API documentation

‘Gil’

KoreaUSeoul-2015

Web Visualization

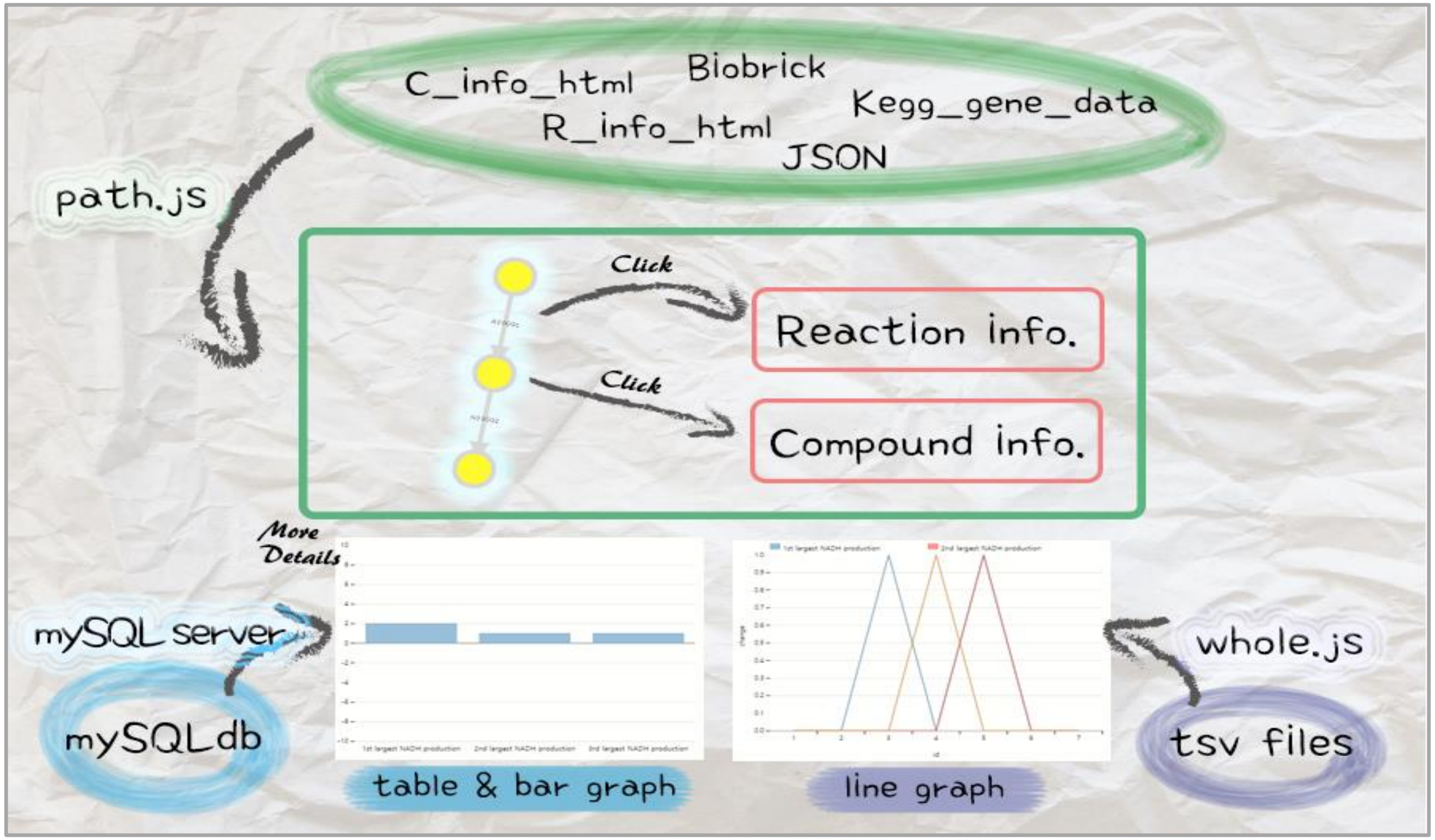

Figure 1. How to use the data in Web Visulization

1. Data (from web app background)

A. TSV files -> **line graph** in graph.php

i. File name: (source compound ID)_(target compound ID).tsv (ex : C01033_C00124.tsv)

ii. Content:

```
id (tab) showname (tab) reaction (tab) change <- format of line
n (pathway relevant to each criterion) (R-number) (quantity) <- content of line
1 1st largest ATP production R10184 0
2 1st largest ATP production R01535 0
3 1st largest ATP production R01192 0
1 1st largest NADH production R10184 0
2 1st largest NADH production R01535 1
3 1st largest NADH production R01535 1
```

id = the order of reactions in one pathway

showname = the rank explanation of the pathway according to each criterion

reaction = the KEGG reaction ID in the pathway

change = mark the change of ATP, CO2, NADH or NADPH in change row as the reactions proceed

