedGraph.hpp
    │   │   └── TopologyGraph.hpp
    │   └── sample_instance
    │       ├── sample1.txt
    │       ├── output_fringe.txt
    │       ├── output_desc.csv
    │       └── output_desc_norm.csv
    ├── Module_2
    │   ├── lasso_eval_linreg.py
    │   └── sample_instance
    │       ├── FV_Alpha_desc_norm.csv
    │       ├── Alpha_norm_values.txt
    │       └── output_reg.txt
    └── Module_3
        ├── infer_2LMM_LLR.py
        ├── read_instance_2layer_2LMM_L.py
        ├── twolayered_MILP_2LMM_L.py
        ├── lasso_eval_linreg.py
        ├── lr_inverter.py .4 README.txt
        └── sample_instance
            ├── Hc_desc.csv
            ├── Hc_desc_norm.csv
            ├── Hc_fringe.txt
            ├── Hc_linreg.txt
            └── Hc_values.txt
```


ins_b4_test_fringe_2LMM.txt
instance_b4_test_2LMM.txt
stdout_test_Hc_b4_test_1900_1920.txt
test_Hc_b4_test_1900_1920.txt
test_Hc_b4_test_1900_1920_partition.txt
test_Hc_b4_test_1900_1920_test_all.txt
Module_4
include
cross_timer.h
fringe_tree.hpp
tools.hpp
chemical_graph.hpp
data_structures.hpp
main
generate_isomers.cpp
sample_instance
sample.sdf
sample_fringe_tree.txt
sample_partition.txt

Contents

Chapter 1

Module 1: Calculating a Feature Vector from an SDF File

1.1 Introduction

This chapter serves as a manual and explains the procedures to run Module 1 of the `mol-infer/2LMM-LLR-monomer` project. The input and output of Module 1 are as follows.

Input: A set $D = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_p\}$ of chemical graphs.

Output: A set $\mathcal{F}(D) \triangleq \{f(\mathbb{C}_1), f(\mathbb{C}_2), \ldots, f(\mathbb{C}_p)\}$ of feature vectors, such that $f(\cdot)$ is a feature vector of chemical graphs as details in the accompanying article [1].

The output is written to a csv (comma-separated value) file. This csv file is used in Module 2 of the project.

The remainder of this chapter is organized as follows.

- Chapter 1.2: Summary of essential terminology.
- Chapter 1.3: A short computational example.
- Chapter 1.4: Detailed explanations of the program's input and output.

1.2 Preliminaries

Chemical Graph. A **graph** is an abstract combinatorial construction comprising a set of **nodes** and a set of **edges**, where an edge is an unordered pair of nodes. A **cycle** in a graph is a sequence of nodes such that except for the first and the last node, each node is unique, and there is an edge in the graph for each pair of consecutive nodes in the sequence.

A graph where each node is assigned a chemical element (such as carbon, nitrogen, oxygen, etc.) and each edge is assigned a multiplicity between 1 and 4, is called a **chemical graph**.

Descriptor. A **descriptor** is a numerical value that indicates a certain characteristic of a chemical graph. In this project, among others, descriptors include the number of non-hydrogen atoms, average mass over all atoms, etc. For a complete list of descriptors, please refer to the accompanying article [1].

Feature vector. A vector that comprises the numerical values for the descriptors of a chemical graph.

1.3 Execution Example

1.3.1 Validation of the Data

Data on chemical compounds (equivalently, chemical graphs) is stored in a standard SDF file(more information on the structure of SDF files is given in Chapter 1.4). Each chemical graph $\mathbb{C}$ must satisfy the following conditions

(i) $\mathbb{C}$ must contain at least four carbon atoms, and each atom must have atomic mass as listed in Chapter 1.4.3; and

(ii) Must not include an aromatic edge.

The Python script `eliminate.py` can be used to remove the graphs that do not satisfy condition (i). For condition (ii), the user must confirm whether it is satisfied or not on his/her own.

Elimination of chemical graphs that are out-of-scope. To check whether each chemical graph in a given SDF file satisfies condition (i) or not, please use the Python script named `eliminate.py`. The script generates a new SDF file that consists of all chemical graphs in the input SDF file that satisfy (i).

To use `eliminate.py`, execute the following command.

