Table of Contents

1 Abstract 2

2 Introduction 2

3 Parse a CellDesiger file into Matlab 5

4 Width and colour attribute modifications 7

4.1 Change the thickness of reaction links 7

4.2 Change the colour of reaction links 9

4.3 Export coloured nodes to a text file 11

4.4 Change the colour of metabolise nodes 12

4.5 Change colour of metabolite nodes of reactions 15

5 Error corrections 20

5.1 Check and correct species names 21

5.2 Repair the XML file 23

5.3 Check reaction names 24

6 Annotation 26

6.1 Free-text annotation 26

6.2 MIRIAM information 29

6.3 Other annotation types 32

1 Abstract

The CellDesigner interface packages is developed to serve as a bridge between constraint-based modelling and the popular process diagram editor, CellDesigner, for visualisation and annotation of metabolic network. The package can parse an XML file created by CD, change the thickness of reaction links and colours of nodes in the XML file according to the flux values obtained from constraint-based modelling. There are also auxiliary functions not only to compare a network model stored in the CD XML file and a COBRA model in the Matlab file and correct the discrepancies between them, but also integrate different types of omics data into the XML file in line with the Miriam standard.

2 Introduction

The overall structure of the package is illustrated in Figure 1. All functions implemented in the package are described in detail in each Matlab scripts and summarised in Table 1. The main methods and functionalities of the package are demonstrated in a quick tutorial below:

Most of the package functions require one of the three inputs:

1) An XML file created by CellDesigner (CD) (such as those pathways retrieved from http://www.reactome.org/); other SBML-compliant file formats can be opened in CellDesigner and exported as a new CD XML file for the package. The popular graphical notation formats such as BioPAX and SBGN can be easily converted into CD XML file using a published Cytoscape plugin, BiNoM (https://binom.curie.fr/).

2) A data structure generated from parsing the XML file using “parseCD” function

3) A COBRA Matlab structure. The COBRA Matlab structure is needed for FBA modelling to generate fluxomic data that can be translated into widths and highlighted using different colours. The COBRA Matlab structure is also an annotation repository accommodating

various omics data collected during the reconstruction process. To integrate the XML files with other user-defined annotations, it would be required to formulate the annotations into data arrays as new fields for the COBRA model structure, which are used as inputs for “addAnnotation” and “addMiriam” functions.

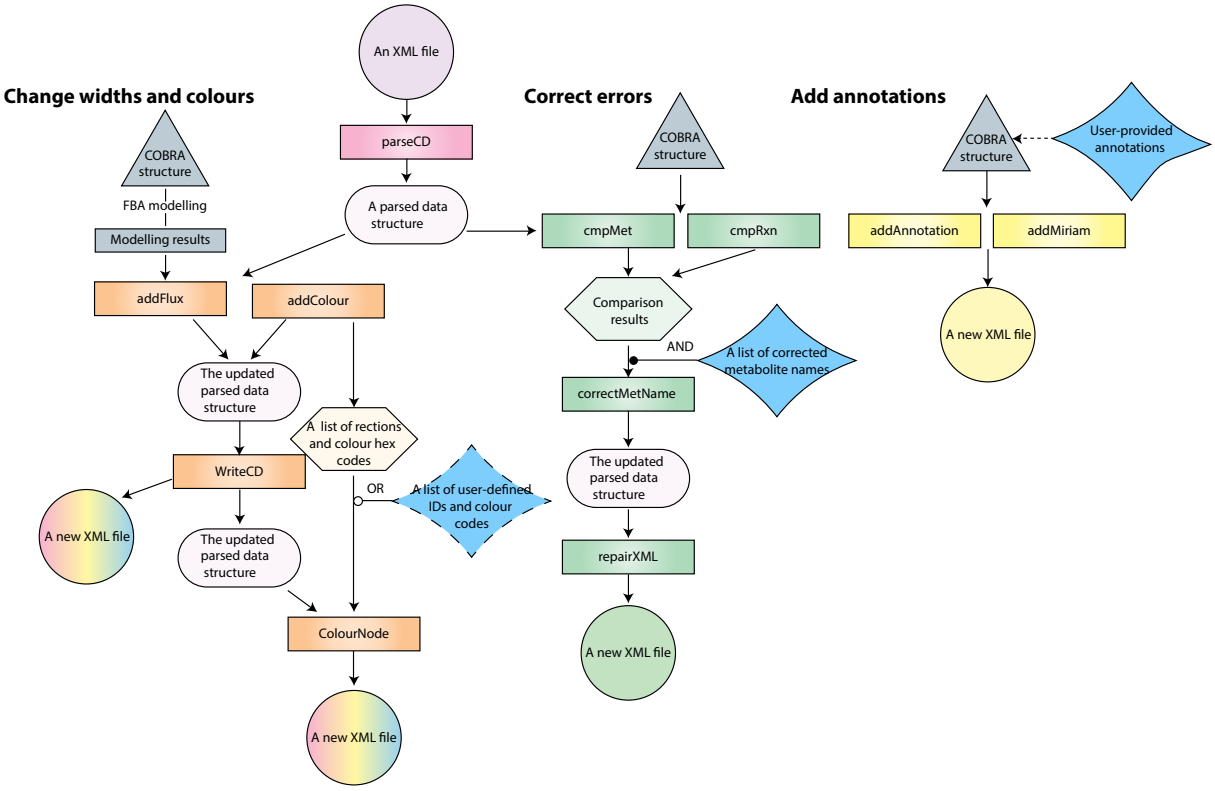


Figure 1: Schematic overview of the CellDesigner package.

Table 1: A summary of functions in the package

Name	Main functions
Read	
parseCD	Import reaction and metabolite ID, names, width and colour into Matlab
Modify	
cmpRxn	Comparison of reactions in the CD model imported by “parseCD” and COBRA model

cmpMet	Comparison of metabolites in the CD model imported by “parseCD” and COBRA model
addFlux	Add flux values to a list of reactions in the CD model. The width of lines can be visualised in CD.
addColour	Change colour of each reaction link
colourNode	Change the colours of metabolite nodes
colourAll	Change the colours of all metabolite nodes to a specific colour
colourNodeWhite	Highlight the metabolite nodes using the same colour scheme as in "addColour" function while changing the colours of rest nodes to white
Correct	For correction of discrepancies between the CD model and COBRA model
correctMetName	Correct the inconsistent species name in the test model (identified by the 'cmpMet' function) according to a reference list of species names.
writeCD	Write the parsed CD model structure to a CD-compatible XML file.
repairXML	Write the corrected CD model structure to an XML file
writeXML	Write the parsed CD structure to a compatible XML file
writeTXT	Write a txt file for online PD map to highlight specific reaction nodes.
Annotation	
addAnnotation	Retrieve omics data from a COBRA model structure and add them to a CellDesginer XML file; The omics data will be shown as texts in CellDesigner or ReconMap online
addMiriam	Add Miriam information to CellDesigner XML file; the Miriam information is retrieved from a COBRA model structure using Metbolite/Reaction IDs as the name of entry. The omics data will be shown as texts hyperlinking to external databases in CellDesigner or ReconMap online.
Auxiliary functions	Convert the CD model into a variable as an input for other functions
updateCD	Correct (Update) the species name according to a reference list of the names.
retrieveMet	Retrieve all the identifiers for a metabolite in the parsed CD model structure; there could be multiple identifiers for a metabolite.
position	Retrieve the value of an attribute (str_ID) in the line (str_long) of the XML file and identify the starting and ending indices of the value in the attribute line of the XML file.
readCD	Convert the a type of the parsed model structure (organised by reaction) into the other type of the parsed model structure (organised by property (namely, ID, width, colour, etc.)

3 Parse a CellDesiger file into Matlab

The “parseCD” function read a CellDesigner (CD) file into two types of Matlab variable structures. The first type is suitable for user to modify the attributes of reactions, such as ‘color’ and ‘width’ (these key words are retrieved directly word for word). The second is similar to COBRA model structure and can be used as the inputs for other Matlab functions.