B. mySQLdb.txt -> **bar graph & table** in graph.php

i. Content:

```
C#_C#_Criterion_# (tab) showname (tab) ATP (tab) CO2 (tab) NADH (tab) NADPH (tab) net_name (tab)
Reactions#
(Source compound ID)_(Target compound ID)_(Criterion)_(number) (ATP production) (CO2 loss) (NADH
production) (NADPH production) (Net equation in chemical names) (Reaction included in the path)
```

```
C03680_C00804_ATP_1 1st largest ATP production  1   -1  0   -1  CO2 + 4-Imidazolone-5-propanoate + H + Acetate + NADPH + Acetyl-CoA -> 4-Oxoglutaramate + H2 + Pyruvate + Propynoate + CoA + NADP + Ammonia ['R04283', 'R00522', 'R00338', 'R00344', 'R00742', 'R00740', 'R01611']
C03680_C00804_ATP_2 2nd largest ATP production  1   0   0   -1  4-Imidazolone-5-propanoate + H + (S)-Methylmalonyl-CoA + NADPH + H2O + 2 Acetyl-CoA -> 4-Oxoglutaramate + H2 + Propanoyl-CoA + Pyruvate + Propynoate + 2 CoA + NADP + Ammonia    ['R04283', 'R00212', 'R00930', 'R00344', 'R00742', 'R00740', 'R01611']
```

ii. How to upload to MySQL

1. MySQL server must be available in your computer. (link)
2. If so, upload with the following commands.

```
CREATE TABLE IF NOT EXISTS path_score_table1 (
            id int auto_increment,
            path VARCHAR(255),
            showname VARCHAR(255),
            atp VARCHAR(255),
            co2 VARCHAR(255),
            nadh VARCHAR(255),
            nadph VARCHAR(255),
            net_name VARCHAR(255),
            reaction VARCHAR(255),
            PRIMARY KEY (id)
      )
```

```
load data local infile 'your path/mySQLdb.txt' REPLACE INTO TABLE path_score_table IGNORE 1 LINES (path, showname, atp, co2, nadh, nadph, net_name, reaction);
```

```
alter table path_score_table add index index1(path);
```

C. JSON files -> **visualized** in path.js

i. File name : (source compound ID)_(target compound ID).json (ex : C01033_C00124.json)

ii. Content (Orange : format, **black** : example):

```
{
    "nodes":
        [
      {
      "path_selected_by_criterion": 0 or 1,
      "name": "CXXXXX",
      "chemical_name":"compound",
       }


      {
          "NADH_3": 1,
          "name": "C00022",
          "CO2_1": 1,
          "CO2_2": 1,
          "CO2_3": 1,
          "chemical_name": "Pyruvate",
          "ATP_1": 1,
          "NADPH_2": 1,
          "ATP_3": 1,
          "ATP_2": 1,
          "NADPH_3": 1,
          "NADH_1": 1,
          "NADPH_1": 1,
                "NADH_2": 1,
            },

        },{.......}]
"links": [
            {
                "reaction": "RXXXXX",
                "source": 0(number),
                "target": 6(number),
                "path_selected_by_criterion": 0 or 1,
                "ecoli": "0" or “1”,
            },

            {
                "reaction": "R00212",
                "target": 6,
                "NADH_2": 0,
                "CO2_1": 0,
                "CO2_2": 1,
                "CO2_3": 0,
                "ecoli": "1",
                "source": 0,
                "NADPH_1": 0,
                "ATP_2": 1,
                "NADPH_3": 1,
                "NADPH_2": 0,
                "ATP_3": 0,
                "NADH_2": 0,
                "NADH_3": 0,
            },

        },{.....}]
}
```

iii. Detailed explanation above content:

1. ‘**nodes**’ contain the information of compounds.
2. ‘**criteria**’ expresses with 0 or 1 whether one pathway satisfies the corresponding criteria. (0 : not satisfied , 1: satisfied)