```
$ python eliminate.py input.sdf
```

If the `input.sdf` contains a chemical graph that does not satisfy (i), the CID is printed.

After the execution of `eliminate.py`, a new SDF file `input_eli.sdf` is created. The file consists of all chemical graphs in `input.sdf` that satisfy condition (i). This means that, if all chemical graphs in `input.sdf` satisfy (i), then `input.sdf` and `input_eli.sdf` are equivalent.

1.3.2 Restricting Atom Set (Optional)

To extract chemical graphs from an SDF file with a given atom set Λ, please use the Python script named `limit_atoms.py`. The script generates a new SDF file that consists of all chemical graphs in the input SDF file that have atom set to be the subset of Λ.

To use `limit_atoms.py`, for the input SDF file `input.sdf` and the atom set $\Lambda = \{\mathtt{C}, \mathtt{O}, \mathtt{N}\}$, execute the following command.

```
$ python limit_atoms.py input.sdf C O N
```

After the execution of `limit_atoms.py`, a new SDF file `input_C_O_N.sdf` is created. The file consists of all chemical graphs in `input.sdf` that whose atom set is a subset of $\Lambda = \{\mathtt{C}, \mathtt{O}, \mathtt{N}\}$.

1.3.3 Calculating a Feature Vector

Please use the program compiled from the source file `fv_2LMM.cpp` to calculate feature vectors for an SDF file such that every chemical satisfies conditions (i), and (ii).

A compiled executable is included in the folder `bin`, for each of the following operating systems:

- linux
- osx
- windows (cygwin).

To compile the program, the included `Makefile` can be used by issuing the following command in the command prompt.

```
$ make FV_2LMM_V018
```

In case the `make` command is not available on the system, then the program can be compiled in the following way.

```
$ g++ -o FV_2LMM_V018 fv_2LMM.cpp -O3 -std=c++11
```

In order to calculate feature vectors from `sample1.sdf` and to output the result in `output.csv`, issue the following command on the terminal.

```
$ ./FV_2LMM_V018 ../src/Module1/sample_instance/sample1.sdf output.csv
```

The program `FV_2LMM_V018` prints on the terminal instructions on how to provide the arguments and halts if the arguments are not provided appropriately.

1.4 Details in the Input and Output of the Program

1.4.1 Input

The programs in Module 1 use SDF (Structure Data File), a standard format, for input. For the detail of SDF, splease check the following reference:

- `http://help.accelrysonline.com/ulm/onelab/1.0/content/ulm_pdfs/direct/reference/ctfileformats2016.pdf` (accessible on Feb 1, 2021)

1.4.2 Output

The program outputs three files for an input SDF:

- `output_desc.csv`: unnormalized feature vectors;
- `output_desc_norm.csv`: normalized feature vectors; and
- `output_fringe.txt`: text file that contains fringe trees.

The only difference between the files `output_desc_norm.csv` and `output_desc.csv` is that the former has normalized values. Therefore we explain the structure of the files `output_desc.csv` and `output_fringe.txt`.

The fringe tree file `output_fringe.txt` contains all the non-isomorphic fringe trees that appear in the chemical graphs of the input file.

We consider a depth-first-search (DFS) ordering on the vertices of a fringe tree and represent it by the following three sequences:

- `(atom depth)-seq`: An alternating sequence which contains the information of atom and the depth of each vertex,
- `Multiplicity seq`: A sequence which contains the information of multiplicity of an edge between two adjacent vertices following DFS ordering,
- `Charge seq`: A sequence which contains the information of charge of atom of each vertex following DFS ordering.

These three sequences are separated by a comma and the entries of each of these sequences are separated by a blank space in the file `output_fringe.txt`. The first line of `output_fringe.txt` shows the header of each column and the following lines show the values of those headers. The following frame shows an example of `output_fringe.txt` file for the input SDF `sample1.sdf`.

```
FID,(atom depth)-seq, Multiplicity seq, Charge seq
1, C4 0 H1 1 H1 1, 1 1, 0 0 0
2, C4 0 H1 1 H1 1 C4 1 O2 2 O2 2 H1 3, 1 1 1 2 1 1, 0 0 0 0 0 0 0
```

The first line of `output_desc.csv` shows the components of FV (Feature Vector) and the following lines show the values for those components of FV. The following frame shows an example of `output_desc.csv` file for the input SDF `sample1.sdf`.