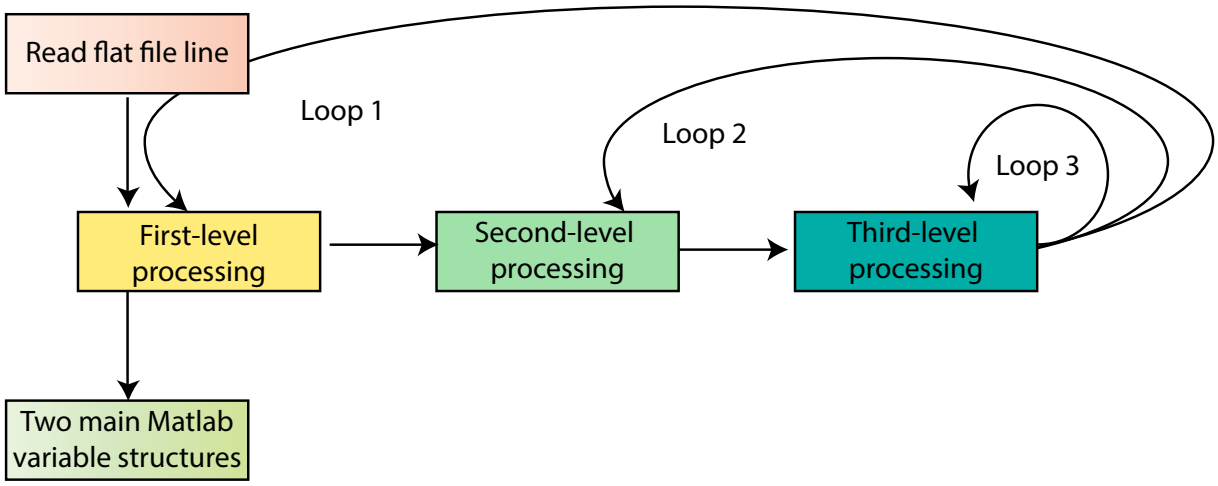
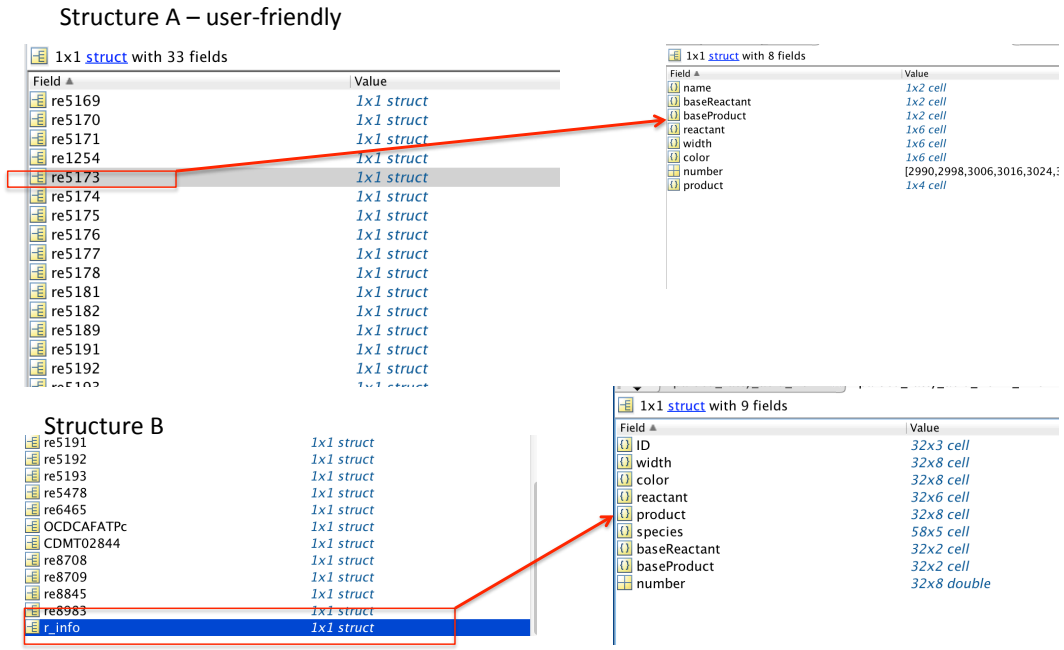


Figure 2: A flow diagram of the “parseCD” function. The input file is processed by different processing units, which can extract different levels of information. For example, each processing unit is responsible for iterative retrieval of information in the XML line with desirable tags. For example, the first-level processing unit reads reaction IDs, the second level reads metabolite ID, and the third reads the property of the link on the graphic layout such as width and colour. Once all lines from the input file are processed, all information are formulated into two Matlab variable structures for people or functions to use.



Structure type A can be converted into type B using function 'readCD'

Figure 3: Two types of Matlab structures generated by "parseCD" function.

Just enter "parseCD" in the command window of Matlab, a file dialogue will pop up asking to choose the XML file that needs to be parsed; alternatively, if the XML file name is known, the file name with single quotation marks can be typed in the parenthesis as follows:

```
>> parsed= parseCD('fatty_acid.xml')
```

"parsed" is the return variable of the function storing the parsed information of the XML file.

whereas the "r_info" is used for algorithms to exchange and process information, the "parsed" form is used for user-end to read.

Structure A is good user-friendly but takes much longer time to produce.

Structure B is not good for human to read but is more suitable for other Matlab functions to access the parsed information.

The following tutorial is intended to provide users a quick guide to getting started using the useful functions of the package. The tutorial use a network layout of human fatty acid metabolism drawn using CellDesigner based on Recon2 – the human metabolic network reconstruction. This tutorial requires the CellDesigner file “atty_acid.xml”. The omics data are extracted from the Matlab file of recon2 - “recon2.mat”.

4 Width and colour attribute modifications

4.1 Change the thickness of reaction links

The following command parses the network layout of fatty acid in CellDesigner format into a Matlab variable.

```
>> parsed= parseCD('fatty_acid.xml')
```

A COBRA function “optimizeCbModel” can be called to obtain a flux distribution for Recon2,

```
>> FBA=optimizeCbModel(recon2)
```

where ‘recon2’ is the Matlab variable structure of recon2 (human metabolic reconstruction)

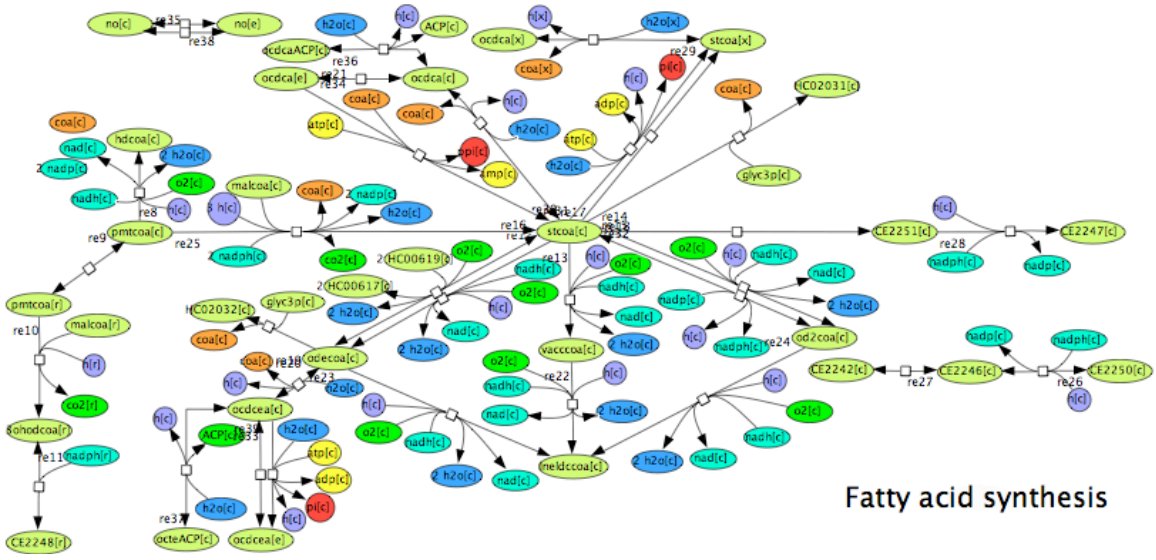


Figure 4: The graphic layout of the fatty acid synthesis network.

The widths of lines are changed according to flux calculated using FBA using the following commands:

1) Change the width of lines.

```
>> [parsed_new, flux] = addFlux(recon2,FBA,parsed,parsed.r_info.ID(1:32,3))
```

2) Write a new XML file.

```
>> writeCD(parsed_new,'fatty_acid_new.xml')
```

A new file, “fatty_acid_new.XML”, can be seen in the current working directory. Open the file with CellDesigner to visualise the network layout. The screenshot of network layout is shown below:

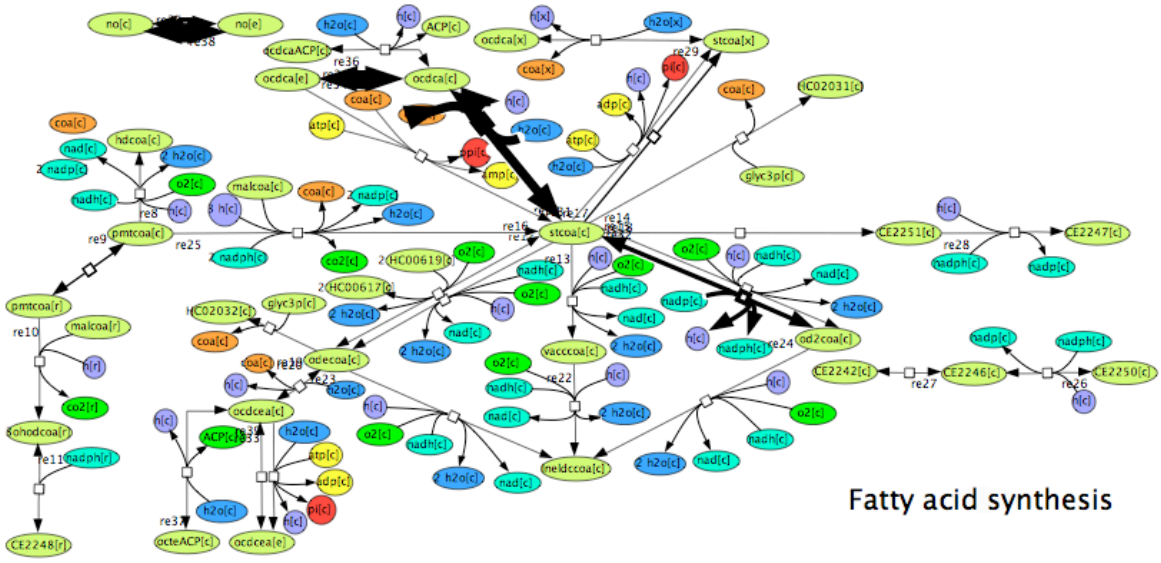


Figure 5: The result of "addFlux" function.