3. **'name'**: changes the data input by the user into C number.
4. **'chemical name'**: chemical name of the compound shown to user when mouse over.
5. **'links'** connect one compound to one another through path.js with the information of the reaction equation.
6. **'reaction'** : KEGG reaction ID of the equation.
7. **'source', 'target'** : numbered by the order of the nodes in the json file.
8. **'ecoli'** : expressed in 0 or 1 whether the reaction is included in *E. coli K12* metabolism. (0 : not included, 1 : included)

2. php

A. graph.php

i. -> to path.js & phpself

1. Content:

```
$input = explode(';',$_GET["start_compound"]);
       $output = explode(';',$_GET["end_compound"]);


<script>
                var input = "<?php echo $input[1]; ?>";
                var output = "<?php echo $output[1]; ?>";
</script>
```

2. Detailed explanation : receives data input by the user (receives the C##### that comes right after the semicolon(;))

ii. -> use mySQLdb.txt for bar graph & table

1. Content

2. Detailed explanation : create a table in the web using the table of MySQL according to the data entered by the user

iii. -> use tsv files for line graph

1. Content:

```
<script type="text/javascript">
 var pathname = input + '_' + output
 var svg0 = dimple.newSvg("#chartContainer", 590, 400);
   d3.tsv("data/" + pathname + ".tsv", function (data0) {
       data0 = dimple.filterData(data0, "showname", [
           '1st largest ATP production ',    ' 2nd largest ATP production ', '3rd largest ATP production'
       ])
   var myChart0 = new dimple.chart(svg0, data0);
   myChart0.setBounds(60, 30, 505, 305);
   var x = myChart0.addCategoryAxis("x", "id");
   x.addOrderRule("id");
   myChart0.addMeasureAxis("y", "change");
   myChart0.addSeries("showname", dimple.plot.line);
   myChart0.addLegend(60, 10, 500, 20, "right");
   myChart0.draw();
   });
</script>
```

2. Detailed explanation:

✓ draw a line graph using the tsv in div (chartContainer) according to the value that the user entered. (Depending on the 'show name')

✓ X axis shows the ID of the tsv, and Y axis shows the value of 'change' column.

3. Javascript

A. path.js *(The contents below explain the codes of path.js)*

i. Set height and width according the the size of the div.

```
var force = d3.layout.force()
        .gravity(0.07).
        .charge(-2000).
        .linkDistance(0)
        .friction(0.95)
        .size([w,h]);
```

ii. Adjust the property of the node and edge.

```
var w = $( 'div.a' ).width();
var h = $( 'div.a' ).height()*0.1;

var focus_node = null
var text_center = false;
var size = d3.scale.pow().exponent(1)
    .domain([1,100])
    .range([1,100]);
```

iii. Set the range of the node color to assign colors to each degree level according to the degree information in json

```
var color = d3.scale.linear()
        .domain([0, 1, 2, 3])
        .range(["#fffa30", "#fffbb6", "#ffffec", "#fff88e"]);
```

iv. Set the size and stroke range of the node for the zoom in and out function.

```
var nominal_base_node_size = 15;
var nominal_text_size = 6;
var max_text_size = 24;
var nominal_stroke =4;
var max_stroke = 15;
var max_base_node_size = 80;
var min_zoom = 0.5;
var max_zoom = 100;
var svg = d3.select(".a").append("svg");
var zoom = d3.behavior.zoom().scaleExtent([min_zoom,max_zoom])
var g = svg.append("g");
```

v. Load json by input and output name. e.g. C0001_C0002.json

```
var json_file = 'json/' + input + '_' + output + '.json';
```

vi. Draw a graph according to the 'nodes' and 'links' part of json.

```
var link = g.selectAll(".link")
          .data(json.links)
          .enter().append("line")
          .attr("class", "link")
          .attr("marker-end", "url(#end)")
          .style("stroke-width",nominal_stroke);
```

vii. Show the compound name when mouse over.

```
var tip = d3.tip()
      .attr('class', 'd3-tip')
      .offset([-10, 0])
      .html(function (d) {
      return    d.chemical_name + "";
})
svg.call(tip);
```

viii. Assign equation number to the edge.

```
var labels = g.selectAll("text")
      .data(json.links)
   .enter().append("text")
```

```
        .attr("class", "linktext")
        .attr("x", function(d) { return (d.source.y + d.target.y) / 2; })
        .attr("y", function(d) { return (d.source.x + d.target.x) / 2; })
        .attr("text-anchor", "middle")
        .style("font-size", nominal_text_size + "px")
        .text(function(d) { return d.reaction });
```