```
CID n rank n_in ms dg_1 dg_2 dg_3 dg_4 dg_in_1 dg_in_2 dg_in_3 dg_in_4 \
bd_in_2 bd_in_3 na_in_C4 na_ex_C4 na_ex_O2 C4_2_C4_2_1 \
FC_1_C4_0_H1_1_H1_1_/_1_1_/_0_0_0 \
FC_2_C4_0_H1_1_H1_1_C4_1_O2_2_O2_2_H1_3_/_1_1_1_2_1_1_/_0_0_0_0_0_0_0 \
LeafAC_O2_C4_1 LeafAC_O2_C4_2
196 10 0 4 73 0 4 0 0 2 2 0 0 0 0 4 2 4 3 2 2 2 2
```

The symbol \ at the end of a line indicates that there is no line break between the two lines. Here is the overview of descriptors. See [1] for details.

- **CID:** Compound ID.
- **n:** Number of atoms except for the hydrogen.
- **rank:** Rank of the chemical graph.
- **n_in :** The number of interior-vertices.
- **ms:** Average molecular mass.
- **dg_i:** For each **i** $\in [1, 4]$, the number of non-hydrogen vertices with degree **i** in the hydrogen-suppressed chemical graph.
- **dg_in_i:** For each **i** $\in [1, 4]$, the number of interior-vertices with degree **i**.
- **bd_in_i:** For each **i** $\in [2, 3]$, the number of interior-edges with bond multiplicity **i**.
- **na_in_Xx:** The frequency of chemical element **Xx** with valence **x** in the interior-vertices.
- **na_ex_Xx:** The frequency of chemical element **Xx** with valence **x** in the exterior-vertices.
- **Xx_i_Yy_j_z:** The frequency of edge configuration $(\mathtt{a}d, \mathtt{b}d', m)$ with $\mathtt{a} = \mathbf{Xx}$, $d = \mathbf{i}$, $\mathtt{b} = \mathbf{Yy}$, $d' = \mathbf{j}$ and $m = \mathbf{z}$.
- **FC_i_Xx_a_Yy_b_..._/_m_n_..._/_r_s_... :** The frequency of the **i**-th fringe tree ψ with atom sequence (**Xx, Yy, ...**), depth sequence (**a, b, ...**), multiplicity sequence (**m, n, ...**), and charge sequence (**r, s, ...**) in depth first search order.
- **LeafAC_Xx_Yy_z:** The frequency of the adjacency-configuration $(\mathtt{a}, \mathtt{b}, m)$ with $\mathtt{a} = \mathbf{Xx}$, $\mathtt{b} = \mathbf{Yy}$ and $m = \mathbf{z}$.

1.4.3 Attention

The mass of each atom is hard-coded in the program. They are written in the function `init_MassMap()` in `fv_common.cpp` as follows. If one needs to change values or to add another atoms, edit the source code directly and compile again.

```
Mass["e*"] = 0;
Mass["H"] =  10;
Mass["B"]  = 108;
Mass["C"]  = 120;
Mass["O"]  = 160;
Mass["N"]  = 140;
Mass["F"]  = 190;
Mass["Si"] = 280;
Mass["P"]  = 309;
Mass["S"]  = 320;
Mass["Cl"] = 354;
Mass["V"]  = 509;
Mass["Br"] = 799;
Mass["Cd"] = 1124;
Mass["I"]  = 1269;
Mass["Hg"] = 2006;
Mass["Pb"] = 2072;
Mass["Al"] = 269;
```

Chapter 2

Module 2: Constructing Prediction Function Using Lasso Linear Regression

2.1 Introduction

In this article, we explain how to proceed to Module 2, i.e., to construct a prediction function for a given dataset of chemical property by using lasso linear regression (LLR) in our project (`mol-infer/2LMM-LLR-monomer`).

We denote by $D_\pi = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_p\}$ a given set of chemical graphs, and by f a function that maps a chemical graph to a feature vector. We define $\mathcal{F}(D_\pi) \triangleq \{f(\mathbb{C}_1), f(\mathbb{C}_2), \ldots, f(\mathbb{C}_p)\}$. Let us denote by π a chemical property that is considered. For example, π can be boiling point, heat of combustion, Kow, and so on. The input and output of the module are summarized as follows.

Input: A set $\mathcal{F}(D_\pi) = \{x_1, x_2, \ldots, x_p\}$ of molecule feature vectors, a set $\{a(x_1), a(x_2), \ldots, a(x_p)\}$ of observed values $a(x_i)$ for each $\mathbb{C}_i \in D_\pi$ (and $x_i = f(\mathbb{C}_i)$) for the property π, and a value λ that is used in the objective function of Lasso Regression.

Output: A prediction function, that estimates $a(x)$ "well" for many feature vectors $x \in D_\pi$.

Concretely, the output consists of scalars (weights) and a constant (bias) for a hyperplane. The article is organized as follows.

- Chapter 2.2: Some basic terminology.
- Chapter 2.3: A brief example of executing the programs.
- Chapter 2.4: Details about the input and the output of the programs.

2.2 Preparation

Feature vector. A numerical vector that represents features of a molecule. Each dimension corresponds to a *descriptor* that describes such features as the number of atoms of different chemical elements (e.g., carbon, nitrogen, oxygen, etc), the molecular mass averaged over the number of atoms, etc.

Lasso Linear Regression (LLR). A well-known model in machine learning. In this project, we use LLR to solve the *regression* problem. In this problem, given a data set that consists of numerical vectors and the "correct" values assigned to each of the vectors, we are asked to predict values for vectors not in the data set as accurately as possible. In our project, the numerical vectors are feature vectors that are extracted from molecules and the values assigned to the vectors are property values of a prescribed chemical property. The number of dimensions is assumed to be equal over all the feature vectors.

2.3 A Quick Start

Constructing a Prediction Function. The following command constructs a prediction function using LLR from the dataset such that

- the set of feature vectors is written in `src/Module_2/FV_Alpha_desc_norm.csv`; and
- the set of property values (Alpha in this case) is written in `src/Module_2/Alpha_norm_values.txt`