4.2 Change the colour of reaction links

Using the following two commands, we can change the colour of reaction links to reflect the flux values (e.g., the higher the flux values, the darker the colours). In the example, the reaction carrying high fluxes are highlighted in red, whereas those with comparatively low fluxes are highlighted in blue.

```
>>[parsed_new] = addColour(parsed_new,(parsed.r_info.ID(1:32,3)))
```

This command returns an updated parsed data structure, whose the colour attributes are modified for some reactions.

The CD file uses a variation of the standard hex colour codes to specify colours. Examples of colour codes can be found at http://www.color-hex.com. The colour is represented by a six-digit code, e.g., '#b3c628', but, for CellDesigner to recognise, it is needed to remove '#' from the code and add 'ff' before 'b3c628', which converts the six-digit code into an eight-digit code, 'ffb3c628'. In the tutorial, it is not necessary to include '#' in the colour code when

calling package functions. The functions can automatically check if a prefix, '#', exists in the colour code and remove the prefix if it exists.

Note:

The second argument may contain two columns. The first column is a list of reaction IDs, while the second is optional containing a list of hex colour codes. When the second column is empty, a default colour scheme will be used for highlighting; on the other hand, when the colour codes are provided in the second column, these provided codes will be used to highlight reactions.

There are five predefined sets of colour palettes. Each palette number (1-5) is used as the third agurment for "addColour" function. For example, Red pt1 Color Palette (5) is chosen in the following command.

```
>>[parsed_new] = addColour(parsed_new,(parsed.r_info.ID(1:32,3)), 5)
```

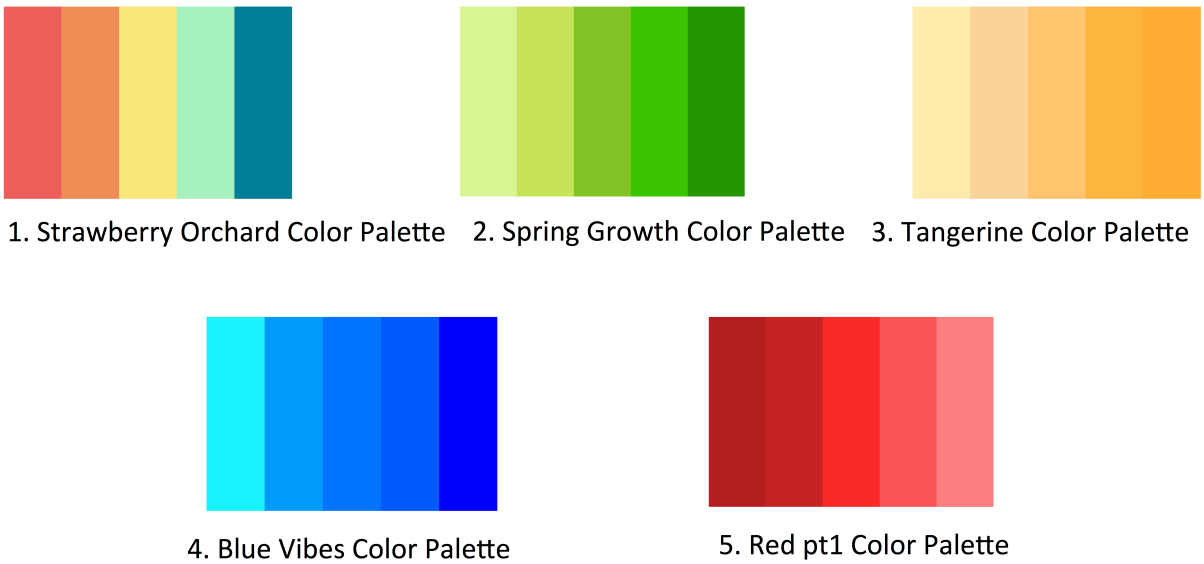


Figure 6: Five predefined colour palettes.

```
>> writeCD (parsed_new,'fatty_acid_new.xml')
```

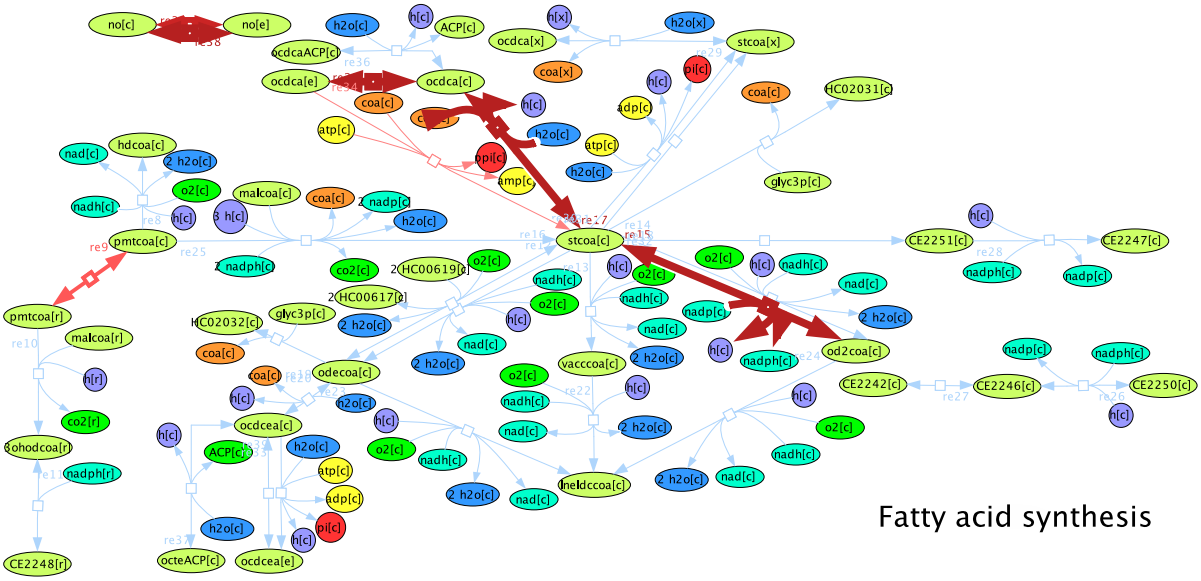


Figure 7: The reaction links are highlighted by "addColour" function.

4.3 Export coloured nodes to a text file

A simple function 'writeTXT' can be called to produce a text file containing a list of reaction IDs and a list of colour hex codes. The text file can be uploaded to ReconMap online to highlight reaction links on ReconMap. This function uses the return variable of "addColour" function as the input.

```
>> writeTXT(list_nodes,'test.txt')
```

"list_nodes" contains a list of the reaction IDs and a list of colour codes; it is one of the output variables of "addColour" function.

one of the return variables of the function “addColour”, “listRxn” can be used as the “list_nodes” .

“text.txt” is a name of the text file.

4.4 Change the colour of metabolise nodes

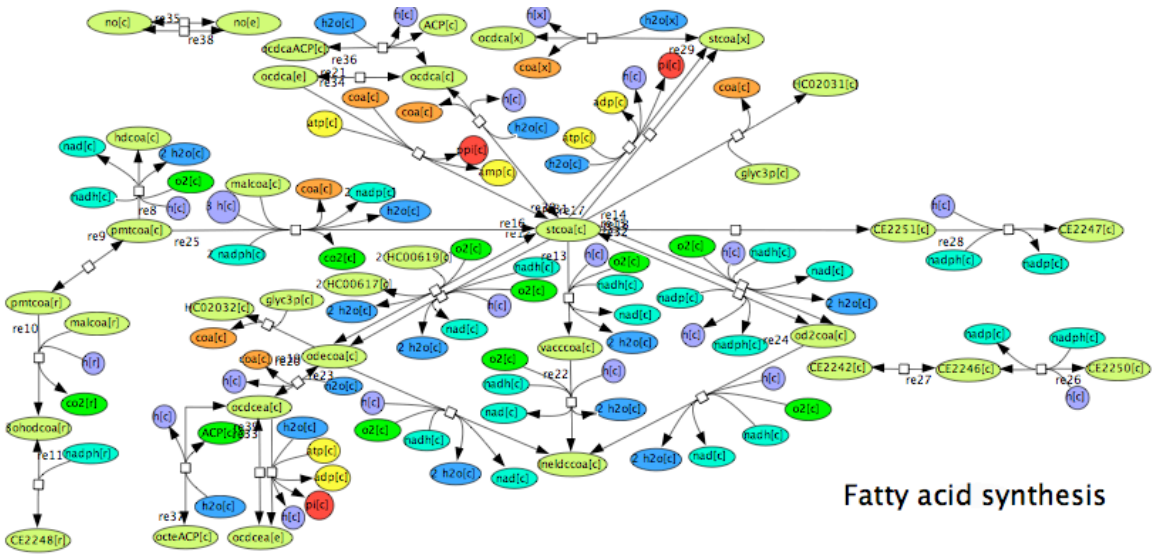


Figure 8: The unmodified network layout.