ix. Assign color to each degree level and shows tool tips when mouse over.

```
var node = g.selectAll(".node")
          .data(json.nodes)
          .enter().append("g")
          .attr("class", "node")
          .call(force.drag)
          .style("fill", function(d) { return color(d.degree);})
          .on('mouseover', tip.show)
          .on('mouseout', tip.hide);
```

x. Zoom in and out function available.

```
zoom.on("zoom", function() {

  var stroke = nominal_stroke;
    if (nominal_stroke*zoom.scale()>max_stroke) stroke = max_stroke/zoom.scale();
    link.style("stroke-width",stroke);
    circle.style("stroke-width",stroke);

  var base_radius = nominal_base_node_size;
    if (nominal_base_node_size*zoom.scale()>max_base_node_size) base_radius = max_base_node_size/zoom.scale();
        circle.attr("d", d3.svg.symbol()
        .size(function(d) { return Math.PI*Math.pow(size(d.size)*base_radius/nominal_base_node_size||base_radius,2); })
        )
          g.attr("transform", "translate(" + d3.event.translate + ")scale(" + d3.event.scale + ")");
          });

  svg.call(zoom);

  resize();
  window.focus();
```

xi. Draw nodes and edges according to the information of json. (The networks is outstretched according to the e.alpha value.)

```
function tick(e) {

      // Push sources up and targets down to form a weak tree.
      var k = 30 * e.alpha;
      json.links.forEach(function(d, i) {
         d.source.y -= k;
         d.target.y += k;
      });

      node.attr("transform", function(d) { return "translate(" + d.x + "," + d.y + ")"; });
// postion of edge lable (R Number)
labels.attr("x", function(d) { return (d.source.x + d.target.x) / 2; })
            .attr("y", function(d) { return (d.source.y + d.target.y) / 2; });

// position of edge.
labels.attr("x", function(d) { return (d.source.x + d.target.x) / 2; })
            .attr("y", function(d) { return (d.source.y + d.target.y) / 2; });
      link.attr("x1", function(d) { return d.source.x; })
         .attr("y1", function(d) { return d.source.y; })
         .attr("x2", function(d) { return d.target.x; })
         .attr("y2", function(d) { return d.target.y; });
// position of node
node.attr("cx", function(d) { return d.x; })
         .attr("cy", function(d) { return d.y; });


   };
```

xii. Insert arrows according to the json information. (source->target).

```
svg.append("svg:defs").selectAll("marker")
      .data(["end"])
   .enter().append("svg:marker")
      .attr("id", String)
      .attr("viewBox", "0 -5 10 10")
      .attr("refX", 15)
      .attr("refY", 0)
      .attr("markerWidth", 4)
      .attr("markerHeight", 4)
      .attr("orient", "auto")
      .append("svg:path")
      .attr("d", "M0,-5L10,0L0,5")
      .style("fill", "#D8D8D8")
      .style("stroke", "#FFFFFF")
```

```
        .style("stroke-opacity", "0.1")
        .style("stroke-width", "1.3px");
```

xiii. Show the information of the compound or reaction when clicking the nodes or edges and receive the html (C_info_html / R_info_html) that holds the data of the clicked nodes or edges according to the Jason information.

```
function showinfo(d) {
    var header1 = document.getElementById('info');
    info.innerHTML = '<iframe src="C_info_html/' + d.name + '.html" width=100% height=300px marginwidth="0" marginheight="0" scrolling="overflow-
x:hidden" frameborder="0" scrolling="no" ></iframe>' ;
};

function showlinkinfo(d) {
    var header2 = document.getElementById('linkinfo');
    linkinfo.innerHTML = '<embed src="R_info_html/' + d.reaction + '.html" width=100% height=280px marginwidth="0" marginheight="0"
scrolling="overflow-x:hidden" frameborder="0" scrolling="no" ></embed>' ;
};
```

