Atom Tracing: An Investigative Tool for the Study of Host-Gut Microbiota Co-Metabolism

A Senior Honors Thesis for the

Tufts University Department of Chemical and Biological Engineering

By Bassel C. Ghaddar

May 2015

Co-Advised by Professor Kyongbum Lee[1] and Professor Soha Hassoun[2]

[1]Department of Chemical and Biological Engineering

[2]Department of Computer Science

Introduction

The gut microbiota are comprised of approximately 10^{14} microorganisms divided into several phyla, the chief of these being *Actinobacteria, Bacteroidetes, Firmicutes,* and *Proteobacteria* (Sridharan, 2014). They colonize the gastrointestinal (GI) tract at birth, and develop a close relationship with the host immune system. Interactions between the microbiota and host immune system, transmitted through a vast array of signaling pathways and molecules, shape the development of the host and the composition of the microbiota (Nicholson, 2012).

Such chemical crosstalk has created a symbiotic relationship between the host and microbiota, and in some respects, has intertwined host cellular metabolic pathways with microbial activity, exemplified by microbial production of various beneficial and detrimental compounds to the host, and the combinatorial metabolism of substrates by the host and microbiota (Nicholson, 2012). For example, microbiota in the colon metabolize complex carbohydrates and ferment them into short-chain fatty acids, mainly acetate, propionate, and butyrate. Colonic epithelial cells in turn utilize butyrate for energy, and acetate and propionate serve as substrates for gluconeogenesis and lipogenesis in the liver and peripheral organs. These short chain fatty acids also modulate colonic gene expression and metabolism through enzymatic inhibition and through interaction with G-protein coupled receptors (Tremaroli & Backherd, 2012), and have been shown to induce differentiation of naive T cells into anti-inflammatory regulatory T cells (Arpaia, 2013). Additionally, the gut microbiota are involved in the synthesis of bile acids, choline, indole, and various other metabolites that aid the health and fitness of the host (Nicholson & Wilson, 2003).

Though an individual’s initial microbial seeding population is passed maternally at birth, the microbiota composition profile varies both spatially and temporally within the GI tract (Gordan, 2012), and can be influenced by changes in health and disease, diet, life-style, and antibiotic use (Nicholson J. , 2012). Disruptions to the microbiota profile (dysbiosis) are associated not only with gastric ulcers and inflammatory bowel diseases (Chassaing & Darfeuille-Michaud, 2011), but are also increasingly correlated with insulin resistance, type 2 diabetes, obesity, and cardiovascular disease (Wang, 2011). Despite the gut microbiota's great implications on host physiology and health, knowledge of which bacterial genomes contribute to the production of which bioactive gut metabolites is currently limited, owing to the complexity of the GI tract's metabolite spectrum, the difficulty of isolating and cultural intestinal bacteria, and the need to account for community level metabolic interactions (Sridharan, 2014).

In this work, we present a computational method to investigate gut microbiota-host metabolic interactions. Due to the diversity of species present in the gut that may contribute to the metabolism of certain substrates and the subsequent community level interactions between the different bacterial species and the host cells, an atomic level approach is taken. In particular, the Kyoto Encyclopedia of Genes and Genomes (KEGG) is used to assemble various metabolic models representing different proportions of murine host cells and various bacterial phyla (Kanehisa Laboratories, 2015). Pathways between common dietary nutrients that would be present in the GI tract to bioactive gut metabolites of interest are then investigated using a reachability analysis and a "random-walks" pathfinding algorithm. Finally, an atom tracing function is implored to determine the conservation of atoms along the pathway, and model results are compared against each other. It is the hope that such an analysis will lead to new insights to the atomic level contribution of various gut flora to the synthesis of bioactive gut metabolites.

Methods

To investigate the atomic level contributions of the murine host cells and the various bacterial phyla to the metabolism of dietary nutrients, the KEGG database was used to construct various models representing different proportions of *Mus musculus* mouse (MMU) cells and gut microbiota (GM) cells. For this analysis, all host cells were considered to have equal metabolic capabilities. The microbiome was divided into five phyla: *Actinobacteria, Bacteroidetes, Firmicutes, Proteobacteria*, and *Miscellaneous*, labeled P1 to P5, respectively.

A composite list of chemical reactions was compiled, and for a first analysis, each reaction's likelihood of occurring was weighted based on the fraction of total cells that could perform it. To determine the bacterial contribution to the reaction weights, a phyla score (Sridharan, 2014) was assigned to each reaction, which was simply the fraction of each phylum that could perform the reaction. Relative abundance of the bacterial phyla in lean mice, mice on a high fat diet, and genetically obese mice were obtained from (Murphy, 2010). The reaction weight for each reaction *i* in a system of *j* bacterial phyla was subsequently calculated as thus:

$$Reaction\ Weight(i) = \left(\sum_{j} (Phyla\ Score)_j (Relative\ Abundance)_j\right)(1 - MMU\ Fraction) + MMU\ Fraction$$

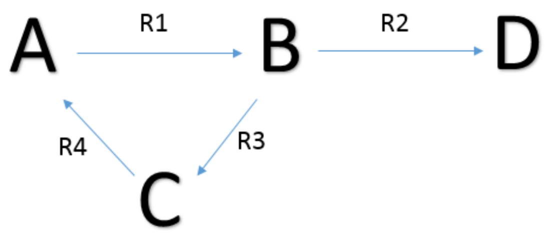
Twelve distinct models were constructed representing different proportions of host cells and bacterial phyla. The twelve cases are detailed in Table 1.

Table 1. Fraction of each cell type in the 12 metabolic models representing different proportions of host cells to the various bacterial phyla assembled using the KEGG database. The *Miscellaneous (Misc.)* phylum is a conglomeration of all other known bacterial phyla and species.

Model	***Host***	***Actinobacteria***	***Bacteroidetes***	***Firmicutes***	***Proteobacteria***	***Misc.***
Host (MMU)	1	0	0	0	0	0
Gut Microbiota (GM)	0	0.2450	0.1690	0.5660	0.0130	0.0070
MMU/GM = 1/10	0.09	0.2227	0.1536	0.5145	0.0118	0.0064
MMU/GM = 1/1	0.5	0.1225	0.0845	0.2830	0.0065	0.0035
MMU/GM = 10/1	0.91	0.0223	0.0154	0.0515	0.0012	0.0006
Host on High Fat Diet	0.5	0.3525	0.0760	0.0655	0.001	0.0050
Genetically Obese Host	0.5	0.3175	0.0610	0.1145	0.0030	0.0015
MMU + GM – P1	0.5	0	0.2238	0.7497	0.0172	0.0093
MMU + GM – P2	0.5	0.2948	0	0.6811	0.0156	0.0084
MMU + GM – P3	0.5	0.5645	0.3894	0	0.0300	0.0161
MMU + GM – P4	0.5	0.2482	0.1712	0.5735	0	0.0071
MMU + GM – P5	0.5	0.2467	0.1702	0.5700	0.0131	0

For each model of *m* compounds and *n* reactions, an *m x n* stoichiometric matrix (S matrix) was constructed, where each row represented a metabolite and each column represented a reversible chemical reaction. Using these S matrices, *m x m* adjacency matrices (AM) were constructed to map the immediate connectivity of each metabolite, where each row and column represent a metabolite. All entries in the AM are zero unless there is a one-step reaction connecting two metabolites, in which case the value of the AM entry is the sum of all the reaction weights of all the one-step reactions that connect the given metabolites. Essentially the S matrices represent undirected graphs whose nodes are metabolites and edges are reaction weights, and the AMs detail which nodes are connected by a single edge. Since the graphs are undirected, the AMs are symmetric.

The AMs were in turn used to construct *m x m* reachability matrices (R matrices), whose (*i*,*j*) entry contains a value of one if a path exists between metabolites *i* and *j*, or a zero if no such path exists. The R matrices can thus be used to identify metabolites in the model whose reachability depends on the presence of certain cell types. An example S matrix, AM, and R matrix are shown for the simple system below.



$$S = \begin{bmatrix} & R1 & R2 & R3 & R4 \\ A & -1 & 0 & 0 & 1 \\ B & 1 & -1 & -1 & 0 \\ C & 0 & 0 & 1 & -1 \\ D & 0 & 1 & 0 & 0 \end{bmatrix} \quad AM = \begin{bmatrix} & A & B & C & D \\ A & 0 & 1 & 1 & 0 \\ B & 1 & 0 & 1 & 1 \\ C & 1 & 1 & 0 & 0 \\ D & 0 & 1 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} & A & B & C & D \\ A & 1 & 1 & 1 & 1 \\ B & 1 & 1 & 1 & 1 \\ C & 1 & 1 & 1 & 1 \\ D & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 1. Example reaction system to illustrate the construction of an S matrix, AM, and R matrix. Similar to the KEGG database, all reactions are assumed reversible. No reaction weights are assumed for this system.