“colourNode” function can be used to change the colours of metabolite nodes. “list_Met” can be set to “false”, then the function will not highlight metabolite and only reaction nodes, specified in a reaction list, are highlighted.

The return variable of the “addColour” can be use as the input for “colourNode”. Then, the same colour of the reaction link will be used to highlight the metabolite nodes involved in a reaction.

>>
[parsed_new,var,final_list]=colourNode(parsed,'fatty_acid_new.xml',parsed.r_info.ID(:,2),'false')

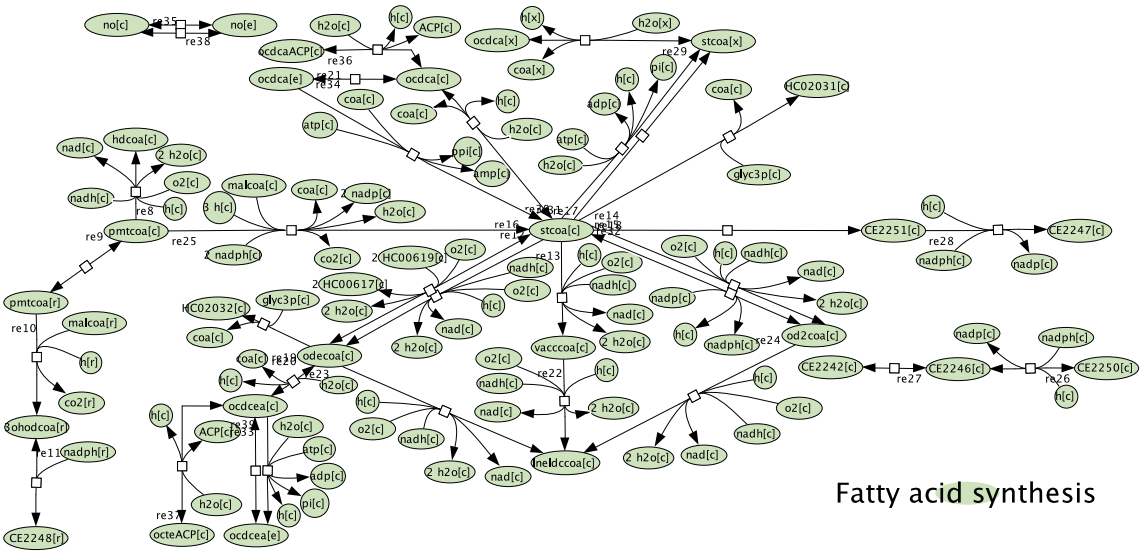


Figure 9: The modified graphic layout generated by "colourNode" function.

>> [parsed_new, flux] = addFlux(recon2,FBA,parsed_new,parsed.r_info.ID(1:32,3))

>> [parsed_new] = addColour(parsed_new,(parsed.r_info.ID(1:32,3)),5)

The value "5" indicates that Red pt1 Color Palette is used to highlight metabolite nodes.

>> writeCD(parsed_new,'fatty_acid_new.xml')

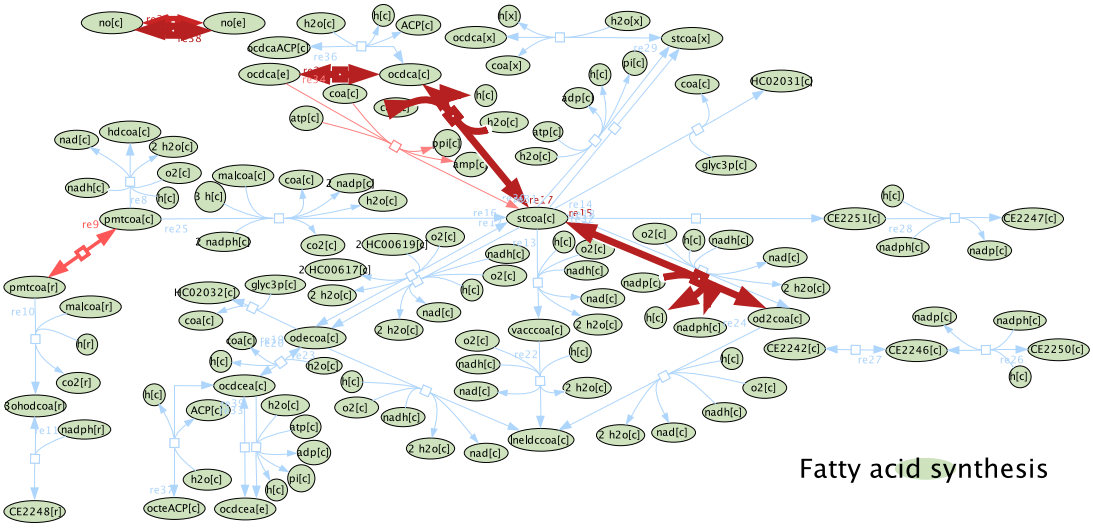


Figure 10: The colours of the metabolite nodes are modified by "colourNode" function.

[parsed_new,var,final_list]=colourNode(parsed_new,'fatty_acid_new.xml',parsed_new.r_info.ID(:,3),'false')

parsed_new.r_info.ID(:,3) contains a list of reaction IDs.

[parsed_new,var,final_list]=colourNode(parsed_new,'fatty_acid_new.xml',listRxn,'false')

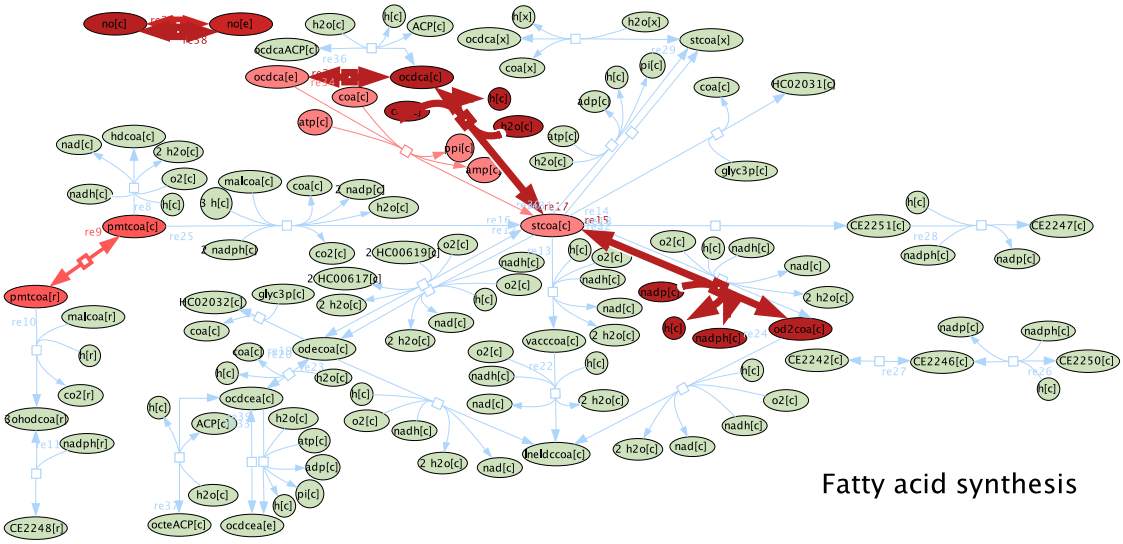


Figure 11: The colours of the nodes are changed using the colour codes outputed by "addColour" function.

>> [parsed_new,listRxn] = addColour(parsed_new,(parsed.r_info.ID(1:32,3)),5)

Another function" colourNodeWhite" can be called as a shortcut to set all other reactions except those listed in "listRxn" to white, and then set reactions listed in "listRxn" using the same colour scheme passed from "addColour" function.