Web app background data formation

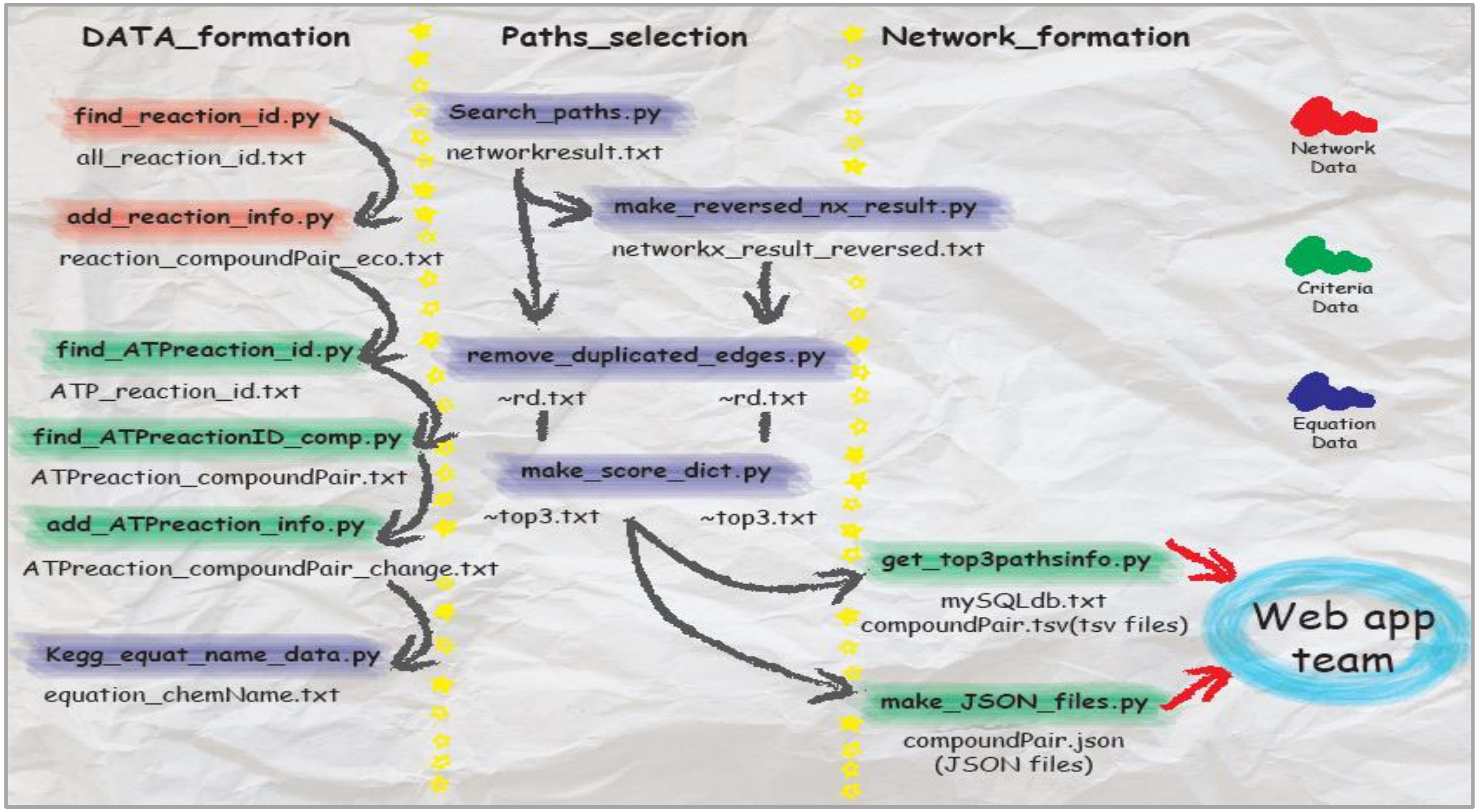

Figure 2. How to make web app background data for network

1. KEGG rest api

A. Description (in KEGG API)

: **KEGG API** (application programming interface) allows customization of KEGG-based analysis, such as for searching and computing biochemical pathways in cellular processes or analyzing the universe of genes in the completely sequenced genomes.

(If you want to more detailed KEGG api information, enter http://www.kegg.jp/kegg/rest/)

B. General URL form

```
http://rest.kegg.jp/<operation>/<argument>[/<argument2>][/<option>]

<operation> = info | list | find | get | conv | link
<argument> = <database> | <dbentries>
```

C. URL that we use

i. URL to get all KEGG reactions

```
http://rest.kegg.jp/list/reaction
```

ii. URL to get all KEGG reactions including C#####

```
http://rest.kegg.jp/list/reaction/<C#####>

<C#####> = C00002 (ATP), C00005 (ADP), C00011 (CO2), C00004 (NADH), C00005 (NADPH)
```

iii. URL to get all KEGG reactions including genes of *Escherichia coli K-12 MG 1655*

```
http://rest.kegg.jp/list/reaction/eco

eco = Escherichia coli K-12 MG 1655
```