```
$ python lasso_eval_linreg.py ../src/Module_2/sample_instance/FV_Alpha_desc_norm.csv\
 ../src/Module_2/sample_instance/Alpha_norm_values.txt  output_reg.txt 1e-05
```

The symbol \ at the end of the first line indicates that there is no line break between the two lines, that is, they should be input without pressing the "enter" key.

The scalars (weights) and a constant (bias) of the constructed hyperplane are output to the file `output_reg.txt`. **The file containing weights and bias is used in Module 3**.

2.4 Details in the Input and Output of the Program

2.4.1 Input

Feature vectors

The feature vectors should be written in a so-called **FV format** to a csv (comma-separated value) file. The feature vector generator in Module 1 generates a csv file in the FV format from a given SDF. As long as one uses the csv file generated in Module 1, no problem should occur in Module 2.

The explanation of the FV format was given in Chapter 1.4.

Property values

The property values should be described in a csv file as follows.

```
CID,a
gdb_1,0.03625663391308918
gdb_2,0.016551941569018975
gdb_3,0.0
gdb_4,0.05238820871210132
```

- **The first line should be `CID,a`.**
- In each of the subsequent lines, the CID and the property value of one molecule should be written in comma-separated style.
- The molecules need not be ordered with respect to CID.

2.4.2 Execution

We use the Python script `lasso_eval_linreg.py` to construct a hyperplane, as outlined in Chapter 1.3.

Arguments

We here revisit the commands explained in Chapter 2.3.

```
$ python lasso_eval_linreg.py ../src/Module_2/sample_instance/FV_Alpha_desc_norm.csv\
 ../src/Module_2/sample_instance/Alpha_norm_values.txt  output_reg.txt 1e-05
```

The meaning of each of the command-line parameters is as follows:

- Parameter 1: A csv file that stores the feature vectors of chemical compounds,
- Parameter 2: A txt file that stores observed values of a chemical property/activity of chemical compounds,
- Parameter 3: Filename where the scalers (weights) and a constant (bias) of a hyperplane will be stored, and
- Parameter 4: The value of λ to be used in LLR.

If the program is invoked with a proper set of command-line parameters, then construction of a hyperplane will begin.

Using the above example of parameter values, where the value of parameter 3 is set to be `output_reg.txt` which stores the weights and bias This file is necessary in Module 3.

Caution on the data set The training set for training the ANN is given in the following two files

- a csv file with feature vectors of chemical compounds, and
- a txt file with observed values for a chemical property/activity.

Both files include the CID number of the corresponding chemical compound. One important point to note is that

each of the CIDs in the former file must appear in the latter

however, the opposite is not enforced. In other words, there might be a chemical compound for which an observed value of some property/activity is listed in the latter file, but whose feature vector is not included in the former. Such values will be silently ignored.

2.4.3 Output

We revisit the command issued as below, and explain the resulting output.

```
$ python lasso_eval_linreg.py ../src/Module_2/sample_instance/FV_Alpha_desc_norm.csv\
 ../src/Module_2/sample_instance/Alpha_norm_values.txt  output_reg.txt 1e-05
```

Standard output

When the above command is issued on the command prompt, the results of the computation process will be displayed on the standard output. An example of this output is stored in the file `src/Module_2/output_reg.txt`.

Content of the output file

Following we explain the contents of the file `output_reg.txt` that contains the values of the weights and bias and some extra information.

The contents of the file `output_reg.txt` would be as follows.