[parsed_new,var,final_list]=colourNodeWhite(parsed_new,'fatty_acid_new.xml',listRxn,'false')

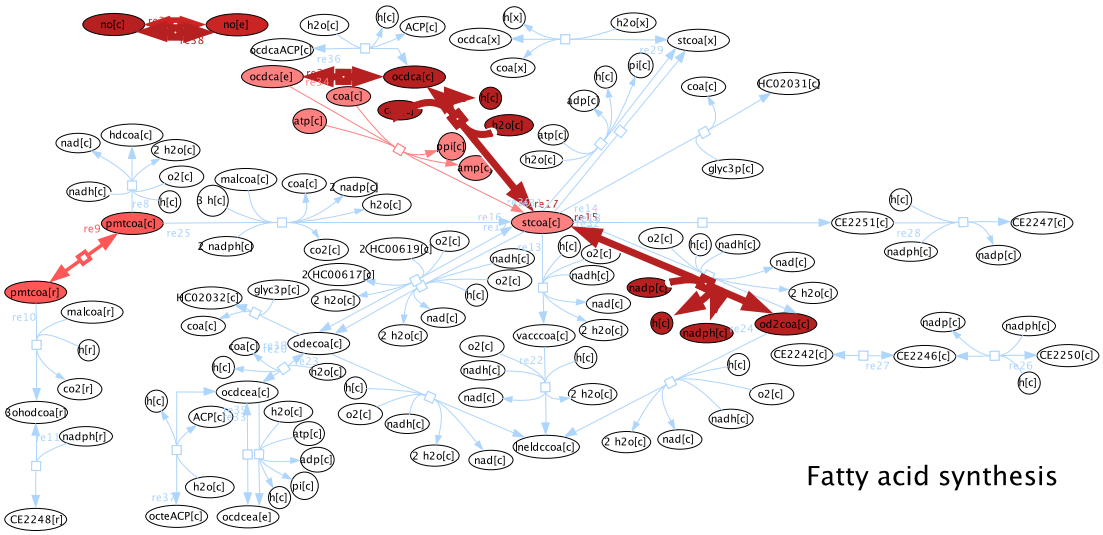


Figure 12: The colours of metabolite nodes of reactions with zero fluxes are changed to white.

4.5 Change colour of metabolite nodes of reactions

>> [parsed_new,var,final_list]=colourNode(parsed_new,'fatty_acid_new.xml',listRxn,listMet)

The additional argument " listMet" contains the metabolite IDs that are needed to be highlighted for two reactions 're5161' and 're5168' of the variable "listRxn"; in other words, 'pmtcoa[r]' in reaction 're5161', and 'nadp[c]' and 'nadph[c]' in reaction 're5168' will be highlighted in orange colour by default.

The content of the “listRxn” variable

	1	2
1	re5161	#fffe5757
2	re5168	#ffb62020
3	re5170	#ffb62020
4	re5174	#ffb62020
5	OCDCAFATPc	#fffe8181
6	CDMT02844	#ffcb2424
7	re8845	#ffb62020
8		

The content of the “listMet” variable

2x2 cell

	1	2	3
1	'pmtcoa[r]'	[]	
2	'nadp[c]'	'nadph[c]'	
3			
4			

Figure 13: The relationship between “listRxn” and “listMet” variables.

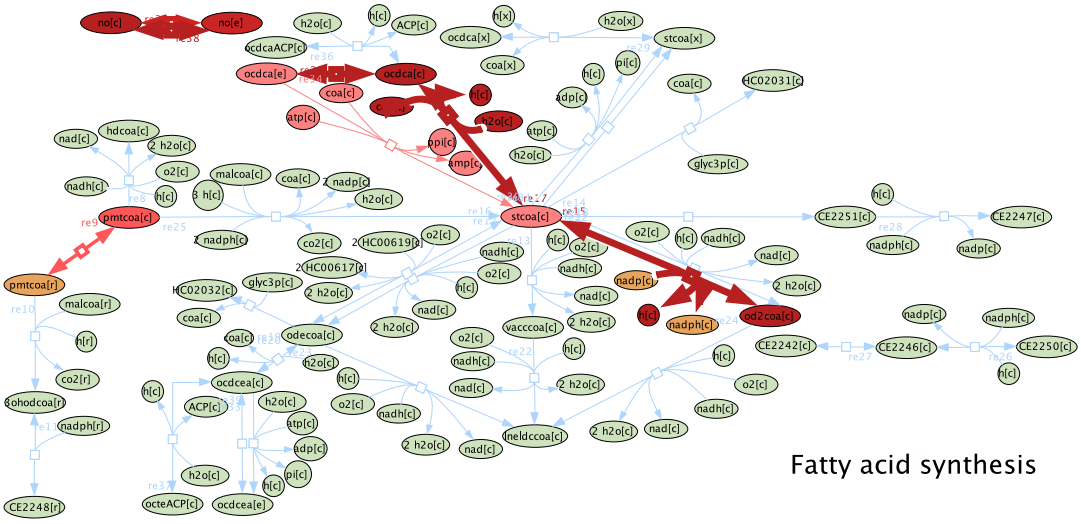


Figure 14: The colours of particular metabolite nodes are modified.

The default orange for highlighting metabolite nodes can be changed by using the fifth attribute of the function to specify a list of colour codes for the list of metabolite IDs in the “listMet” variable.

>>

[parsed_new,var,final_list]=colourNode(parsed_new,'fatty_acid_new.xml',listRxn,listMet,listMetColour)

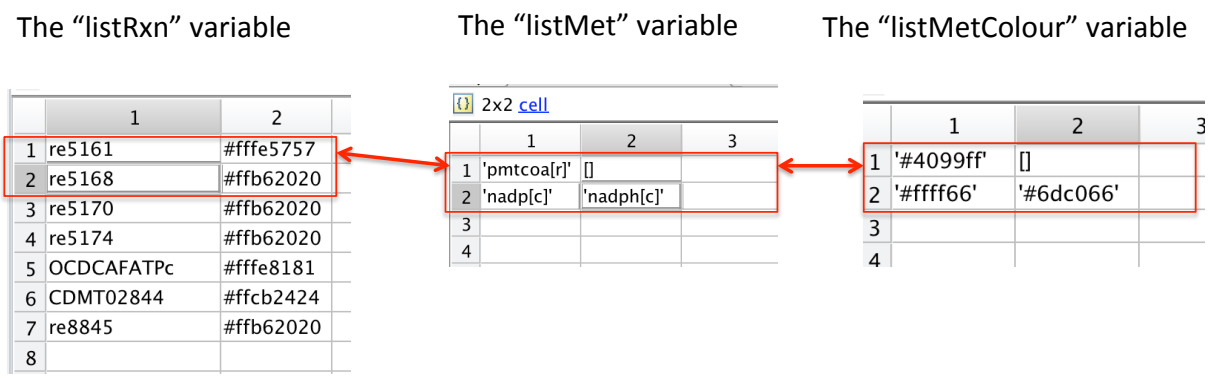


Figure 15: The relationship between “listRxn” and optional “listMet” & “listMetColour” variables.

Figure 16: The hex colour codes and their colours

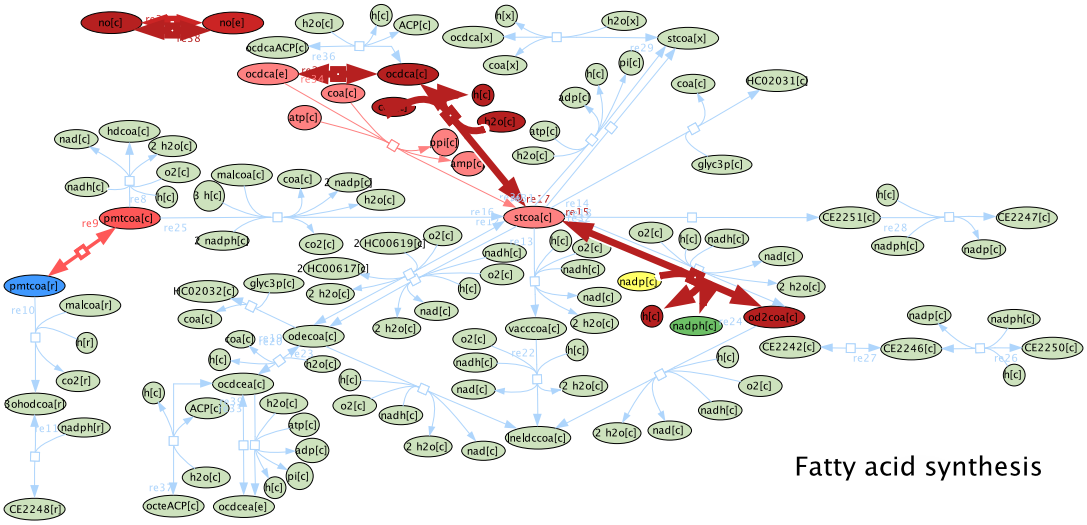


Figure 17: The colours of particular metabolite nodes are modified.

The row number in “listMet” variable corresponds to the same row number in “listRxn” variable. The “colourNode” function first check if the metabolites indicated in “listMet” appear in the corresponding reactions in “listRxn’. For example, a screenshot of the “listMet” below shows that the first 30 rows are empty and the metabolites in the first 30 reactions in “listRxn” are not particularly highlighted. Each of rows 31 & 32 contain two metabolite IDs; the function first check if reaction 31 contains metabolite “no[c]” and “o2[c]”, and reaction 32 contains the metabolites “h[c]” and “adp[c]” respectively; only those listed metabolite IDs existing in the corresponding reaction will be particularly highlighted on the graphical layout. The default colour is orange, but can be changed to any other colours by specifying the hex colour codes in “list_Colour_Met”, which is an optional argument of the “colourNode” function.