***Note:** If you want to know more about Escherichia coli K-12 MG 1655, enter http://www.genome.jp/dbget-bin/www_bget?genome:T00007

iv. URL to get information of KEGG reaction

```
http://rest.kegg.jp/get/reaction/<R#####>

<R#####> = KEGG reaction ID ex)R00002
```

D. More explanation

: refer to KEGG rest api (http://www.kegg.jp/kegg/docs/keggapi.html)

2. Main python modules

A. Networkx

i. **all_simple_paths** in Simple Paths of Networkx Algorithms

```
all_simple_paths(G,source,target,cutoff=None)
```

# function: Generate all simple paths in the graph G from source to target.

# parameters

- ✓ **G** (NetworkX graph) –
- ✓ **source** (node) – Starting node for path
- ✓ **target** (node) – Ending node for path
- ✓ **cutoff** (integer, optional) – Depth to stop the search. Only paths of length <= cutoff are returned.

More explanation
:http://networkx.readthedocs.org/en/stable/reference/generated/networkx.algorithms.simple_paths.all_simple_paths.html?highlight=simple#networkx.algorithms.simple_paths.all_simple_paths

B. JSON

i. json.dumps in 18.2.1 Basic Usage

```
json.dumps(obj,   skipkeys=False,   ensure_ascii=True,   check_circular=True,
allow_nan=True,  cls=None,  indent=None,  separators=None,  encoding="utf-8",
default=None, sort_keys=False, **kw)
```

# function: Serialize `obj` to a JSON formatted **str** using this conversion table. If *ensure_ascii* is False, the result may contain non-ASCII characters and the return value may be a unicode instance.

More explanation : https://docs.python.org/2/library/json.html

3. Python files that produce Web app 'Gil' DATA

A. DATA formation

i. KeggCompoundName.txt : the text file of KEGG compound IDs, which are changed into chemical names

ii. Network data

1. find_reaction_id.py

   # function: save the entire reaction lists in KEGG to the file.

   # output: 'all_reaction_id.txt' (Format: R#####)

2. add_reaction_id.py

   # function: bring main C#, C# of each reaction using KEGG rest api and reaction list and mark whether *E. coli K12* contains the gene of the reaction.

   # python usage:

   ```
   python add_reaction_info.py -i <KEGG_reaction_id> -r <root_dir>
   ```

   -i : The input text file of all KEGG reaction id list.
   (It must be the output file of *find_reaction_id.py*)
   -r : The root directory. All files will be generated here.

   # output: '<root_dir>/reaction_compoundPair_eco.txt' (Format: R#####, C#####, C#####, 0 or 1)

   0 means that gene of this reaction (R#####) is not included in *E. coli* genome.

   1 means that gene of this reaction (R#####) is included in *E. coli* genome.

iii. Criteria data

The number of total python files in criteria data is 12. (ATP -> CO_2, NADH, NADPH)

1. find_ATPreaction_id.py

   # function: save the KEGG reaction list related to ATP.

   # output: 'ATP_reaction_id.txt'

2. find_ATPreaction_comp.py

   # function: save main C#, C# of each reaction that uses ATP.

   # python usage:

```
python find_ATPreaction_comp.py -i <ATPreaction_id> -r <root_dir>
```

   -i : The input text file of reaction id list, which are related to ATP

   (It must be the output file of *find_ATPreaction_id.py*)

   -r : The root directory. All files are generated here.

   # output: '<root_dir>/ATPreaction_compoundPair.txt' (Format: R#####, C#####, C#####)

   ***Note**: the reaction ids of this output file are related to ATP.

3. add_ ATPreaction_info.py

   # function: save each reactions adding variations of ATP.

   # python usage:

```
python    add_ATPreaction_info.py    -i    <ATPreaction_compoundPair>    -r
<root_dir>
```

```
-i : The input text file of reaction ids and compound pairs, which are related to ATP
     (It must be the output file of find_ATPreaction_comp.py)
-r : The root directory. All files are generated here.
```
# output: '<root_dir>/ATPreaction_compoundPair_change.txt' (Format: : R#####, C#####, C#####, #(change))

*Note: #(change) means ATP value changes in this reaction (R#####) //. + : production, - : consumption

iv. Equation data

1. KEGG_equat_name_data.py

# function: formation of database classifying the reactant and the product that is expressed with chemical name of the main reaction.

