Vienna-RNL

A Library for Chemical Reaction Networks

Ivo L Hofacker and Peter F Stadler

Published by TBI

1 Introduction

The Vienna Reaction Network Library `Vienna-RNL` implements basic set-theoretic operations on chemical reaction networks. It provides basic ANSI C data structures for chemical reactions and their networks, operations such as the union, intersection, or difference of chemical reaction networks. It is intended for the use in conjunction with the user's own C programs or PERL scripts.

1.1 No Warranty

2 Input and Output Files

`Vienna-RNL` uses its own format for encoding chemical reaction networks. An example is the file `test.rct`, which is part of the distribution.

```
>Test-Murkelei
# A + C + D + E <=> F + G + H ;
R1 Ca(OH)_2 + CO_2 -> CaCO_3 + H_2O # Blah blah
R2 Ca(OH)_2 -> CaO + H_2O @ murkl
R3 2 *   H_2O <=> 2 *     H_2     + O_2 @ bazong ; 345
R4 2 * H_2O -> 2 * H_2 + O_2 ; 3452345.23 @ urgl
R5 2 * H_2O -> 2 * H_2 + O_2 ;
R6 2 * H_2O <=> 2 * H_2 + O_2 ;
P1 [H_2O] >< H_2O ;
P2 [H_2]  >< H_2 ;
```

The line starting with `>` defines the name of network. All lines starting with `#` are comments that are ignored by `Vienna-RNL`.

Each reaction must be contained on a single line (of arbitrary length). A reaction line must begin with a reaction label (arbitrary string without white spaces). Each reactant is specified by a unique name, which again can be an arbitrary string that must not contain white space. Multiplicities are specified by a prefix of the form `2 *`. The space between the number and the multiplication operator are required, in fact any two tokens on the line must be separated by white space as in the example above.

Special information can be included using the `@` character for a reaction name and `;` for a reaction rate. This information is included into the reaction data-structure.

The `#` character is used to mark the rest of the line as a comment.

Each reaction line must contain one of the following reaction symbols

- `<=>` or `<->` for a reversible reaction
- `->` for an irreversible reaction
- `><` or `><` for a reversible pseudo-reaction, i.e., exchange with an external pool.

In addition to the `.rct` file, `Vienna-RNL` contains routines for reading and writing `metatool` formatted files. For the format specifications see *[1]*, an examples is included with the distribution.

3 Data Types

Reactions networks are represented by the library using a a hierarchy of C structs. In the Perl interface these are mapped to equivalent objects.

Network [Data type]

A Network is represented by a C struct containing a name, a list of reactions, and a list of reactants. The length of the reactions and species list is given by the num_react and num_species variables. For convenience, we also include the number of reversible reactions num_rev_react. Thus, the full C declaration reads:

```
struct Network {
  char *name;
  int num_react;
  int num_rev_react;
  int num_species;
  Reactant *species;
  Reaction *reactions;
};
```

Reactant [Data type]

A reactant is represented by a string (typically the chemical formula), and a unique numeric id. The C declaration reads:

```
struct Reactant {
  char *formula;
  int ID;
};
```

Reaction [Data type]

A reaction is presented primarily by the list `eq` of equation terms. The length of this list is stored in `members`, a reaction is reversible if the flag `reversible` is not zero, `pseudo` marks pseudo reactions (exchange with an external pool). In addition the reaction structure contains strings holding the name and an arbitrary comment. The `rate` variable may be used to store a reaction rate.

```
typedef struct Reaction {
  char *name;
  char *comment;
  int ID;
  int members;
  int reversible;
  int pseudo;
  double rate;
  EqTerm *eq;
} Reaction;
```

Eqterm [Data type]

An equation term consists simply of a stoichiometric coefficient `c` and a pointer to the corresponding reactant. For non-reversible reactions the stoichiometric coefficient should be negative for the educts and positive for the products. The Reactant pointer `r` is usually just a pointer into the substrate list of the network.

```
typedef struct Eqterm {
  double c;
  Reactant *r;
} EqTerm;
```

4 Functions

4.1 Input Output

`Network*` **ReadNetwork** (`FILE *fp`); [Function]
: read Network from `.rct` file.

`void` **fPrintNetwork** (`FILE *fp, Network *NET`); [Function]
: print Network in `.rct` format.

`void` **fPrintReaction** (`FILE *fp, Reaction *R`); [Function]
: print a single reaction in `.rct` format.