Subsequent pathway analysis was narrowed down to pathways from common dietary nutrients (Table S1) to a subset of metabolites (Table S2) suspected of requiring host-microbiota metabolic interactions (Nicholson J. , 2012). Utilizing the constructed AMs, a "random walks" algorithm was implemented to find pathways from the starting nutrients to the target metabolites. Each random walk begins at a starting node on the metabolic graph and takes steps based on the weights of the connecting edges available. A walk is terminated if the target node is reached, if the only available paths to take lead to an already traversed node, if a biochemical cofactor (Table S3) is reached, or if the maximum number of steps is reached. Because cofactors are involved in a wide array of reactions, they were used as a walk-terminating factor to limit the number of unrealistic reaction pathways found by the random-walks function. Traversing only new nodes was also included as a condition to prevent walking through cycles.

The resulting lists of reached walks were analyzed for path distances, path probability, and path connectivity. These properties were defined below:

$$Path\ Distance = Number\ of\ nodes\ traversed\ in\ the\ walk \tag{1}$$

$$Path\ Probability = \prod_{Starting\ Metabolite}^{Ending\ Metabolite} \frac{Chosen\ Reaction\ Weight}{\sum Possible\ Adjacent\ Reaction\ Weights} \tag{2}$$

$$Path\ Connectivity = \sum_{Starting\ Metabolite}^{Ending\ Metabolite} Number\ of\ adjacent\ nodes \tag{3}$$

Once a list of reached walks from the starting to the target metabolites is obtained, an atom tracing function is used to trace the atoms of the starting metabolite by using KEGG RPair data. The atom tracing function determines which atoms of the starting metabolite are conserved throughout the reaction path, as well as their location on the final metabolite.

The abovementioned analyses were performed in MATLAB; please see the Supplementary Information section for details on the functions developed to perform these analyses.

Results and Discussion

Reachability Analysis

Metabolite level (Figure 2) and atomic level (Figure 3) R matrices were computed for the models with unique groupings of cell types: MMU, GM, MMU+GM-P1, MMU+GM-P2, MMU+GM-P3, MMU+GM-P4, and MMU+GM-P5.

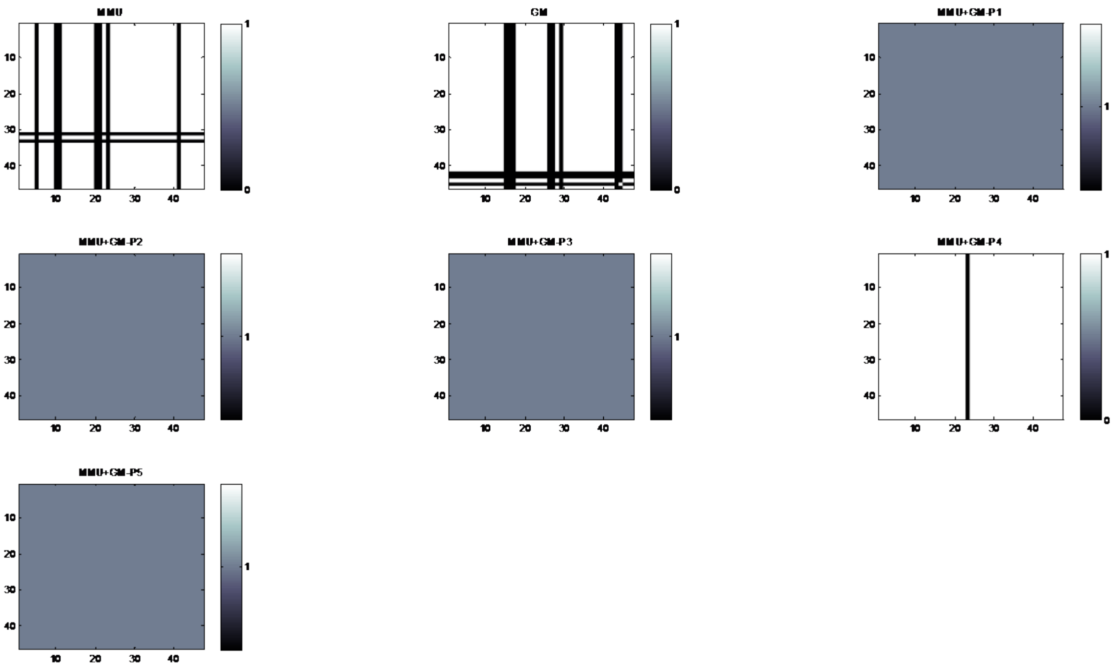


Figure 2. Color-maps of metabolite level R matrices from the metabolites in Table S1 (vertical axis) to the metabolites in Table S2 (horizontal axis) for the following models: MMU, GM,

MMU+GM-P1, MMU+GM-P2, MMU+GM-P3, MMU+GM-P4, and MMU+GM-P5. Values of "1" indicate the existence of a path connecting a metabolite pair, while values of "0" indicates the absence of such a path.

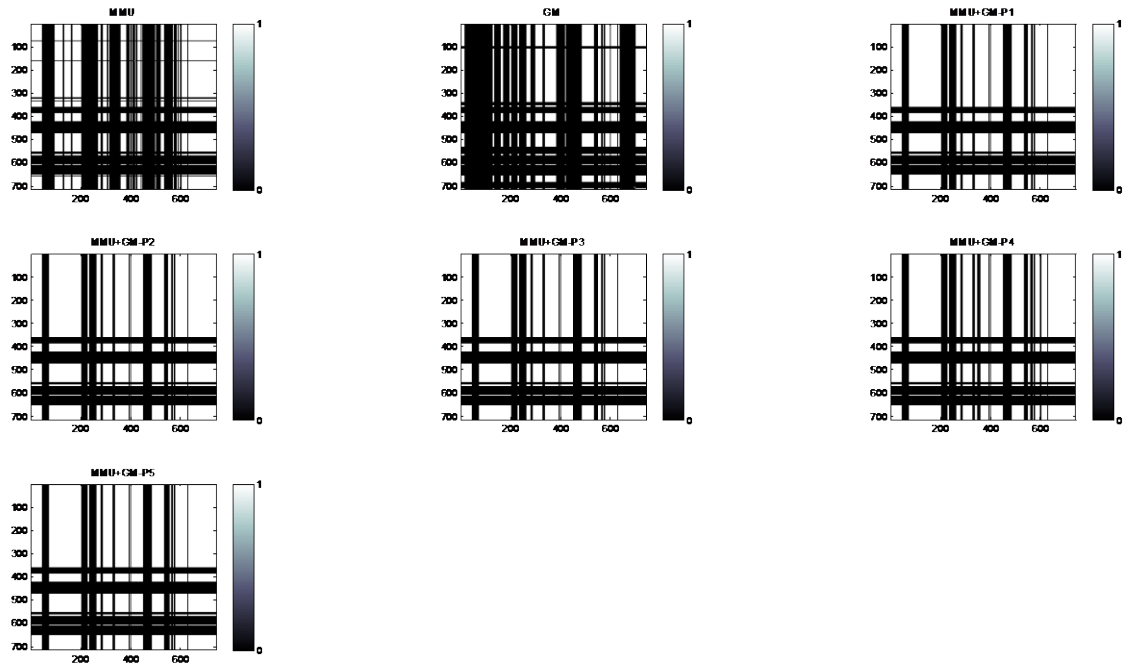


Figure 3. Color-maps of atomic level R matrices from the metabolites in Table S1 (vertical axis) to the metabolites in Table S2 (horizontal axis) for the following models: MMU, GM, MMU+GM-P1, MMU+GM-P2, MMU+GM-P3, MMU+GM-P4, and MMU+GM-P5. Values of "1" indicate the existence of a path connecting a metabolite pair, while values of "0" indicates the absence of such a path.

The R matrices indicate whether a path exists between any two nodes on the constructed metabolic graphs. Figure 2 shows that there are some metabolite pairs that are not reachable from each other without either host or bacterial metabolism. A closer look at the GM alone metabolite level R matrix (Figure S1) reveals that trimethylamine, trimethylamine-N-oxide, dimethylglycine, phenylacetylglycine, phenylacetate, melatonin, sphingomyelin, and cholesterol are not reachable from the listed starting metabolites without a metabolic contribution from the host.

Inspection of the MMU R matrix (Figure 2 and Figure S2) and the MMU+GM minus a bacterial phylum (Figure 2) suggests that there are reactions shared across the various microbiota phyla that the host cannot perform. It was found that removal of one phyla at a time only changed metabolite reachability when *Proteobacteria* were eliminated, and only for reachability to 3-hydroxycinnammate (Figure 2). However, the general MMU R matrix shows that reachability to deoxycholate, glycodeoxycholate, taurodeoxycholate, hippuric acid, 3-hydroxybenzoic acid, and general lipopolysaccharide, in addition to 3-hydroxycinnammate, depends on the presence of some bacterial species.

A comparison of the metabolite level and atomic level reachability matrices (Figures 2 and 3, respectively) indicates a much smaller degree of connectivity at the atomic level, which is immediately obvious by the significantly greater number of zero entries in the atomic level R matrices. When the loss-of-phyla metabolic networks are analyzed at the metabolite level, it appears that only removing *Proteobacteria* affects reachability, but the atomic level analysis indicates that all phyla are essential for the reachability of some metabolites. Furthermore, the various atomic level R matrices show that upon removing certain phyla, it is not only certain atoms of the target metabolites that become unreachable, but some metabolites in their entirety. For example, removing phylum 5 (*Misc.*), which in all the models accounted for at most 1.61% of all cell types, completely erased reachability to dimethylamine, trimethylamine, trimethylamine-N-oxide, and 3-hydroxybenzoic acid; this result was not apparent from the metabolite level R matrices, which, due to the simpler structures of the metabolite level AMs, find many unphysical paths. This highlights the necessity of taking an atomic approach for the analysis of complex, combinatorial metabolic networks.