```
# data file : FV_Alpha_desc_norm.csv
# value file: Alpha_norm_values.txt
# R^2 train:            0.9765231781304885
# R^2 test:             0.9612416641336052
# comp time:            0.07280755043029785
# nonzeros:             122
# split_seed in KFold:  8
# fold:                 3
297
0.142088502372554 -0.009993150551405839 0.0 -0.023611185238800913 ...
0.24749148648401187
```

- The first eight rows gives some extra information related to the input files, score of coefficient of determination, etc.
- The ninth row gives the size of the feature vector.
- The tenth row gives the weights.
- The eleventh row give the bias.

Chapter 3

Module 3: Inferring a Chemical Graph Using Lasso Linear Regression and MILP

3.1 Outline

This chapter explains how to use an implementation of a mixed-integer linear programming (MILP) formulation that can infer a vector of graph descriptors given a target value and the weights and bias values of a prediction function (hyperplane) obtained by LLR.

The MILP is implemented in Python, using the PuLP modeling module of the COIN-OR package [6, 7, 8, 9].

The chapter is organized as follows. Chapter 3.2 gives an explanation of the used terms and notation. Chapter 3.3 explains the input and output data of the program, and Chapter 3.4 gives a concrete example of input data and the results from a computation example.

3.2 Terms and Notation

This section explains the terms and notation used in this note.

- **Feature vector**
  A *feature vector* stores numerical values of certain parameters, called *descriptors*. In this work, we choose graph-theoretical descriptors, such as number of non-hydrogen atoms, number of vertices of certain degree, etc.

- **Lasso Linear Regression - LLR**
  Lasso linear regression is one of the methods in machine learning. It provide a means to construct a prediction function (hyperplane) between pairs of feature vectors as input and target data as output.

- **Weights**
  The scalars of the hyperplane are real values, called *weights*. Part of *constructing prediction*

function using LLR is to determine values for each of the weights based on known pairs of feature vectors and target values.

- **Bias**
  The constant term of the hyperplane is a real value, called a *bias*, determined while constructing prediction function.

- **Mixed-Integer Linear Programming (MILP)**
  A type of a mathematical programming problem where all the constraints are given as linear expressions, and some of the decision variables are required to take only integer values. For more details, see any standard reference, e. g. [4].

- **Graph**
  An abstract combinatorial construction comprising a finite set of *vertices*, and a finite set of *edges*, where each edge is a pair of vertices. We treat *undirected* graphs, i. e., graphs where edges are unordered pairs of vertices. For more information, see e. g. [5].

3.3 The Program's Input and Output

This section explains the format of the input and the output of the program. Chapter 3.3.1 illustrates an example of the program's input format. Following, Chapter 3.3.2 illustrates an example of the program's output format, and Chapter 3.3.3 gives a concrete computational example.

3.3.1 Program Input

This section gives an explanation of the input to the program.

First the input requires five files containing
- the feature vector files in normalized and non-normalized form, in csv format
- the weights and bias of a hyperplane in txt format
- the property values file in txt format that contains non-normalized property value of each chemical graph in the underlying data set
- the fringe tree file in txt format

For a common prefix `TT` which the program accepts as a command-line parameter, these files must be saved with file names `TT_desc.csv` and `TT_desc_norm.csv` for the files containing the descriptor names with non-normalized and normalized values, resp., `TT_linreg.txt` for the file containing the weights, and the bias, and `TT_fringe.txt` for the file containing the list of non-isomorphic fringe tree for the underlying data set. The format of these files has been discussed in detail in Chapters 1.4 and 2.4.3. Next, comes the lower and upper bounds for the target value for which we wish to infer a chemical graph based on the prediction function obtained by LLR. Following is a chemical specification given in a textual file, as described in [1], the list of fringe trees given in a txt file as well as a filename prefix for the output files.

Finally, comes a choice of MILP solver program to be used. In `infer_2LMM_LLR.py`, we can choose
- 1: CPLEX, a commercial MILP solver [10], free for academic use.

(Note, in this case the parameter `CPLEX_PATH` in the file `infer_2LMM_LLR.py` must be set to the

correct path of the CPLEX program executable file.)
- 2: CBC, a free and open-source MILP solver. It comes together with the PuLP package for Python [6].

An example of fringe teer file is given below:

File Fringe tree