# python usage:

```
python kegg_equat_name_data.py -k <KEGG_reaction_list> -r <root_dir>

-k : The input text file including KEGG reaction IDs, compound ID pairs, and E. coli K-12 gene
information (It must be the output file of add_reaction_info.py)
-r : The root directory. All files are generated here.
```
# output: '<root_dir>/equation_chemName.txt' (format: R#####, C#####, C#####, reactants, products)

v. Gene data *(It will be used in Web visualization not Network formation)*

1. KEGG data

A. KEGG Gene data = KEGG GENES database (http://www.genome.jp/dbget-bin/www_bget?XXXXXXXX) ex. syn:ssr3451

B. KEGG_reaction_ec.tsv : the tsv file to provide the enzyme list corresponding to each reaction

C. download_KEGGenzyme.py

# function: download the names of enzymes and gene information (nucleotide sequence) in new directory.

# python usage:

```
python download_KEGGenzyme.py -r <root_dir>
```

-r : The root directory. The folder named 'Enzyme' included in all files is generated here.

#output : the folder 'Enzyme'

D. get_geneInfo.py

# function arrange the names of enzymes and gene information according to KEGG reaction ID:

# python usage:

```
python get_geneInfo.py -i <kegg_reaction> -r <root_dir>
```

-i : The input file of "kegg_reaction_ec.tsv"

-r : The root directory. The folder named 'GeneGene' included in all files will be generated here

# output: the folder 'GeneGene'

2. Biobrick data

A. process_blastn_output.py

# function: get proper Biobrick IDs corresponding to reaction gene data, and write hitml files which have links to each corresponding BioBrick registry pages

# python usage

```
python  process_blastn_output.py  -i  <reaction_id_file>  -b  <blast_output_file>  -r
```

```
<root_dir>

-i : The text file of KEGG Reaction IDs and Compound IDs of main pairs.
-b : The blast output file
-r : The root directory. All files will be generated here.
```

# output: 1) R#####.html, 2)kegg_gene_seq.blast.filtered , 3)<genecode>.html

vi. table data *(It will be used in Web visualization not Network formation)*

1. compound table.
   A. kegg_compound_part.tsv : provide the all information of each KEGG compound ID.

      (format : C##### chemical_name chemical_formula exact_weight exact_mol_weight)

   B. table_format.txt : provide the format of web app 'Gil' compound table.

   C. compound_table.py

      # function: formation of HTML table that contains compound name, C number, formula, exact weight and mol weight with kegg database.

      # python usage

```
python compound_table.py -i <input_paths> -f <table_format> -r <root_dir>

-i : The input file is 'kegg_compound_part.tsv'
-f : The format file is 'table_format.txt'
-r : The root directory. All files will be generated here
* if you want to change components of table, adjust compound_table.py and kegg_compounds.tsv.
```

#output XXXXXX.html (names after kegg_compounds.tsv contents)

2. reaction table
    A. kegg_reaction.tsv : provide the all information of each KEGG reaction ID
    B. Info_deltaG.csv : provide the △G information of each reaction
    C. table_format.txt : provide the format of web app 'Gil' reaction table.
    D. reaction_table.py

    # function: formation of HTML table that contains reaction name, equation, C_equation, delta g, and gene download button with kegg and Biobrick database.

    # requirements: kegg_reactions.tsv, table_format, reactionMain3.tsv

    # python usage

```
reaction_table.py -i <input_paths> -g <deltaG_path> -f <table_format> -r
<root_dir>


-i : The input file is the kegg_reaction.tsv
-g : The input file is Info_deltaG.csv
-f : The format file is 'table_format.txt'
-r : The root directory. All files will be generated here


* if you want to change components of table, adjust reaction_table.py and kegg_reaction.tsv.
```

    #output XXXXXX.html (names after kegg_reactions.tsv contents)

B. Paths selection

i. search_paths.py

# function: save all the paths of the compound pair that do not exceed the cutoff

# python usage:

```
python search_path.py -i <reaction_id_file> -c <networkX_cutoff> -r
<root_dir>
```

-i : The text file of KEGG Reaction IDs and Compound IDs of main pairs.

-c : The value for networkX cutoff. The default value is 7.

-r : The root directory. All files are generated here.