Random Walks and Atom Tracing

The R matrices were used to identify metabolites whose reachability from the amino acids did or did not depend on the presence of some bacterial phyla. Indolepyruvate, dopamine, indole-acetate, and serotonin were identified as metabolites whose reachability on the metabolite graphs from the amino acids was independent of bacterial presence, and 3-hydroxycinnamate was identified as requiring *Proteobacteria*. The random walk and atom tracing analysis was performed for these two groups of metabolites.

Indolepyruvate and Dopamine GM Walks Analysis

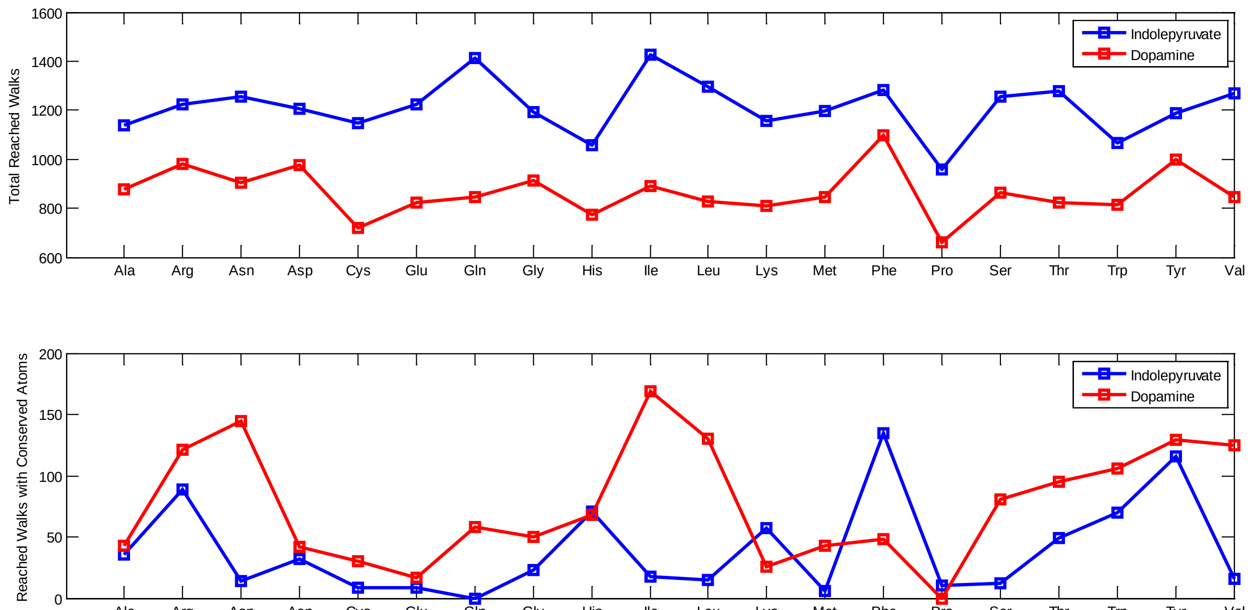


Figure 4. Random walk results from the amino acids to indolepyruvate and dopamine on the GM graph. Data shown is total number of reached walks and the reached walks with at least one conserved atom. One hundred thousand walks were run from each amino acid to each target molecule.

The binary reachability matrices do not give an indication to the degree of reachability, thus it is important to implore a pathfinding algorithm, such as the random walks method utilized in this analysis, to compare synthesis routes to the target molecules from various starting points. The random walks results to indolepyruvate and dopamine on the GM graph confirm the GM R matrix result that these two metabolites are reachable from all the amino acids. Out of 100,000 random walks run to these two metabolites, glutamine and isoleucine had the most reached walks to indolepyruvate (1415 and 1425 walks, respectively), while phenylalanine had the greatest number of reached walks to dopamine (1098 walks). This data did not correlate with the atom tracing results, which indicated that phenylalanine atoms ended up in indolepyruvate most frequently (143 walks), and isoleucine atoms ended up in dopamine most frequently (166 walks). These results also affirm the notion that metabolites are connected much less on the atomic level that it would appear when simply analyzing reactions at the metabolite level.

To confirm that 100,000 walks was sufficient to reach the target metabolites at a consistent frequency, the percent of walks reached was plotted against the number walks for both indolepyruvate and dopamine (Figures S3 and S4). It was found that for both metabolites, about 30,000 walks were enough to reach a consistent fraction of reached walks. The reached walks were then analyzed for path distance (Figure 5), path probability (Figure 6), and path connectivity (Figure 7).

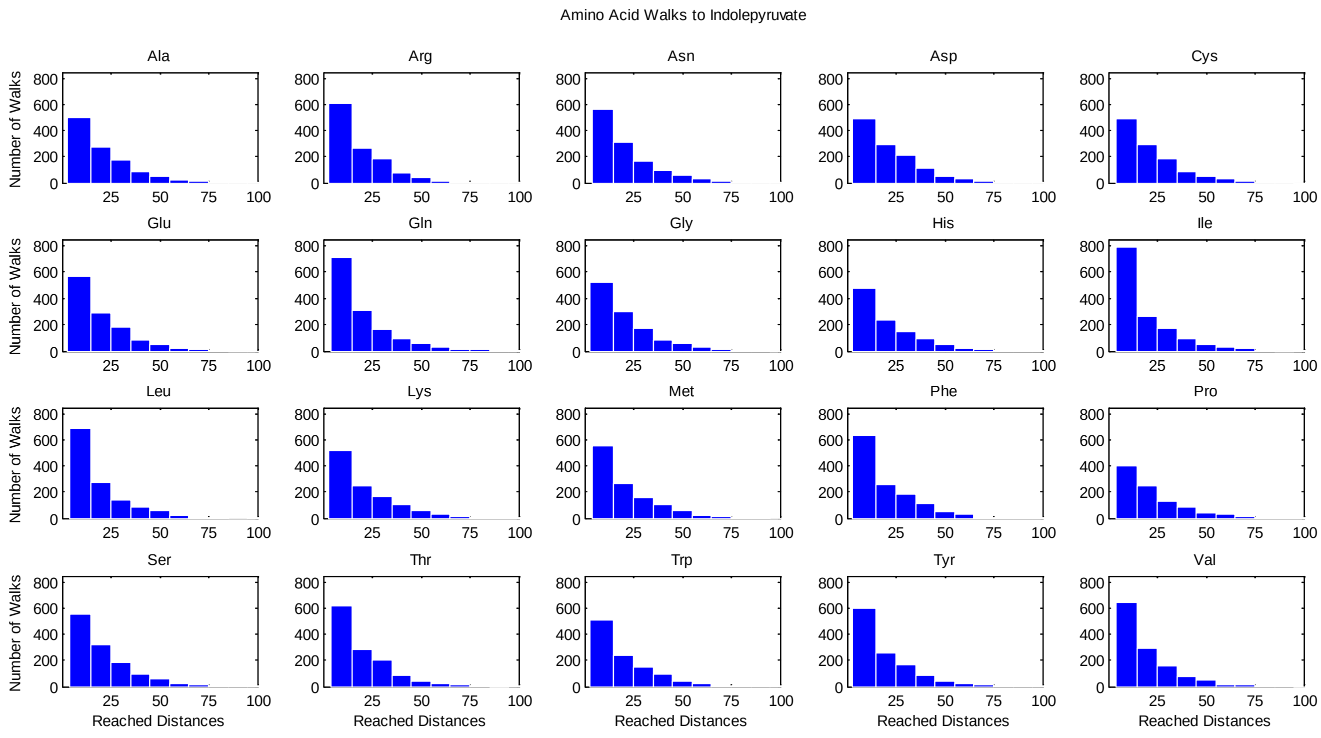


Figure 5. Histograms of path distances as defined in eq. (1) for the reached walks from the amino acids to indolepyruvate on the GM graph. One hundred thousand walks walks were run from each amino acid.