	1	2
10	[]	[]
11	[]	[]
12	[]	[]
13	[]	[]
14	[]	[]
15	[]	[]
16	[]	[]
17	[]	[]
18	[]	[]
19	[]	[]
20	[]	[]
21	[]	[]
22	[]	[]
23	[]	[]
24	[]	[]
25	[]	[]
26	[]	[]
27	[]	[]
28	[]	[]
29	[]	[]
30	[]	[]
31	'no[c]'	'o2[c]'
32	'h[c]'	'adp[c]'

Figure 18: An example of highlighting particular metabolites in reaction 31 and 32.

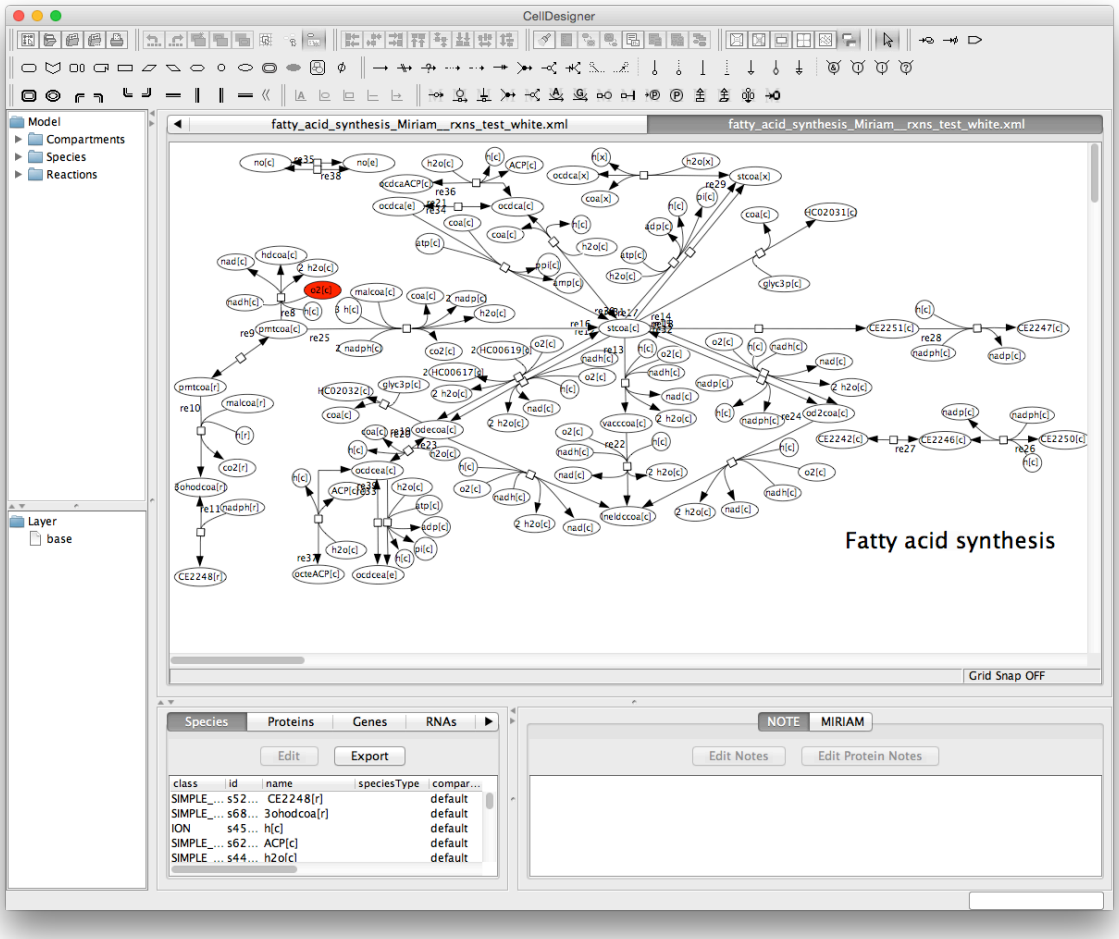


Figure 19: The result using the "list_Colour_Met" as an additional input in the "colourNode" function.

5 Error corrections

There could be errors with the species names when the network layout is drawn by human effort. The content (e.g., species names) of a CellDesigner files can be corrected using a COBRA model structure as the reference. The errors in the network layout of 'fatty acid pathway' are corrected through the following steps.

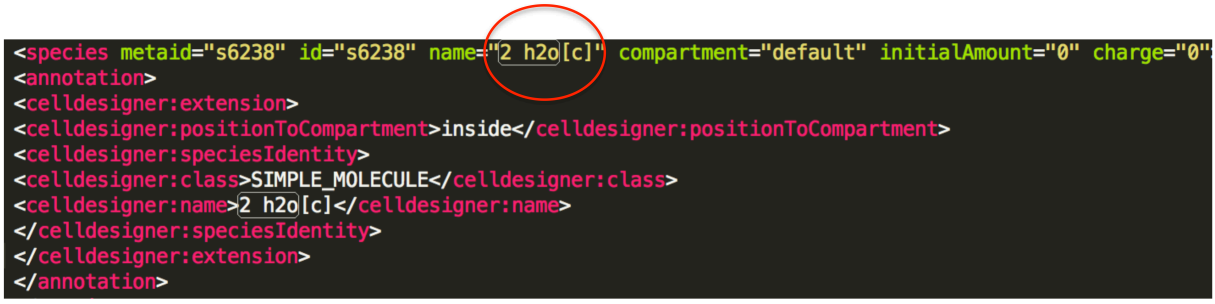


Figure 20: A screenshot of spelling errors in metabolite names.

5.1 Check and correct species names

"cmpMet" and "correctMetName" functions can be used to correct species names in a parsed CD model

```
>> results =cmpMet(parsed_new,model)
```

	1	2	3	4	5	6	7	8
1	's6277'	'3 h[c]'	[]	[]	'2447'	[]	'Not Pres...	8
2	's5086'	'2 nadph[c]'	[]	[]	'2458'	[]	'Not Pres...	9
3	's8194'	'Fatty acid s...	[]	[]	'2590'	[]	'Not Pres...	21
4	's6238'	'2 h2o[c]'	[]	[]	'2744'	[]	'Not Pres...	35
5	's5084'	'2 nadp[c]'	[]	[]	'2766'	[]	'Not Pres...	37
6	's5278'	'2 HC00617[...	[]	[]	'2821'	[]	'Not Pres...	42
7	's6830'	'2 HC00619[...	[]	[]	'2854'	[]	'Not Pres...	45
8								

Figure 21: The main field of the return variable of "cmpMet".

This command intends to produce a "r_info" structure that is updated with the identified changes to the metabolite names.

1. The first argument stores the parsed model structure. The structure is the same as "parsedModel.r_info"

2. The second argument is the return variable of the 'cmpMet' function. Basically, the second field of the variable contains a list of row numbers of the problematic species names which is used by 'correct_M' function to locate the incorrect metabolite names in the array and replace them with the provided substituting species names contained the third argument.

The following command corrects the wrong species names with reference to a list of correct species names.

```
>> parsed_corrected=correctMetName(parsed_new, results, list_new_names)
```

The following is the screenshot of the list of substitute metabolite names

7x1 cell

	1	2
1	h[c]	
2	nadph[c]	
3	Fatty aci...	
4	h2o[c]	
5	nadp[c]	
6	HC00617...	
7	HC00619...	
8		
9		

Figure 22: A list of manually created correct metabolite IDs.

Optionally, "cmpMet" function can then be called again to check if the there would be any further discrepancies existing between the COBRA and parsed CellDesigner models.

```
>> results=cmpMet(parsed_corrected,model)
```

	1	2	3	4	5	6	7	
1	's8194'	'Fatty acid synthesis'	[]	[]	'2590'	[]	'Not Prese...	21
2								
3								
4								

Figure 23: The dummy metabolite in the graphic network.

The result indicates that there is only one metabolite that cannot be found in the reference COBRA model structure. The metabolite name is "Fatty acid synthesis', which is a dummy metabolite shown on the network layout as a name label of the whole network. Therefore, it is fine to leave where it is.

5.2 Repair the XML file

The following command writes an updated parsed model structure to a new XML file.

```
>> XML = repairXML(parsed_corrected,'fatty_acid_corrected.xml')
```

This command intends produce a variable "text" that stores all the text lines of the XML file, using "annotedText" as a reference, which is outputted by "writeXML" function.

- "parsed_corrected" is the corrected parsed model structure outputted by "correctMetName" function.

- "fatty_acid_corrected.xml" is the name of the output XML file.

- "XML" contains text lines of the output XML file.

The following is the screenshot of the corrected network layout, in metabolite IDs with stoichiometric coefficients (such as “3 h[c]”) are changed to metabolite IDs (such as “h[c]”).