# output: '<root_dir>/networkx_result_<networkx_cutoff>.txt' (format : C#####_C#####   <all paths>)

*Note: <all paths> = C#####,C#####,C#####@C#####,C#####,C#####,C#####@C#####,C#####

@ means division of paths.

ii. make_reversed_nx_results.py

# function: produce the file without paths that make errors due to their overlapping R#####

# python usage:

```
python make_reverse_nx_result.py -i <input_path> -o <output_name> -r
<root_dir>
```

-i : The input text file of paths of all compound pairs

-o : The output file name that you want

-r : The root directory. All files are generated here.

# output: The output file name that you want (we recommend the input name to be added '_reversed')

(format : C#####_C#####   <all paths>)

iii. remove_dumplicated_edges.py

# function: produce the file that overturns the order of the compound pair and the order of the path.

# python usage:

```
python remove_duplicated_edges.py -i <input_path> -k <KEGG_reaction_list>
-o <ouput_name> -r <root_dir>
```

-i : The input text file of paths of all compound pairs

-k : The input text file including KEGG reaction IDs, compound ID pairs, and *E. coli K-12* gene information

-o : The output file name that you want

-r : The root directory. All files are generated here.

# output: The output file name that you want (we recommend the input name to be added '_rd')

(format : C#####_C#####   <all paths>)

iv. make_score_dict.py

# function: produce the file that arranges within 3 paths that satisfy each criteria(ATP, CO_2, NADH, NADPH)

# python usage:

```
python  make_score_dict.py  -i  <input_path>  -k  <KEGG_reaction_list>  -a
<KEGG_ATP_reaction_info>       -c       <KEGG_CO2_reaction_info>       -d
<KEGG_NADH_reaction_info> -p <KEGG_NADPH_reaction_info> -o <ouput_name> -
r <root_dir>
```

-i : The input text file of paths of all compound pairs

-k : The input text file including KEGG reaction IDs, compound ID pairs, and E. coli K-12 gene information

-a : The input text file including KEGG ATP reaction IDs, compound ID pairs, and ATP change information

-c : The input text file including KEGG CO2 reaction IDs, compound ID pairs, and CO2 change information

-d : The input text file including KEGG NADH reaction IDs, compound ID pairs, and NADH change information

-p : The input text file including KEGG NADPH reaction IDs, compound ID pairs, and NADPH change information

-o : The output file name that you want

-r : The root directory. All files are generated here.

# output: The output file name that you want (we recommend the input name to be added '_top3')

(format : C#####_C##### ATP(CO2,NADH,NADPH) <top3 paths>)

C. Networkx formation

i. get_top3pathsinfo.py

# function: produce tsv files corresponding to each compound pair and mySQLdb.txt after adding ATP, CO_2, NADH, NADPH, equation data about selected top3 paths.

# python usage:

```
python get_top3pathsInfo.py -i <input_path> -k <KEGG_reaction_list> -a
<KEGG_ATP_reaction_info>      -c      <KEGG_CO2_reaction_info>      -d
<KEGG_NADH_reaction_info>     -p     <KEGG_NADPH_reaction_info>     -e
<KEGG_equation_info> -o <output_name> -r <root_dir>
```

-i : The input text file of top3 paths of all compound pairs

-k : The input text file including KEGG reaction IDs, compound ID pairs, and *E. coli K-12* gene information

-a : The input text file including KEGG ATP reaction IDs, compound ID pairs, and ATP change information

-c : The input text file including KEGG CO2 reaction IDs, compound ID pairs, and CO2 change information

-d : The input text file including KEGG NADH reaction IDs, compound ID pairs, and NADH change information

-p : The input text file including KEGG NADPH reaction IDs, compound ID pairs, and NADPH change information

-e : The input text file including KEGG reaction IDs, compound ID pairs, and equation information

-o : The output file name that you want

-r : The root directory. All files are generated here.

# output: 1) tsv files (C#####_C#####.tsv) 2) mySQLdb.txt (that will be created in <root_dir>)

ii. make_JSON_files.py

# function: produce the selected paths of compound pair into JSON files.

# python usage:

```
python  make_JSON_files  -i  <input_paths>  -k  <KEGG_reaction_list>  -c
<compound_name> -r <root_dir>
```

-i : The input text file of top3 paths of all compound pairs

-k : The input text file including KEGG reaction IDs, compound ID pairs, and *E. coli K-12* gene information

-c : The input text file of KEGG compound IDs, which are changed into chemical names

-r : The root directory. All files are generated here.

# output: json files that will be created in <root_dir>   (C#####_C#####.json)