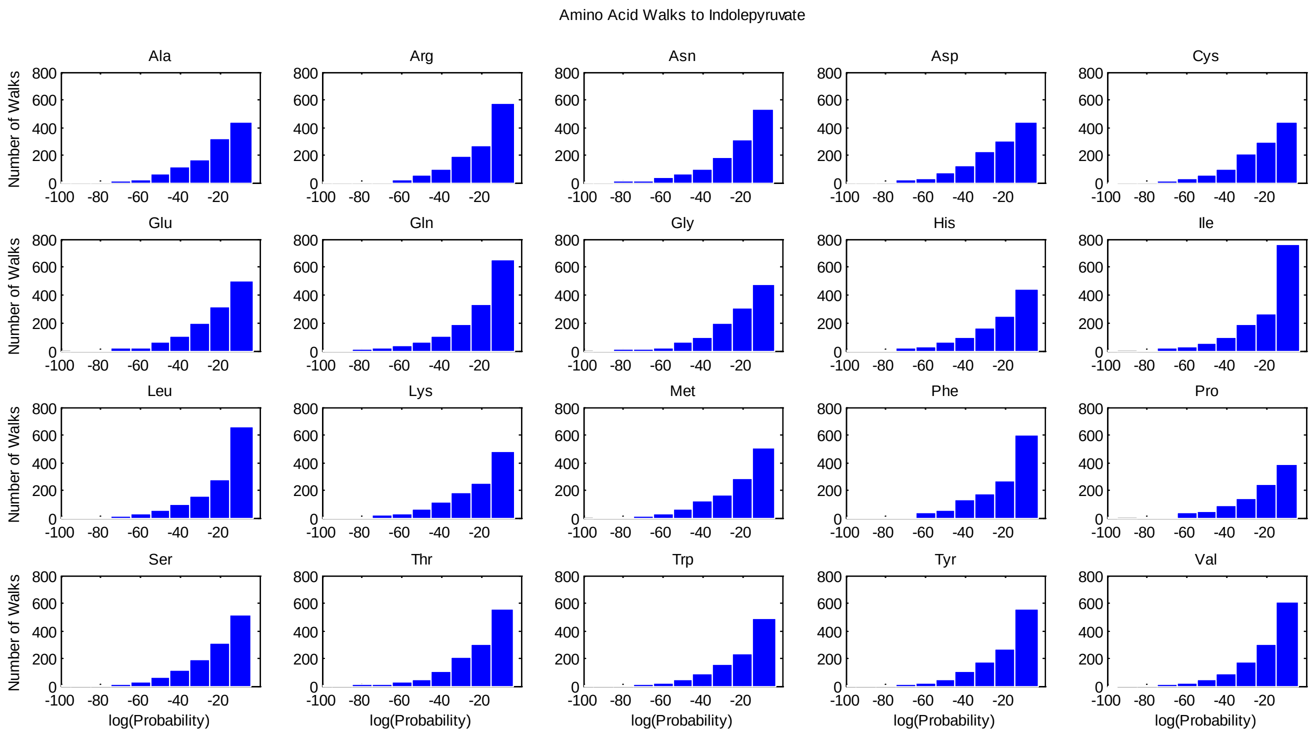


Figure 6. Histograms of the logarithm of path probabilities (eq. (2)) for the reached walks from the amino acids to indolepyruvate on the GM graph. One hundred thousand walks were run from each amino acid.

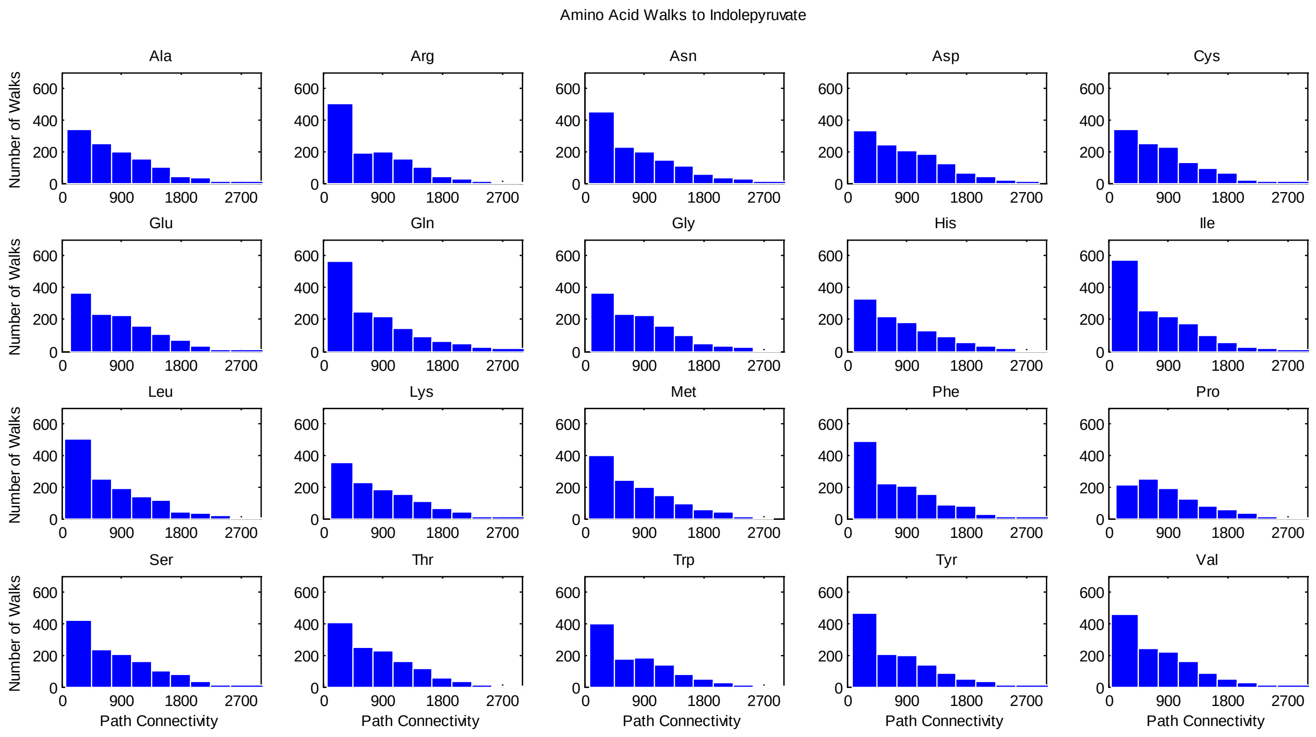


Figure 7. Path connectivity (eq. (3)) histograms for the reached walks from the amino acids to indolepyruvate on the GM graph. One hundred thousand walks were run from each amino acid.

The path distance histograms were all positively skewed, indicating that the shorter a reached path, the more frequently this path was found, though the slopes of these histograms were different for the various amino acids. The log (probability) histograms were all negatively skewed, indicating that paths with higher reaction weights were reached more frequently, though at different rates for the amino acids. The connectivity histograms were all positively skewed, indicating that reached paths with a lesser number of branching nodes were reached more frequently. Together, these histograms show that the likelihood of reaching a target metabolite from some starting point on a metabolic graph is a function of the node distance, the edge weighting, and the degree of path branching. Note that the same trends were observed for reached walks to dopamine (Figures S5, S6, and S7).

The atom tracing function was used to determine which atoms of the starting amino acids were conserved throughout the reached walks. The results are shown in Figure 8.

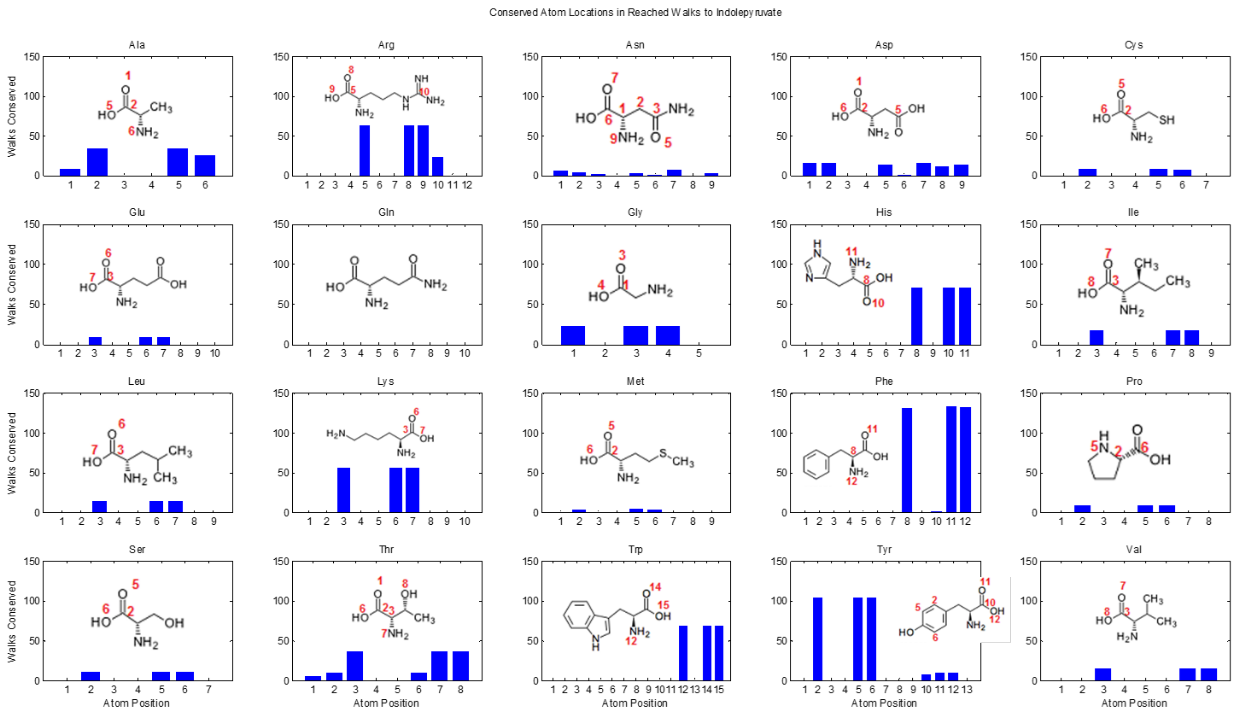


Figure 8. Bar plot of the number of times an atom was conserved vs. the atom's original location on each amino acid for reached walks from the amino acids to indolepyruvate on the GM graph, out of a total of 100,000 walks.