```
1, C4 0 H1 1, 1, 0 0, 0, 10
2, C4 0 H1 1 H1 1, 1 1, 0 0 0, 0, 10
3, C4 0,, 0, 0, 10
```

Following, Table 3.1 gives a row-by-row explanation of the numerical information in the above fringe tree file.

Table 3.1: Structure of a Fringe Tree File

Value in the file	Explanation
`1, C4 0 H1 1, 1, 0 0, 0, 10`	Index of the fringe tree, (color, depth)-sequence, weight sequence, charge sequence, frequency lower bound, frequency upper bound
`2, C4 0 H1 1 H1 1, 1 1, 0 0 0, 0, 10`	
`3, C4 0,, 0, 0, 10`	

3.3.2 Program Output

This section gives an explanation of the output of the program. If there exists a chemical graph with a feature vector that would result with the given target value as a prediction of the given hyperplane, the program will output the feature vector. In case such a chemical graph does not exist, the program will report this. The next section gives an explanation of the output of the program.

3.3.3 Output Data Format

This section describes the output data of the program as obtained on a personal computer.

Once invoked, the program will print some messages to the standard error stream, which appear on the terminal. Once the MILP solver completes the computation, the status of the computation is printed on the terminal.

Text output on the terminal

Initializing Time: 0.511	# Written to `stdout`
Number of variables: 7157	# The number of variables in the constructed model
- Integer: 6800	# Written to `stdout`
- Binary: 5447	# Written to `stdout`
Number of constraints: 5633	# The number of constraints in the constructed model
Status: Feasible	# Solution status
Solving Time: 2.712	# Time taken by the MILP solver
MILP y*: 1913.633	# Calculated target value in the MILP

Finally, if there exists a feasible solution the program writes to disk two text files. Recall that as a part of the input the program requires a filename used for the output files. Assume that the supplied parameter is `filename`. Then, the resulting two text files are named
- `filename.sdf`
- `filename_partition.txt`.
The file `filename.sdf` contains information on the inferred chemical graph in the SDF (Structure Data File) format. For more information see the official documentation (in English)
`http://help.accelrysonline.com/ulm/onelab/1.0/content/ulm_pdfs/direct/reference/ctfileformats2016.pdf`
for detail.
The file `filename_partition.txt` contains a decomposition of the graph from the file `filename.sdf` into acyclic subgraphs, as described in [11].

3.4 Invoking the Program and a Computational Example

This section explains how to invoke the program and explains a concrete computational example. Following is an example of invoking the program `infer_2LMM_LLR.py`.

3.4.1 Executing the Program

First make sure that the terminal is correctly directed to the location of the `bin` folder, as described in Chapter 3.1.

```
python infer_2LMM_LLR.py prefix_for_the_property lower_bound_for_predicted_value
upper_bound_for_predicted_value instance_file.txt fringe_tree_file.txt prefix_for_output
```

As an example we use the target property Heat of Combustion, that is, the files from the process of constructing prediction function with filename prefix `Hc`, target value lower and upper bounds 1900 and 1920, respectively, the file `instance_file.txt` for a chemical specification, the file `fringe_tree_file.txt` for a list of fringe trees, and `prefix_for_output` is the prefix for the output files.

```
python infer_2LMM_LLR.py ../src/Module_3/sample_instance/Hc 1900 1920
```

../src/Module_3/sample_instance/instance_b4_test_2LMM.txt
../src/Module_3/sample_instance/ins_b4_test_fringe_2LMM.txt test_Hc_b4_test_1900_1920
By executing the above command (without a line break), the following text should appear on the terminal prompt.

Text output on the terminal

Initializing Time: 0.511
Number of variables: 7157
- Integer : 6800
- Binary : 5447
Number of constraints: 5633
Status: Feasible
Solving Time: 2.712
MILP y*: 1913.633

The contents of the output files test_Hc_b4_test_1900_1920.sdf and test_Hc_b4_test_1900_1920_partition.txt are as follows.

File test_Hc_b4_test_1900_1920.sdf

```
1
MILP_2LMM
MILP_2LMM_Linear_Reg
 45 46  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 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 H   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 H   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 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 S   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 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 H   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 H   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 H   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 S   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 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  3  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 O   0  5  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 H   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 H   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 H   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 H   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 H   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 H   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 H   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 O   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
1  2  1  0  0  0  0
1  4  1  0  0  0  0
1 34  2  0  0  0  0
2  3  1  0  0  0  0
4  5  1  0  0  0  0
4  6  2  0  0  0  0
6  7  1  0  0  0  0
6  8  1  0  0  0  0
8  9  1  0  0  0  0
9 10  1  0  0  0  0
9 11  1  0  0  0  0
```

