Hole Finding Protocol [Detailed descriptions of the steps involved]

The identification of missing enzymes in a metabolic pathway require information about the various reactions of that pathway. This information about the biological reactions are useful in the accurate reconstruction of metabolic pathways having missing enzymes. The details about these reactions are usually stored in some kind of reaction database which constitute a comprehensive resource of all the known reactions. We have used the the upgraded Ma-Zeng bioreaction database (provided herewith) contains 6851 reactions catalyzed by a set of 3525 enzymes and uses KEGG Ligand and Brenda for compiling the information.

Dependcies Required: Graph CPAN package (we have used Graph-0.94) and Storable CPAN package.

1. Parsing of Bioreaction database

The database is parsed using the perl script developed in-house 1_r_db_parse.pl .

```
$ perl 1_r_db_parse.pl <bioDB_updated.csv>
```

where, bioDB_updated.csv is the upgraded Ma-Zeng database in ‘csv’ format.

This script generates a file ‘connections’ which is a three column file having reactants, products and enzymes. The ‘connections’ file is then further parsed to create a ‘edge file’ that can be used to create a metabolic graph using the script 2_make_edge_file.pl

2. Generate metabolic connections

```
$ perl 2_make_edge_file.ple <connections>
```

The output of this script is a file named ‘edge_file’ which have the connection information for each reactant and product with its enzyme. Any duplicated connections are removed from the ‘edge_file’ using the sort command. The output file 'edge_file.unique' have unique connections .

```
$ sort -u edge_file >edge_file.unique
```

4. Create Metabolic Graph

A metabolic graph can be defined to have the reactant and products as nodes, and the enzyme catalyzing that particular reaction as an edge. This graph is created by using the perl script 3_make_metabolic_graph.pl

```
$ perl 3_make_metabolic_graph.pl <edge_file.unique>
```

The output of this script is a metabolic graph stored as ‘metabolic_graph.out’ using the perl module ‘Storable’ in a binary format. This graph is further used to create an enzyme-enzyme dependency graph.

5. Create Enzyme-Enzyme Dependency Graph

A Enzyme-Enzyme dependency graph (EE graph) is created by parsing the metabolic graph in a way that two enzymes E1 and E2 are connected if the product of the reaction catalyzed by E1 is the substrate for the reaction catalyzed by E2. This graph represents a superset of connections for enzymes from all the known reactions. It is created using the perl script '4_make_enzyme_dependency_graph.pl' which reads the metabolic graph.

```
$ perl 4_make_enzyme_dependency_grpah.pl
```

The output of this script is an EE graph ‘EE_graph.out’ which is also stored in a binary format using the perl module ‘Storable’.

6. Retrieval of enzyme list using ModEnzA

The enzyme list of any organism under study is retrieved using the in-house developed method ModEnzA. This was done using ‘hmmsearch’ where, query hmmfile is the database of all enzyme profiles concatenated into one file and all the sequences from a species represent the target sequence database.

```
$ hmmsearch –tblout <tabular_output> --noali –cut_ga <ModEnzA_DB> <any_species_proteome>
```

The tabular output is parsed to identify all the enzymes that produce hits above the discrimination threshold used. The file with the listing of enzymes is saved as ‘complete_ec_list’.

7. Identification of neighbors

The enzyme list obtained from the above step was used to identify the neighbors using the EE graph. The inbuilt function ‘neighbors’ from the CPAN package ‘Graph’ was used to identify the neighbors for each of the enzymes in the input list. The resulting neighbors were mapped back to the input enzyme list and all those enzymes which are already known in the organism are discarded since these constitute the known metabolic enzymes. The identification of neighbors is done by a perl script 'get_neighbor.pl'

```
$ perl get_neighbor.pl <complete_ec_list>
```

The output of this script is a file (in txt format) having the neighbors for each of the input enzyme and is saved as ‘complete_ec_list.neighbor’.

8. Removal of neighbors having partial EC number

This list of neighbors ‘complete_ec_list.neighbor’ is then checked for the presence of partial EC number and all those enzymes having incomplete information were discarded from further analysis using ‘sed’ command. Duplicated entries were removed using sort. These neighbors constitue a probable set of candidates for identification of local holes in the given organism.

```
$ sed -i '/-/d'   complete_ec_list.neighbor
$ sort -u  complete_ec_list.neighbor > complete_ec_list.neighbor.sort
```

9. Identification of probable local holes

The neighbors for each of the enzymes were mapped to the list of all known enzymes for the organism and all those nodes were predicted as local holes in that organism for which all the neighbors are known in the organism. This is done by the script 'identify_local_holes.pl'

```
$ perl identify_local_holes.pl <complete_ec_list> <complete_ec_list.neighbor>
```

The output of this script gives a tab separated file with the list of local holes, followed by number of out-edges, names of out-edges (i.e. the EC numbers), number of in-edges and names of it, followed by sum of in-edges and out-edges. From this file, we can separate the choke points, where there should be both in and out degree and their sum is 2. On the other hand, the lhighly probable local holes are those which have both in-degree and out-degree and their sum is greater than 2.