The atom tracing data identifies which atoms were conserved throughout a pathway. Figure 8 shows that often times, groups of atoms will be conserved together; for example, arginine, aspartic acid, cysteine, glutamic acid, glycine, isoleucine, leucine, lysine, methionine, serine, and valine all apparently underwent a decarboxylation reaction, and it was the CO_2 that ended up in indolepyruvate. This can be inferred because the carboxylic acid portion of these amino acids was always conserved together in walks from these amino acids. The atom tracing data also reveals that some amino acids had a greater number of atoms conserved per walk than others, and suggests atoms to label for potential labeling experiments.

Indolepyruvate, Dopamine, Indoleacetate, and Serotonin Walks on All Models

Random walks were next run from the amino acids to indolepyruvate, dopamine, indole-acetate, and serotonin on all 12 models.

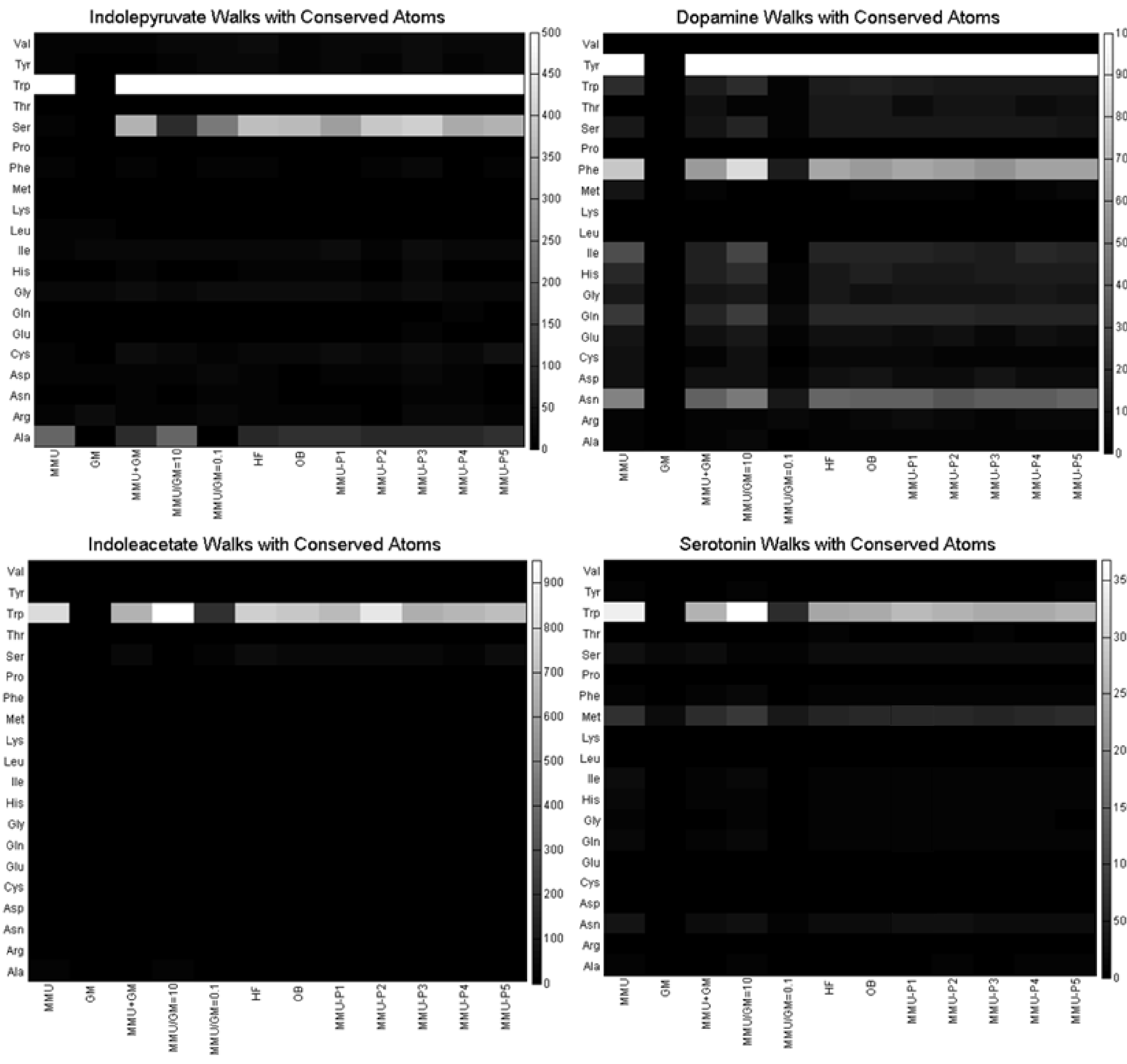


Figure 9. Heat-maps of the number of reached walks from the amino acids to indolepyruvate, dopamine, indoleacetate, and serotonin on all 12 models. One hundred thousand random walks were run from each amino acid to each target molecule. Note that the same data is shown graphically for select amino acids in Figure S8.

The results of these walks indicate a great degree of variation in the likelihood of reaching the target molecule from different amino acids. For example, indoleacetate was significantly more reachable from tryptophan than it was from any other amino acid, while dopamine was reachable to a significant degree by tyrosine, phenylalanine, and asparagine. A comparison of results across various models shows that the greatest factor in determining the degree of reachability from the amino acids to these molecules was the ratio of host cells to microbiota (columns 1-5 in Figure 9). All four target molecules were reached the most when there was a 10:1 MMU:GM ratio, and were least reached on the GM alone graph. Altering the microbiota profile in the high fat diet model, obese model, and loss of phylum models, did not significantly affect the frequency of reaching these target molecules. An interesting result that was observed, however, is that sometimes removing a phylum increased the number of walks reached. When phylum 3 (*Firmicutes*) was removed, indoleacetate was reached much more from tryptophan than when this phylum was present. *Firmicutes* accounts for a significant portion (56%) of the microbiota profile in lean mice (Murphy, 2010). Since the probability of traversing a node was simply proportional to the number of cells and reactions that lead to it, removing a major portion of reactions that do not lead to the target molecule would increase the probability of selecting reactions that do lead to the desired destination.

Random Walks to 3-Hydroxycinnamate

3-Hydroxycinnamate synthesis from the amino acids was identified by the metabolite level R matrices as requiring the *Proteobacteria* genome. Random walks to 3-Hydroxycinnamate were run from the amino acids and select carbohydrates on all 12 models (Figure 10).