`Network*` **ReadMetatool** (`FILE *fp`); [Function]
: read Network in Metatool format.

`void` **fPrintMetatool** (`FILE *fp, Network *NET`); [Function]
: print Network in Metatool format.

`Network*` **ReadPalssonMat** (`FILE *fp`); [Function]
: read Network as stoichiometric matrix in Palsson format *[2]*.

`void` **fPrintStoichMat** (`FILE *fp, double **S, Network *N`); [Function]
: print stoichiometric matrix of network *N* in Palsson format. The matrix S can be generated by calling `StoichMat` function.

`void` **Network2gml** (`FILE *fp, Network *N`); [Function]
: print the network as bipartite graph in gml format *[4]*, suitable for various graph layout programs.

`void` **Network2dot** (`FILE *fp, Network *N`); [Function]
: print the network graph in dot format, suitable for the graph layout program `dotty` `http://seclab.cs.ucdavis.edu/~hoagland/Dot.html`, see also *[3]*.

4.2 Network operations

`Network*` **CopyNetwork** (`const Network *N`); [Function]
: returns a newly allocated copy of the Network **N*.

`Network*` **NetCup** (`Network *N1, Network *N2`); [Function]
: computes the union of the two Networks *N1* and *N2*, consisting of all substrates and all reactions occurring in either network.

`Network*` **NetCap** (`const Network *N1, const Network *N2`); [Function]
: computes the intersection of the two Networks *N1* and *N2*, consisting of only those substrates and reactions that occur in both networks.

`Network*` **NetDiff** (`const Network N1, const Network N2`); [Function]
: computes the difference of the two Networks *N1* and *N2*. The reaction difference Network contains all reaction occurring in *N1* but not *N2*, and all substrate occurring in the remaining reactions.

`Network*` **NetStrongDiff** (`const Network *N1, const Network *N2`); [Function]
: computes the strong difference of the two Networks *N1* and *N2*. The resulting new network contains only those substrates occurring in *N1* but not *N2*, and only those reactions from *N1* that can be performed with the remaining substrates.

`Reaction` **CopyReaction** `(const Reaction` *R*`, const Reactant *`*S*`, int` *numS*`);` [Function]

returns a new copy of th reaction *R*. New memory is allocated for the equation terms, but not the substrates. Instead substrate pointers are taken from **S*.

`Reactant*` **find_formula** `(const char *`*formula*`, const Reactant *`*S*`, int` *num_*`S);` [Function]

searches for the substrate described by **formula* in the list of reactants *S*. Returns a pointer to the found reactant or NULL if none was found. Assumes a sorted list of reactants **S*.

`Reaction` ***find_reaction** `(const Reaction` *R*`, const Reaction *`*RL*`, int` *numR*`);` [Function]

searches for the reaction *R* in the list **RL*. Returns a pointer to the found reaction or NULL if none was found. Assumes the list **RL* is sorted.

4.3 Utility functions

`int` **comp_reactants** `(const void *`*a*`, const void *`*b*`);` [Function]

auxiliary function for sorting lists of reactants with `qsort`.

`int` **comp_reactions** `(const Reaction *`*R1*`, const` [Function]

Reaction **R2*); auxiliary function for sorting Reactions with `qsort`.

`int` **comp_networks** `(Network *`*N1*`, Network *`*N2*`);` [Function]

compare two Networks, suitable for sorting a list of networks with `qsort`.

`void` **sort_reactions** `(Network *`*N*`);` [Function]

sorts the reactions in Network *N*.

`void` **FreeReaction** `(Reaction` *R*`);` [Function]

Free the memory allocated for reaction *R*.

`void` **FreeNetwork** `(Network *`*N*`);` [Function]

Free the memory allocated for network *N*.

`double` ****StoichMat** `(Network *`*N*`) ;` [Function]

return the stoichiometric matrix of network *N* as 2-d array of doubles.

`double` ****AdjMat** `(Network *`*N*`);` [Function]

return the adjacency matrix of the network graph.

5 Perl Interface

System Requirement: Perl 5.004 and later versions must be installed.

The PERL interface allows easy access to the complex C data structures of `Vienna-RNL` from the scripting language Perl. The interface is automatically generated using `SWIG`. All library functions are available through Perl subroutines of the same name. C variables can be accessed from Perl via an object oriented interface, that mimics the C structs. Thus the two equivalent C and Perl expressions

```
Network *N;
N->reactions[2]->eq[3]->formula;
$N->{reactions}->get(2)->{eq}->get(3)->{formula};
```

both retrieve the formula of the 4th reactant in the 3rd reaction of the network N (remember that indices start at 0).