```
  9 12  1  0  0  0  0
 12 13  1  0  0  0  0
 12 14  1  0  0  0  0
 12 18  1  0  0  0  0
 14 15  1  0  0  0  0
 14 16  1  0  0  0  0
 14 17  1  0  0  0  0
 18 19  2  0  0  0  0
 18 20  2  0  0  0  0
 18 21  1  0  0  0  0
 21 22  1  0  0  0  0
 21 25  2  0  0  0  0
 22 23  1  0  0  0  0
 22 24  2  0  0  0  0
 25 26  1  0  0  0  0
 25 27  1  0  0  0  0
 27 28  1  0  0  0  0
 27 35  2  0  0  0  0
 28 29  1  0  0  0  0
 30 31  1  0  0  0  0
 30 32  1  0  0  0  0
 30 34  1  0  0  0  0
 30 38  1  0  0  0  0
 32 33  1  0  0  0  0
 34 38  1  0  0  0  0
 35 36  1  0  0  0  0
 35 38  1  0  0  0  0
 36 37  1  0  0  0  0
 38 39  1  0  0  0  0
 39 40  1  0  0  0  0
 39 41  1  0  0  0  0
 39 42  1  0  0  0  0
 42 43  2  0  0  0  0
 42 44  1  0  0  0  0
 44 45  1  0  0  0  0
M  CHG  2  22   1  23  -1
M  END
$$$$
```

File `test_Hc_b4_test_1900_1920_partition.txt`

```
4
30
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
34
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
35
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
38
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
5
30 34
0 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
30 38
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
34 1 4 6 8 9 12 18 21 25 27 35
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
34 38
0 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
35 38
0 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Following, Table 3.2 gives a row-by-row explanation of the numerical information in the above partition file.

Table 3.2: Structure of a Partition Information

Value in the file	Explanation
4	Number of base vertices
30	The index of a base vertex in the input SDF
0 30	Core height lower and upper bound
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	Indices of fringe trees in the fringe tree file
34	
0 30	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
35	
0 30	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
38	
0 30	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
5	Number of base edges
30 34	Indices of vertices in the base edge from the input SDF
0 0	Core height lower and upper bound
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	Indices of fringe trees in the fringe tree file
30 38	
0 30	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
34 1 4 6 8 9 12 18 21 25 27 35	
0 30	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
34 38	
0 30	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
35 38	
0 0	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	

Chapter 4

Module 4: Listing Chemical Isomers of a Given Chemical Graph

4.1 Introduction

This chapter explains how to use the program for listing chemical isomers of a given chemical graph [1].

The chapter is organized as follows. Chapter 4.2 explains some of the terminology used throughout the text. Chapter 4.3 gives an explanation of the program for generating chemical isomers of a given chemical graph, explaining the input, output, and presents a computational example.

4.2 Terminology

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

- Chemical Graph
  AA 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-unsuppressed graphs possibly with multi-valence atoms or charged atoms (cation and anion). For more details, please see [1].

- 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 [12].

4.3 Program for Generating Chemical Isomers

4.3.1 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 chemical graph. We call the program `generate_isomers`.

The Program's Input

The input to the `generate_isomers` program consists of ten necessary and one optional item.
First, comes information of a chemical graph (in SDF format).
Second, comes a time limit in seconds on each of the stages 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.

Recall that the partition information is stored as a text file and is an output of Module 3.

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.

4.3.2 Executing the Program and a Computational Example

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

Compiling and Executing the `generate_isomers` Program

Again, 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.sdf a b c d e f output.sdf instance_partition.txt instance_fringe_tree.txt`
  Above, `generate_isomers` is the name of the program's executable file, and the remaining command-line parameters are as follows:
  `instance.sdf` a text file containing a chemical specification
  `a` upper bound (in seconds) on the computation time on each stage 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.

Computational Example

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

- Input graph: File `sample.sdf` from the folder `../src/Module_4/sample_instance`
- Time limit: 10 seconds
- Upper limit on the number of partial feature vectors: 1000000
- Number of graphs per base vertex or edge: 5
- Global time limit for enumeration of paths: 300
- Global upper bound on number of paths: 1000000
- 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 `../src/Module4/sample_instance`.

- Fringe tree information of the input and output graphs: File `sample_fringe_tree.txt` from the folder `../src/Module4/sample_instance`.