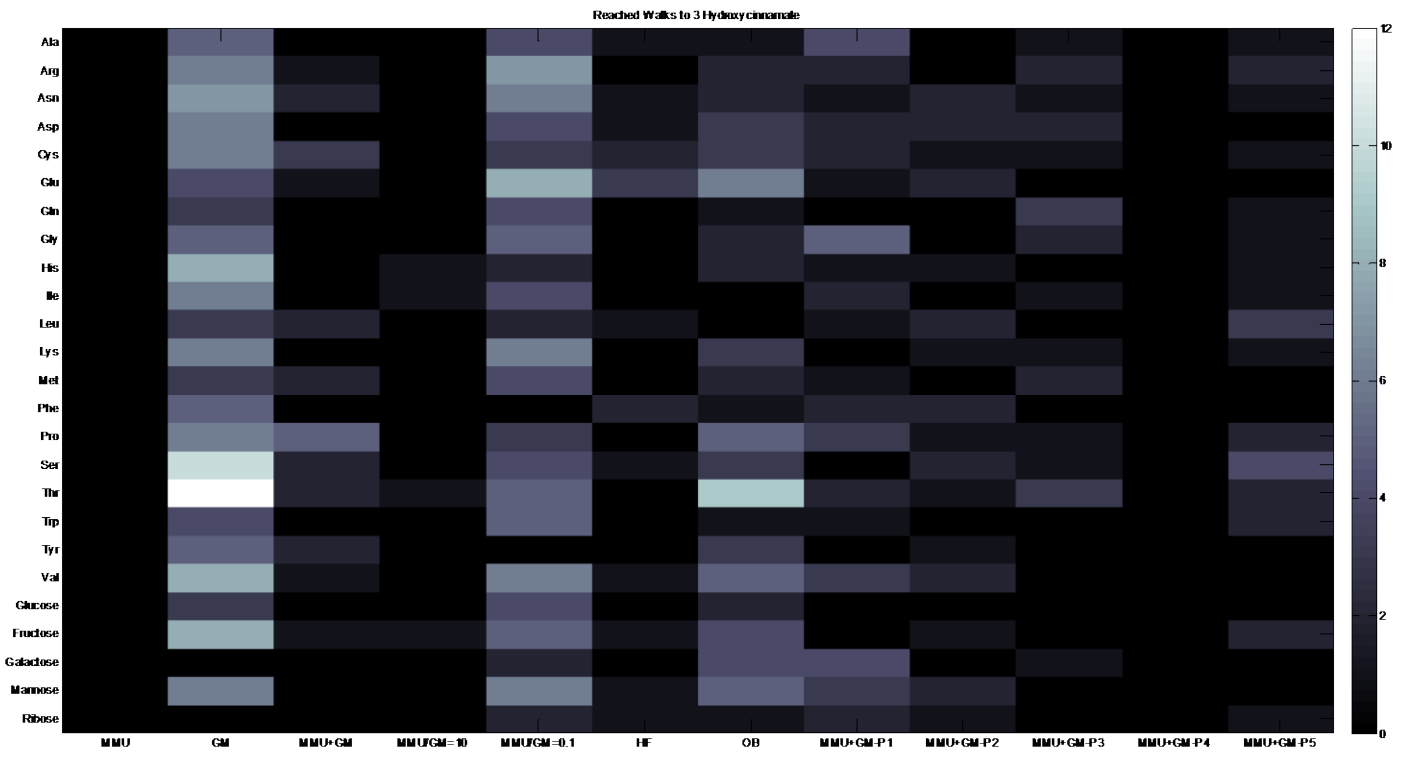


Figure 10. Heat-map of the number of reached walks from the amino acids and some carbohydrates to 3-Hydroxycinnamate on all 12 models. One hundred thousand random walks were run from each starting molecule.

As expected, the models that excluded *Proteobacteria* (MMU and MMU+GM-P4) did not reach 3-hydroxycinnamate from any of the starting molecules. These results show in the number of walks reached from the various starting points in a given model, as well as from the same starting molecule across the various models, including the models with varying bacterial phyla proportions. In this experiment, since reachability depended on the presence of *Proteobacteria*, models with smaller MMU:GM ratios had more reached walks.

Though Figure 10 shows differences in the number of reached walks to 3-hydroxycinnamate, the differences across the models are not significant, as the most the target metabolite was reached from any starting point was 12 times. Examination of the reaction pathways to 3-hydroxycinnamate shows that the closer to the target metabolite, the more specific the path becomes.

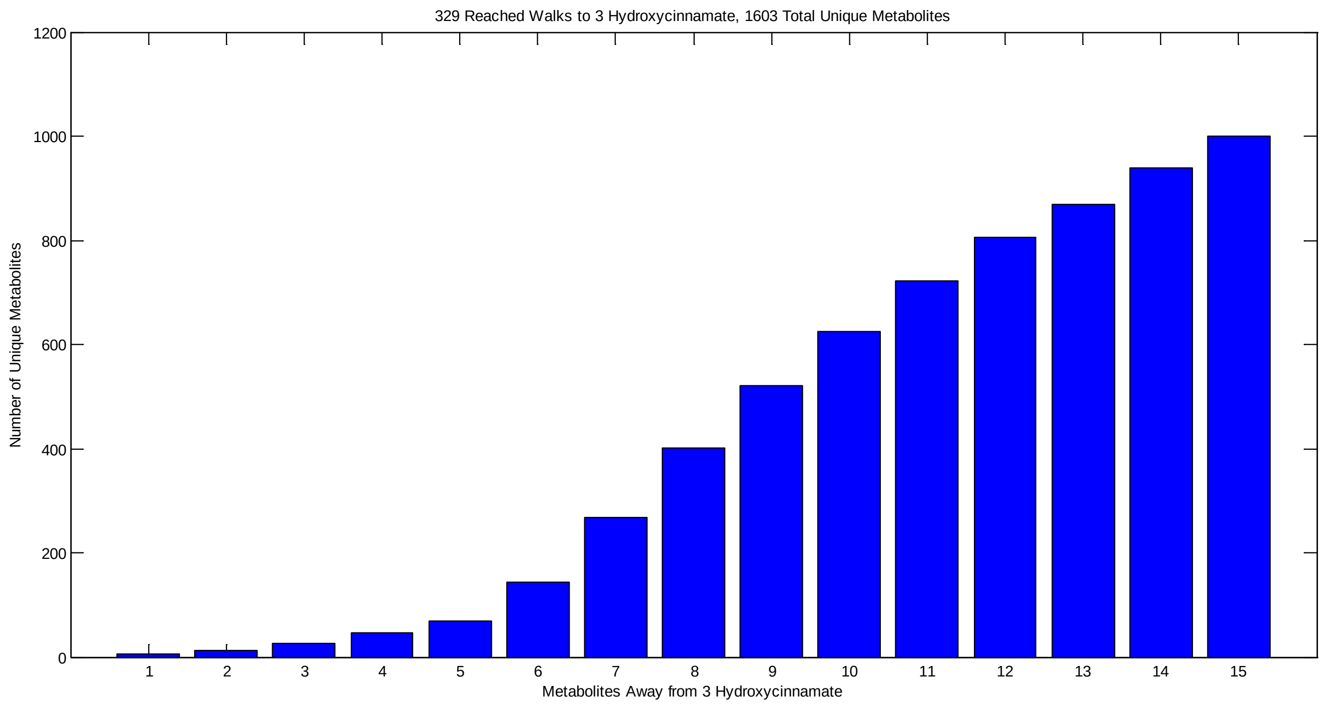


Figure 11. One hundred thousand random walks were run from each of the amino acids and glucose, fructose, galactose, mannose, and ribose to 3-hydroxycinnamate. Of all these walks, 329 walks reached the target, and of these reached walks, there were 1603 unique metabolites traversed. The plot above shows the number of unique metabolites in the paths versus the number of metabolites away from the target.

The data in Figure 11 shows that there is a limited number of pathways that lead to 3-hydroxycinnamate, for closer to the target metabolite there are less possible nodes that the walk could be on. One potential way to increase the number of reached walks to 3-hydroxycinnmate is to start the paths at a point closer to the target. The paths leading to 3-hydroxycinnamate were searched for potential common dietary metabolites that may be used as good starting metabolites to reach 3-hydroxycinnammate. Though no common nutrients other than the starting metabolites originally tested were found, it was found that about 90% of the reached walks to 3-hydroxycinnamate went through 4-hydroxy-2-oxopentanoate. Synthesis of 4-hydroxy-2-oxopentanoate is regulated by the enzyme 4-hydroxy-2-oxovalerate aldolase, which catalyzes the reaction between pyruvate and acetaldehyde to form 4-hydroxy-2-penatanoate. The enzyme is found primarily in the bacterium *Burkhodleria xenovorans,* a member of the phylum *Proteobacteria* (Baker, et al., 2009), which may explain why the presence of *Proteobacteria* is required for 3-hydroxycinnamate synthesis. It is conceivable that the introduction of more *Burkhodleria xenovorans* to the gut, or the reverse engineering of existing cell to overexpress 4-hydroxy-2-oxovalerate aldolase would increase the synthesis of 4-hydroxy-2-oxopentaoate; random walks were thus run to determine the degree of 3-hydroxycinnammate's reachability from 4-hydroxy-2-oxopentanoate.

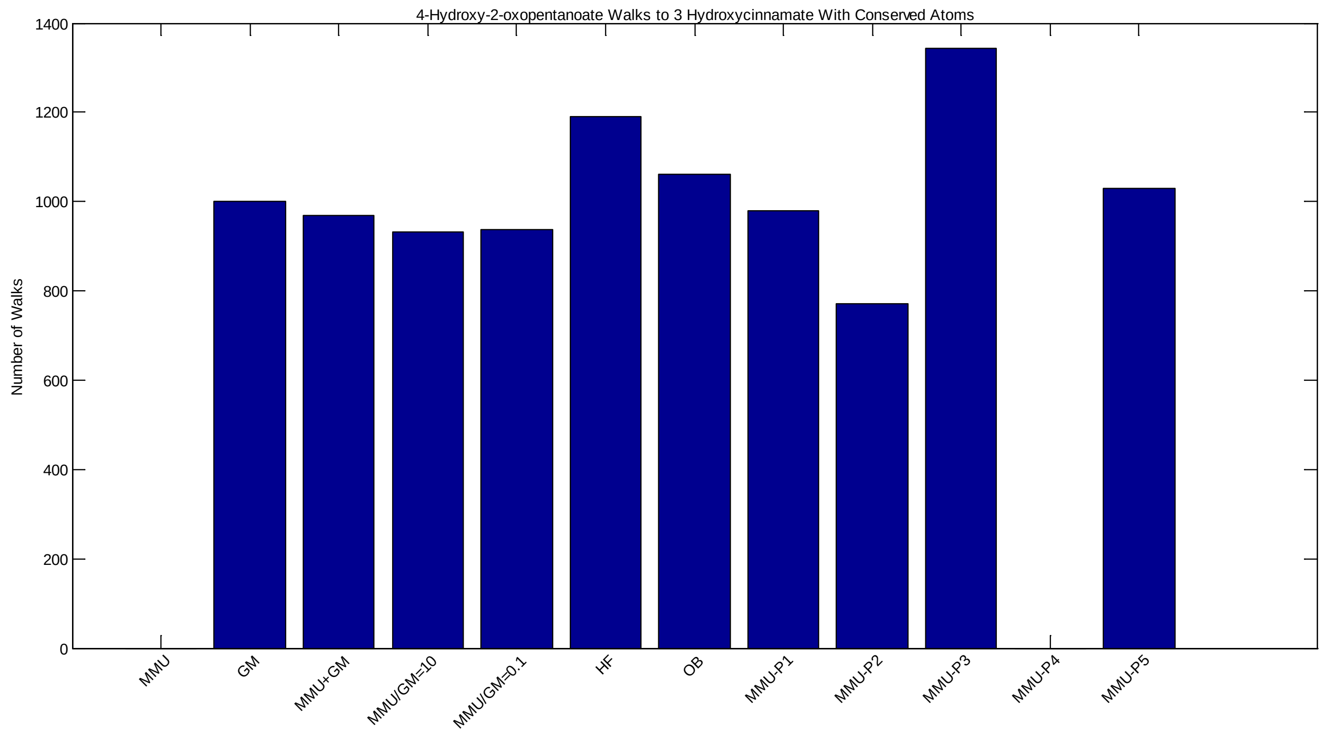


Figure 12. Reached walks with conserved atoms from 4-hydroxy-2-oxopentaoate to 3-hydroxycinnamate for all 12 models. Thirty thousand walks were conducted.