As an example part of the self-test script of the installation is shown below.

```
#!/usr/bin/perl
use Networks;

open(FP, '<../EXAMPLES/test.rct') or die "can't open test.rct";
$N1 = Networks::ReadNetwork(*FP);
close(FP);

print "$N1->{num_react}\n"; # should be 6
print "$N1->{num_species}\n"; # should be 9
# print formula of 1st reactant in 3rd reaction
print $N1->{reactions}->get(2)->{eq}->{r}->{formula};

open(FP, '<../EXAMPLES/Io-reduced.rct') or die "can't open Io-reduced.rct";
$N2 = Networks::ReadNetwork(*FP);
close(FP);
print "read Network $N2->{name} with $N2->{num_react} reactions ",
       "and $N2->{num_species} species\n";

$Ncup = Networks::NetCup($N1, $N2);
print "Union has $Ncup->{num_react} reactions ",
      "and $Ncup->{num_species} species\n";

$Ncap = Networks::NetCap($N1, $N2);
print "Intersection has $Ncap->{num_react} reactions ",
      "and $Ncap->{num_species} species\n";
```

6 Examples

The distribution includes two executable programs, `reactions` and `testit` that are small applications of `Vienna-RNL`.

The program `reactions` converts various I/O formats. At this point input can be provided in the `.rct` format described above, in Metatool format, or in the form of Palsson's matrices. Output formats include the substrate adjacency matrix, GML (graph meta language) which can be used e.g. as input for the graph editor `graphlet` *[4]*, and input for the graph layout program `dotty`. The program `reactions` reads from stdin and writes to stdout.

```
./reactions [-i #in_format] [-o #out_format] < infile [ > outfile ]
```

The input and output format options are summarized below

```
 # |  format      | in     out
-------------------------------
 0 |  .rct        | Y       Y   [default]
 1 |  Metatool    | Y       Y
 2 |  Palsson     | Y       N
   |              |
 6 |  SubstrMat   | N       Y
 7 |  .dot        | N       Y
 8 |  .gml        | N       Y
```

The program `testit` is intended as a test program for the library. It takes two `rct` files as arguments and performs a number of simples operations.

The Perl module can be checked by typing

```
cd Perl
make check
```

The output, apart from various messages from `make`, should look something like

```
1..2
ok 1
Warning: removing duplicate reaction
R5         2 * H_2O -> 2 * H_2 + O_2
Warning: removing duplicate reaction
R6         2 * H_2O <=> 2 * H_2 + O_2
ok 2
Ca(OH)_2N1
 $VAR1 = bless( {
      'name' => 'Test-Murkelei',
      'num_react' => 6,
      'num_rev_react' => 4,
      'num_species' => 9,
      'species' => bless( {
            'formula' => 'CO_2',
            'ID' => 0
            }, 'Networks::Reactant' ),
      'reactions' => bless( {
            'name' => 'P2',
            'comment' => undef,
            'ID' => 0,
            'members' => 2,
            'reversible' => 1,
            'pseudo' => 1,
            'rate' => '0',
            'eq' => bless( {
                 'c' => '1',
                 'r' => bless( {
                      'formula' => 'H_2',
                      'ID' => 4
                      }, 'Networks::Reactant' )
                  }, 'Networks::EqTerm' )
             }, 'Networks::Reaction' )
       }, 'Networks::Network' );

read Network Io with 79 reactions and 27 species
Union has 85 reactions and 35 species
Intersection has 0 reactions and 0 species
```

7 References

1. T. Pfeiffer, I. Sánchez-Valdenebro, J. C. Nuño, F. Montero and S. Schuster METATOOL: For Studying Metabolic Networks Bioinformatics 15 (1999) 251-257.
2. Schilling, C.H., Letscher, D., and Palsson, B.O., "Theory for the Systemic Definition of Metabolic Pathways and their use in Interpreting Metabolic Function from a Pathway-Oriented Perspective", J. theor. Biol, 203: 229-248, (2000).
3. An open graph visualization system and its applications to software engineering E.R. Gansner and S.C.North Software — Practice and Experience 30(11), pp. 1203-1233, September 2000.
4. M. Himsolt, GML — Graph Modelling Language, University of Passau, http://infosun.fmi.uni-passau.de/Graphlet/GML/, 1997.

Table of Contents