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

```
./generate_isomers ../src/Module_4/sample_instance/sample.sdf 2 10000 5 10 10000 2
                   output.sdf ../src/Module_4/sample_instance/sample_partition.txt
                   ../src/Module_4/sample_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 = 192
Number of generated graphs = 2
Total time :  0.081s.
```

Contents of the file output.sdf

```
1
BH-2LM-M
BH-2LM-M
44 45 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 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 O 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 S 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 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 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 S 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 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
1 2 1 0 0 0 0
1 6 1 0 0 0 0
1 11 2 0 0 0 0
2 3 1 0 0 0 0
2 6 1 0 0 0 0
2 27 1 0 0 0 0
3 28 1 0 0 0 0
4 5 1 0 0 0 0
4 6 1 0 0 0 0
4 26 2 0 0 0 0
5 29 1 0 0 0 0
6 7 1 0 0 0 0
7 8 1 0 0 0 0
7 30 1 0 0 0 0
7 31 1 0 0 0 0
8 9 2 0 0 0 0
8 10 1 0 0 0 0
10 32 1 0 0 0 0
```

```
11 12 1 0 0 0 0
11 13 1 0 0 0 0
12 33 1 0 0 0 0
13 14 2 0 0 0 0
13 15 2 0 0 0 0
13 16 1 0 0 0 0
16 17 1 0 0 0 0
16 18 1 0 0 0 0
16 34 1 0 0 0 0
17 35 1 0 0 0 0
17 36 1 0 0 0 0
17 37 1 0 0 0 0
18 19 1 0 0 0 0
18 38 1 0 0 0 0
18 39 1 0 0 0 0
19 20 1 0 0 0 0
19 21 2 0 0 0 0
20 40 1 0 0 0 0
21 22 1 0 0 0 0
21 23 1 0 0 0 0
22 41 1 0 0 0 0
23 24 2 0 0 0 0
23 42 1 0 0 0 0
24 25 1 0 0 0 0
24 43 1 0 0 0 0
25 26 1 0 0 0 0
26 44 1 0 0 0 0
M END
$$$$
2
BH-2LM-M
BH-2LM-M
44 45 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 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 O 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 S 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 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 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 S 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 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
0.0000 0.0000 0.0000 H 0 0 0 0 0 0 0 0 0 0 0 0
1 2 1 0 0 0 0
1 6 1 0 0 0 0
1 11 2 0 0 0 0
2 3 1 0 0 0 0
2 6 1 0 0 0 0
2 27 1 0 0 0 0
3 28 1 0 0 0 0
```

```
4 5 1 0 0 0 0
4 6 1 0 0 0 0
4 26 2 0 0 0 0
5 29 1 0 0 0 0
6 7 1 0 0 0 0
7 8 1 0 0 0 0
7 30 1 0 0 0 0
7 31 1 0 0 0 0
8 9 2 0 0 0 0
8 10 1 0 0 0 0
10 32 1 0 0 0 0
11 12 1 0 0 0 0
11 13 1 0 0 0 0
12 33 1 0 0 0 0
13 14 1 0 0 0 0
13 34 1 0 0 0 0
13 35 1 0 0 0 0
14 15 1 0 0 0 0
14 16 1 0 0 0 0
14 36 1 0 0 0 0
15 37 1 0 0 0 0
15 38 1 0 0 0 0
15 39 1 0 0 0 0
16 17 2 0 0 0 0
16 18 2 0 0 0 0
16 19 1 0 0 0 0
19 20 1 0 0 0 0
19 21 2 0 0 0 0
20 40 1 0 0 0 0
21 22 1 0 0 0 0
21 23 1 0 0 0 0
22 41 1 0 0 0 0
23 24 2 0 0 0 0
23 42 1 0 0 0 0
24 25 1 0 0 0 0
24 43 1 0 0 0 0
25 26 1 0 0 0 0
26 44 1 0 0 0 0
M END
$$$$
```

Bibliography

[1] J. Zhu, N. A. Azam, K. Haraguchi, L. Zhao, H. Nagamochi, and T. Akutsu. A Method for Molecular Design Based on Linear Regression and Integer Programming, Arxiv preprint, arXiv:2107.02381v2.

[2] T. Akutsu and H. Nagamochi. A Mixed Integer Linear Programming Formulation to Artificial Neural Networks, in Proceedings of the 2019 2nd International Conference on Information Science and Systems, pp. 215–220, https://doi.org/10.1145/3322645.3322683.

[3] HSDB in PubChem `https://pubchem.ncbi.nlm.nih.gov` (accessed on February 1st, 2021)

[4] J. Matousek and B. Gärtner. Understanding and Using Linear Programming. Springer, 2007.

[5] M. S. Rahman. Basic Graph Theory. Springer, 2017.

[6] A Python Linear Programming API, `https://github.com/coin-or/pulp`.

[7] Optimization with PuLP, `http://coin-or.github.io/pulp/`.

[8] The Python Papers Monograph, `https://ojs.pythonpapers.org/index.php/tppm/article/view/111`.

[9] Optimization with PuLP, `https://pythonhosted.org/PuLP/`.

[10] IBM Cplex Optimizer `https://www.ibm.com/analytics/cplex-optimizer`.

[11] 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

[12] 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. 2021.