Random walks starting from 4-hydroxy-2-oxopentanoate reached 3-hydroxycinnamate significantly more than did walks from the amino acids and select carbohydrates. There was large difference in the number of walk reached with conserved atoms from the MMU+GM-P2 compared to the MMU+GM-P3 (a difference of 570 walks). Removing phylum 2 (*Bacteroidetes*) decreased the number of walks reached, while removing phylum 3 (*Firmicutes*) increased the number of reached walks. *Bacteriodetes* may be able to perform some unique reactions that aid in the synthesis of 3-hydroxycinnamate, while *Firmicutes* may contain a large number of reactions nonspecific to 3-hydroxycinnmamate synthesis, and thus removing increased the reaction weighting of the reactions leading to the target molecule. Again, reachability to the 3-hydroxycinnamate depended on the presence of *Proteobacteria,* indicating that 4-hydroxy-2-oxovalerate aldolase was not the only limiting enzyme. A closer look at the reached random walks reveals that they all followed the series of chemical reactions shown in Figure 13.

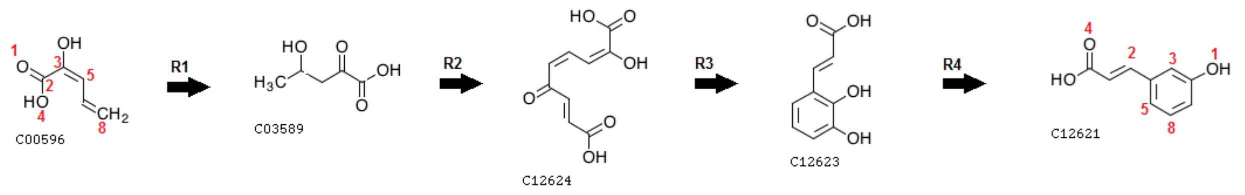


Figure 13. Pathway from 4-hydroxy-2-oxopentanoate to 3-hydroxycinnamate. The metabolite number refers to the KEGG compound ID number. The atoms numbered are the conserved atoms.

All of the reached walks starting from 4-hydroxy-2-oxopentanoate required the transformation of 4-hydroxy-2-oxopentanoate to 2-Hydroxy-2, 4-pentadienoate (Figure 13, R1), a reaction regulated by the enzyme 2-hydroxy-6-oxo-6-phenylhex-2, 4-dienoate reductase, found in the bacterium *Pseudomonas cruciviae* of the *Proteobacteria* phylum (Omori, Ishigooka, & Minoda, 1986). Though 4-hydroxy-2-oxopentanoate is synthesized from pyruvate, random walks starting from pyruvate to 3-hydroxycinnamate did not reach the target with any appreciable frequency, due to the large number of reactions pyruvate can undergo.

Conclusions

This work presents an atomic level analysis of metabolic networks for the investigation of host-gut microbiota metabolic interactions. Various metabolic models representing different proportions of murine host cells and various bacterial phyla were assembled using the KEGG database. Utilizing reachability matrices, it was found that connectivity at the atomic level was significantly more limited than at the metabolite level. This was exemplified by the reachability matrices representing a system consisting of murine host cells (*Mus musculus*), *Actinobacteria, Bacteroidetes, Firmicutes,* and *Proteobacteria,* which was missing bacterial phylum 5, the *Miscellaneous* category. The metabolite level R matrix indicated that all the metabolites in Table S2 were reachable from the metabolites in Table S1. The atomic level R matrix, however, showed that several atoms were in fact not reachable, and that dimethylamine, trimethylamine, trimethylamine-N-oxide, and 3-hydroxybenzoic acid were completely unreachable without the *Miscellaneous* category of gut microbiota.

The random walk and atom tracing analysis were used to investigate the pathways from the amino acids to indolepyruvate, dopamine, indoleacetate, and serotonin, metabolites whose reachability was not found to depend on the gut microbiota. It was found that the likelihood of reaching a target metabolite from some starting point depended on the path distance, path connectivity, and the path probability. Random walks were run on the metabolite level adjacency matrices, so an atom tracing function was implored to find the number and location of conserved atoms throughout the pathways. The number of walks that reached the target node with conserved atoms was significantly less than the total number of walks reached simply at a metabolite level, and that these two measures did not correlate with each other. For these target metabolites, the frequency of being reached differed significantly amongst the amino acids, but for a given starting amino acid, the likelihood of reaching the target metabolite did not vary significantly when the composition of the gut microbiota was altered. The biggest factor that affected reachability was the ratio of host cells to gut microbiota; a ratio of MMU:GM = 10 was generally found to have greatest number of reached walks.

Synthesis of 3-hydroxycinnamate was identified using the metabolite level reachability matrices as requiring *Proteobacteria* genomes. Random walks were run from the amino acids and some select carbohydrates to this target for all the models, but only a negligible fraction of the walk reached their target. Further analysis of the reached pathways revealed that 4-hydroxy-2-oxopentanoate, a metabolite whose synthesis from pyruvate and acetaldehyde is catalyzed by 4-hydroxy-2-oxovalerate aldolase, an enzyme found primarily in the bacterium *Burkhodleria xenovorans*, a member of the *Proteobacteria* phylum, was a key step in the synthesis of 3-hydroxycinnamate. Random walks were run from 4-hydroxy-2-oxopentanoate to 3-hydroxycinnamate, and a significantly greater number of walks were reached. Furthermore, this lead to the identification of another key enzyme belonging to *Pseudomonas cruciviae* of the *Proteobacteria* phylum that regulated the synthesis of 3-hydroxycinnamate. For the 3-hydroxycinnamate trials, the random walks and atom tracing analysis facilitated the identification of key enzymes and reactions in the synthesis pathway, and determined the conservation of atoms along this pathway.

Improvements to the reaction weighting scheme may significantly improve the accuracy and predictive nature of this computational approach to investigating hot-gut microbiota interactions. The current model weights reactions solely based on the number of cells that can perform them; it assumes all metabolites are accessible to each other, and does not account for factors such as transport limitations or enzyme expression. Practical improvements to the reaction weighting scheme may include accounting for the various phyla’s locations along the digestive tract, favoring reactions sequences that occur in the same cell type, appropriately regulating reactions that cross membranes, and accounting for enzyme expression. Other future work should also focus on shifting all the analyses to the atomic level. With continued model improvement, this atom tracing analysis can become a very useful tool for the study of complex metabolic systems.

References

Arpaia, N. et al. (2013). Metabolites produced by commensal bacteria promote peripheral regulatory T-cell generation. *Nature*, 451-455.

Baker, P., Pan, D., Carere, J., Rossi, A., Wang, W., & Seah, S. (2009). Characterization of an Aldolase-Dehydrogenase Complex That Exhibits Substrate Channeling in the Polycholorinated Biphenyls Degradation Pathway. *Biochemistry*, 6551-6558.

Chassaing, B., & Darfeuille-Michaud, A. (2011). The commensal microbiota and enteropathogens in the pathogenesis of inflammatory bowel diseases. *Gastroenterology*, 1720-1728.

Gordan, J. (2012). Honor thy gut symbionts redux. *Science*, 1251-1253.

Kanehisa Laboratories. (2015, April 30). *Kegg: Kyoto Encyclopeida of Genes and Genomes*. Retrieved from KEGG: http://www.genome.jp/kegg/

Murphy, E. et al. (2010). Composition and energy harvesting capacity of the gut microbiota: relationship to diet, obesity, and time in mouse models. *Gut*, 1635-1642.

Nicholson, J. et al. (2012). Host-Gut Microbiota Metabolic Interactions. *Science*, 1262-1267.

Nicholson, J., & Wilson, I. (2003). Opinion: understanding 'global' systems biology: metabonomics and the continuum of metabolism. *Nat Rev Drug Discov*, 668-676.