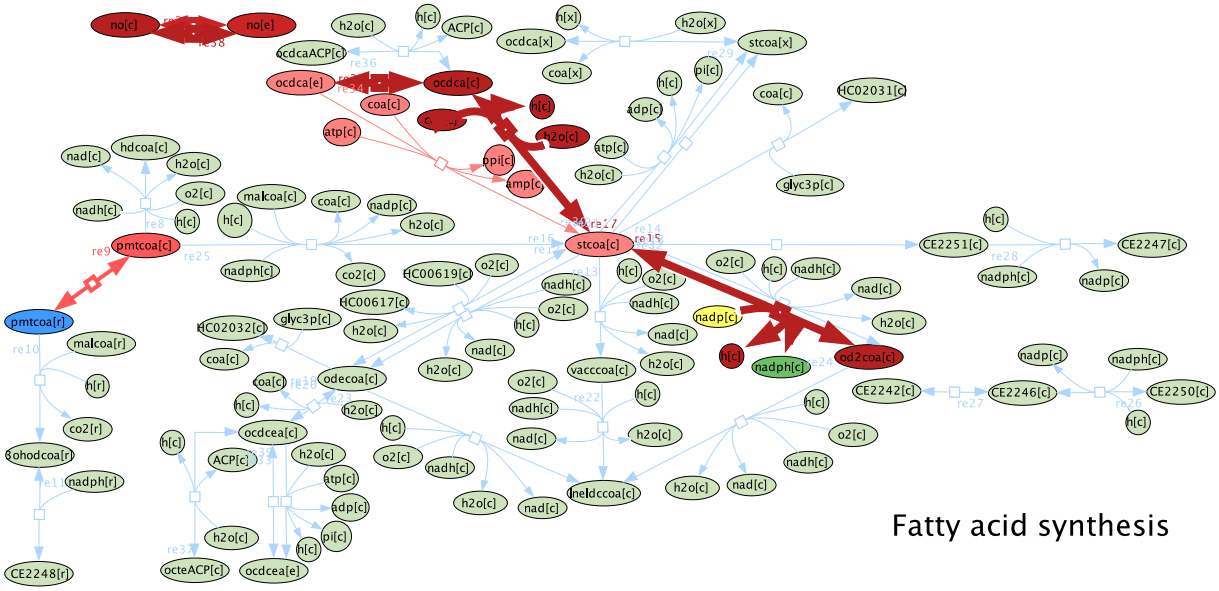


Figure 24: The graphic layout of the network with corrected metaboite IDs.

5.3 Check reaction names

“cmpRxn” function can be called to check if there are any discrepancies in reactions between the parsed CD and the COBRA models.

```
>>results =cmpRxn(parsed_corrected,model)
```

1x1 struct with 3 fields

Field	Value
main	32x5 cell
listOfFound	32x5 cell
found_rxns_and_mets	1x1 struct

Figure 25: The screenshot the variable returned by“cmpRxn” function.

The result indicates that there is no discrepancy found in reaction identifiers between the CD and COBRA model structures.

To demonstrate the discrepancies in reactions can be identified by the ”cmpRxn” function, we use the central metabolic network of Recon2 as an example, and compare it with the fatty acid Matlab model structure.

```
>> parsed_Central=parseCD('CentralMetabolism_Recon2.xml')
```

The following command uses “cmpRxn” function to compare the parsed central metabolism structure (i.e., CD model) with the fatty acid COBRA model structure (i.e., COBRA model).

```
>> results =cmpRxn(parsed_Central,model)
```

The result (see below) indicates that three reactions of the parsed CD model are found in the COBRA model, 214 reactions of the parsed CD model are not present in the COBRA model.

“main” displays all the comparison results. The 1-3 columns list the all of the available reaction IDs in the CD model; the fifth column lists the comparison results: a “found” text indicates the reaction ID of the CD model is also present in the COBRA model.

“listOfFound” contains the reactions of the CD models that are present in both CD and COBRA models.

“found_rxns_and_mets” lists all the substrates and products of the reactions present in both CD and COBRA models; the metabolite IDs are retrieved from the COBRA model.

“list_of_rxns_not_present_in_ReferenceModel’”lists the reactions of the CD model that are absent from the COBRA model.

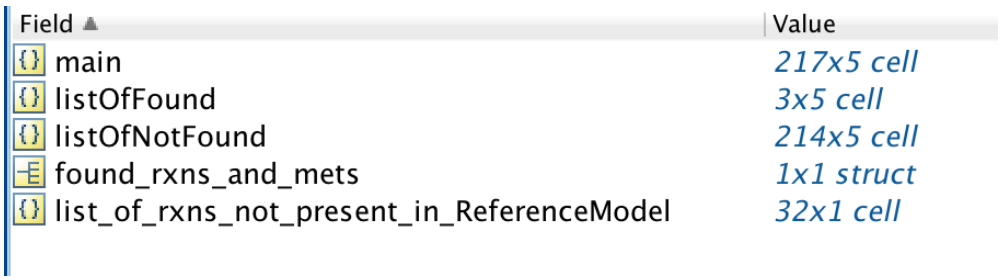


Figure 26: The screenshot the variable returned by"cmpRxn" function

6 Annotation

There are two ways to annotate a component (Species or Reaction), by adding free text or MIRIAM notes (the Minimal Information Requested In the Annotation of Models). Free text notes give you flexibilities to add user-defined types of omics data, whereas MIRIAM annotation is a standard scheme to annotate and curate computational models in biology (http://www.ebi.ac.uk/miriam/) and recommended by SBML Level 2 Version 4.

6.1 Free-text annotation

The following command retrieves relevant omics data for reactions from COBRA structure and integrates them with the CD XML file:

```
>> [var]=addAnnotation ('fatty_acid_new.xml','fatty_acid_new_annotated.xml',
parsed.r_info.ID(:,3),recon2);
```

parsed.r_info.species(:,2) is the list of species names.

The function returns an array of text lines of the XML file, as saved in "var".

The following screenshot shows the added annotations for reaction “OCDCAFATPc” when clicking on the reaction node of the graphical layout in CellDesigner.

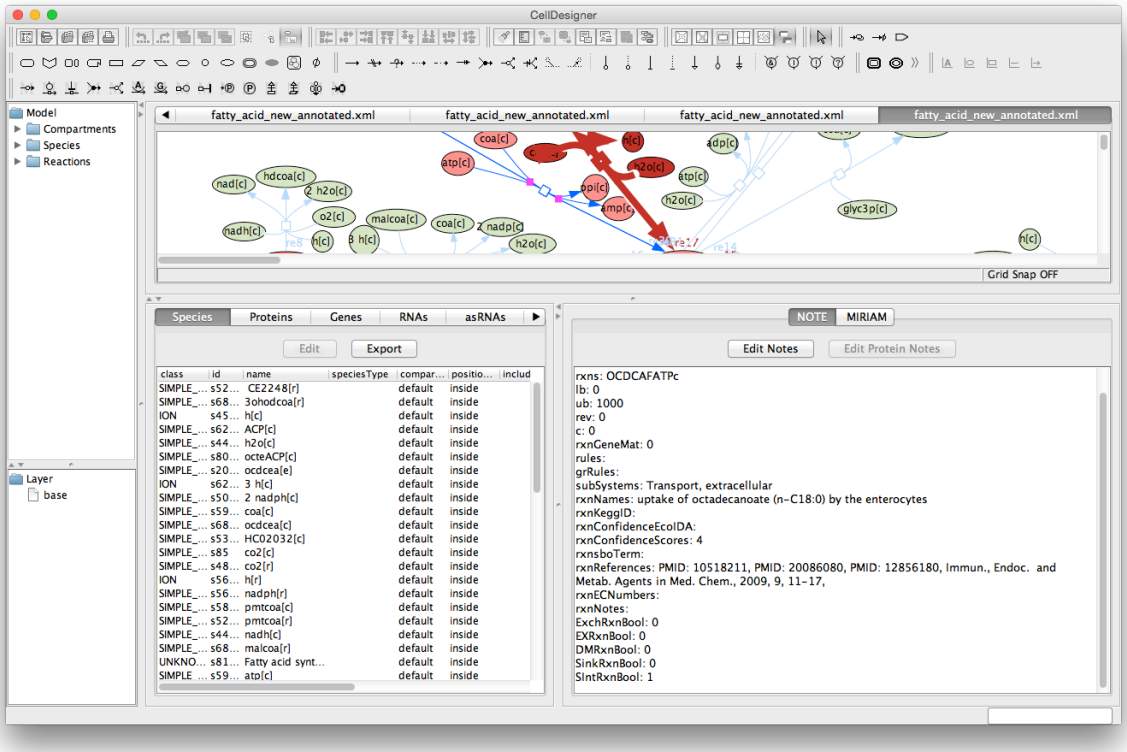


Figure 27: The screenshot of the network layout with reaction annotations

Then, the omics data for metabolite can be added using the following command:

```
>> [var]=addAnnotation ('fatty_acid_new.xml','fatty_acid_new_annotated.xml',
parsed.r_info.species(:,2),recon2);
```

The following screenshot shows the annotations for reaction “coa[c]” when clicking on the metabolite node on the graphical layout in CellDesigner.

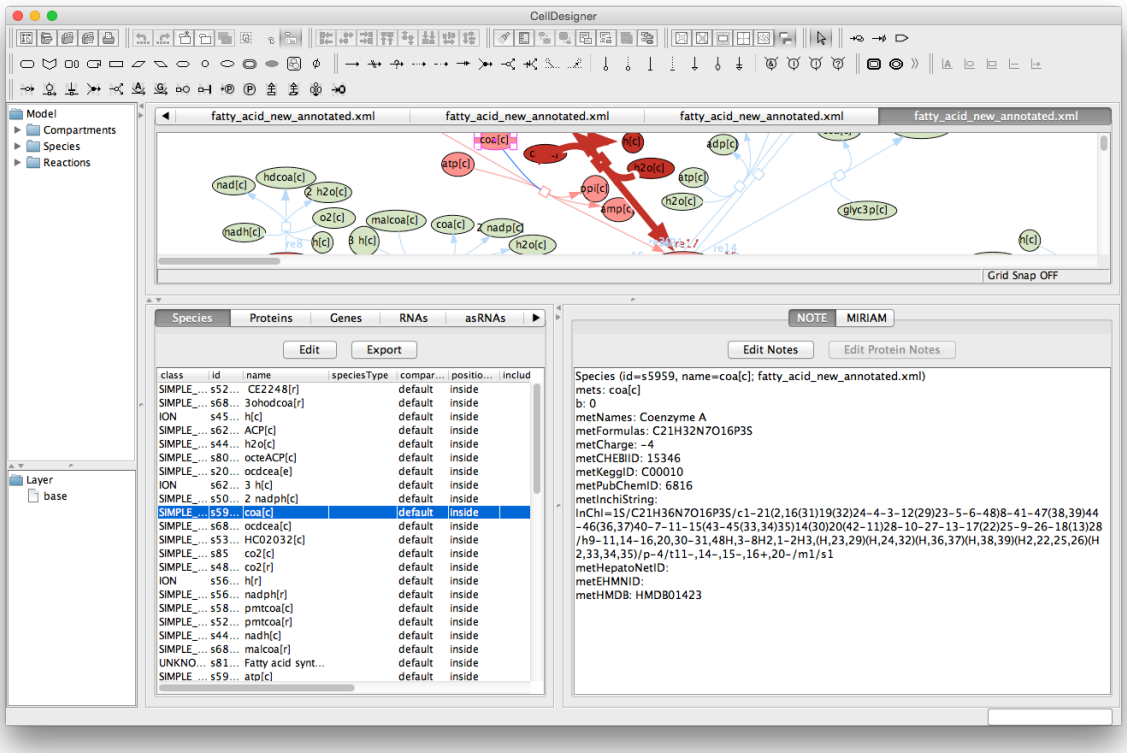


Figure 28: The screenshot of the network layout with metabolite annotations.

Instead of calling “addAnnotation” function twice, it is also possible to add omics data for reactions and metabolites using one single command:

First, run the following command to combine the lists of reactions and metabolites.

```
>> sumList={parsed.r_info.ID{:,3},parsed.r_info.species{:,2}}'
```

Second, use “sumList” as the third argument of the function, which contains a combined list of reactions and metabolites.

```
>> [var]=addAnnotation
('fatty_acid_new.xml','fatty_acid_new_annotated.xml',sumList,recon2);
```

6.2 MIRIAM information

The MIRIAM annotations for both metabolite and reactions can be added to the XML file using the following command:

```
>> addMiriam('fatty_acid_new.xml','fatty_acid_miriam.xml',sumList,model_miriam,'name',
list(1:15,:))
```

'fatty_acid_new.xml' - A CD compliable XML file without annotations

'fatty_acid_miriam.xml' – a new file that contains annotations.

sumList - a mixed list of reaction metabolite names, which is produced by merging the lists of metabolite and reactions into one list using the command:

```
>> sumList={parsed.r_info.ID{:,3},parsed.r_info.species{:,2}}'
```

model_miriam – A COBRA model structure that contains the necessary MIRIAM annotations. A snapshot of an example structure array is as follows:

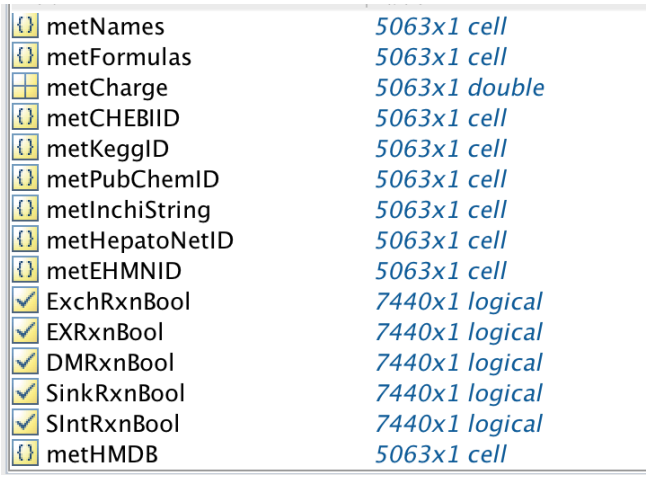


Figure 29: An example data array of different MIRIAM annoations in a COBRA model structure

List - the first column contains a list of field names of data arrays in a COBRA Matlab structure, and each data array contains one type of MIRIAM information; the second column listed the MIRIAM data types for each data array. The third column lists the relations for each type of MIRIAM data. If the third column is missing, the function will generate a list of random relations for the data arrays.

For example, ID is a list of reaction IDs, so 'model_t_list2' should contain the MIRIAM information for a reaction.

The screenshot shows the MIRIAM information is added to the XML file.

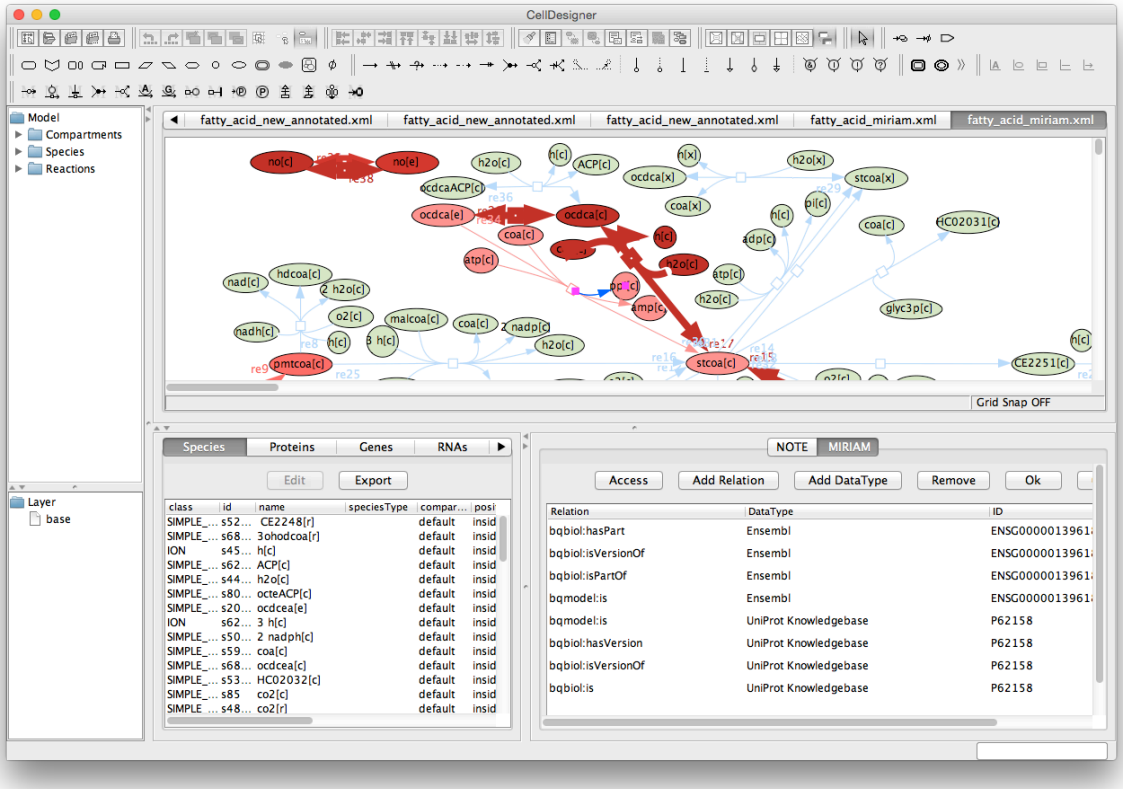


Figure 30: The screenshot of the network layout with MIRIAM annotations.

NOTE:

It should be noted that the 'XML file' is the one that is used to produce the 'parsed model structure'; otherwise there will be errors.

Potential errors may be caused by the fact that there is no corresponding fields referring to annotations in the Matlab structure, or the length of the annotation arrays are not consistent with that of the list of reactions or metabolites in the model structure.

6.3 Other annotation types

You can create any omics data using many COBRA functions. For example, it is possible to use COBRA functions to generate different types of annotations, which can then be added to the XML file as free-text annotations.

Use COBRA function “printRxnFormula” to create a new field of array of reaction formulas

```
>> model_updated.rxnformulas=printRxnFormula(model,model.rxns(:))
```

Add all of the annotations to XML file.

```
>>
addAnnotation('fatty_acid.xml','fatty_acid_annotated.xml',parsed.r_info.species(1:58,2),model_updated);
```