Omori, T., Ishigooka, H., & Minoda, Y. (1986). Purification and some properties of 2-hydroxy-6-oxo-6-phenylexa-2,4-dienoic acid (HOPDA) reducing enzyme from Pseudomonas cruciviae S93B1 involved in the degradation of biphenyl. *Agir. Biol. Chem.*, 1513-1518.

Sridharan, G. V. et al. (2014). Prediction and quantification of bioactive microbiota metabolites in the mouse gut. *Nature Communications*, 5492.

Tremaroli, V., & Backherd, F. (2012). Functional interactions between the gut microbiota and host metabolism. *Nature*, 242-249.

Wang, Z. et al. (2011). Gut flora metabolism of phophatidylcholine promotes cardiovascular disease. *Nature*, 57-63.

Supplementary Information

Table S1. Starting dietary nutrients

Compound Name	KEGG ID
Alanine	41
Arginine	62
Asparagine	152
Aspartic Acid	49
Cysteine	97
Glutamic Acid	25
Glutamine	64
Glycine	37
Histidine	135
Isoleucine	407
Leucine	123
Lysine	47
Methionine	73
Phenylalanine	79
Proline	148
Serine	65
Threonine	188
Tryptophan	78
Tyrososine	82
Valine	183
Glucose	31
Starch	369
Cholesterol	187
Choline	114
Ecosapentanoic Acid	6428

Table S2. Target metabolites suspected of requiring host-gut metabolic interaction

Compound Name	**KEGG ID**
Short Chain Fatty Acids	
Acetate	33
Propionate	163
Butyrate	246
Bile Acids	
Cholate	695

Deoxycholate	4483
Chendeoxycholate	2528
Taurocholate	5122
Glycocholate	5465
Taurochendeoxycholate	1921
Glycodeoxycholate	5464
Taurodeoxylcholate	5463
Choline Metabolites	
Choline	114
Methylamine	218
Dimethylamine	543
Trimethylamine	565
Trimethylamine N oxide	1104
Dimethylglycine	1026
Betaine	719
Phenolic, Benzoyl, Phenyl Derivatives	
Benzoic acid	180
Hippuric acid	1586
3 Hydroxybenzoic acid	587
4 Hydroxybenzoic acid	156
3 Hydroxycinnamate	12621
4 Hydroxyphenylacetate	642
3,4 Dihydroxyphenylacetate	1161
Phenylacetylglycine	5598
Phenylacetate	15583
Indole Derivatives	
Indoleacetate	954
Melatonin	1598
Serotonin	780
Vitamins	
Vitamin K	2059
Biotin	120
Folate	504
Thiamine	378
Riboflavin	255
Pyridoxine	314
Polyamines	
Putrescine	134

Cadaverine	1672
Spermidine	315
Spermine	750
Lipids	
LPS	338
Acylglycerol	1885
Sphingomeylin	550
Cholesterol	187
Phosphatidylcholine	157
Phosphoethanolamine	346
Triglyceride	422

Table S3. KEGG compound numbers of biochemical cofactors used to terminate random walks

1	10	53	104	239	458	1344	2355
2	15	54	105	286	459	1345	2739
3	16	55	112	360	460	1346	2741
4	20	63	130	361	575	1352	3246
5	24	68	131	362	655	1367	3794
6	35	75	144	363	700	1368	5822
7	44	81	167	364	705	2353	
8	51	87	206	365	942	2354	

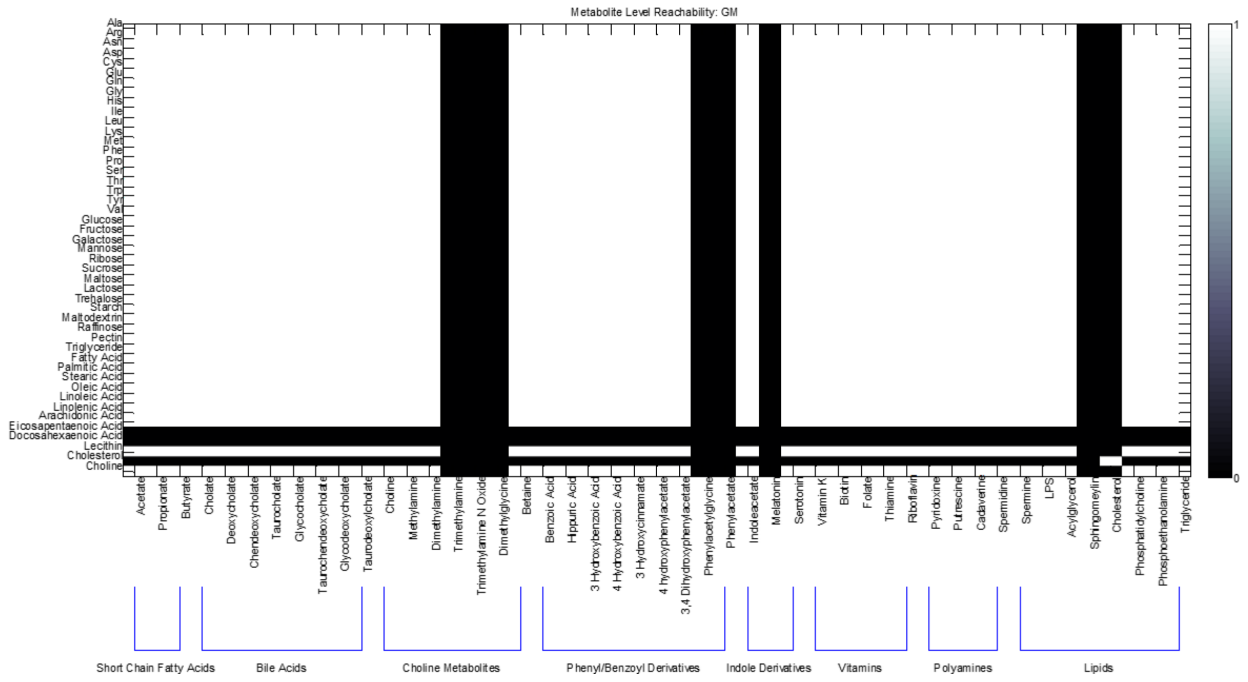


Figure S1. Color-map of the metabolite level R matrix from the metabolites in Table S1 to the metabolites in Table S2 for the GM model. Values of “1” indicate the existence of a path connecting a metabolite pair, while values of “0” indicates the absence of such a path.

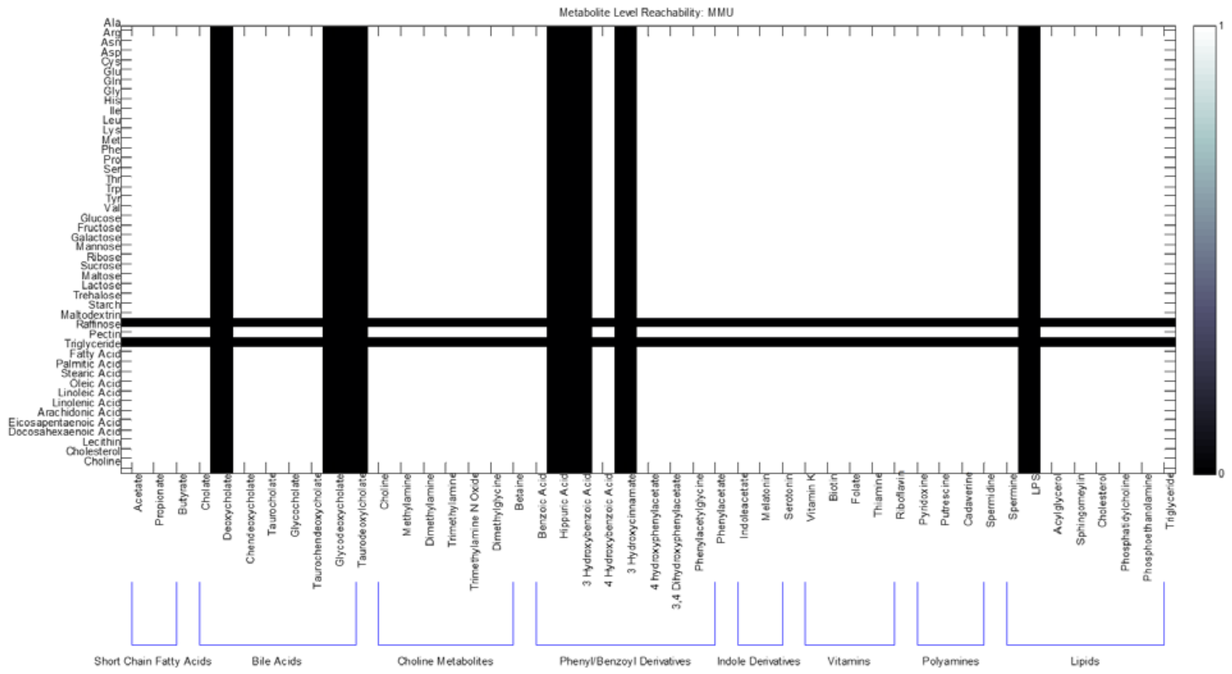


Figure S2. Color-map of the metabolite level R matrix from the metabolites in Table S1 to the metabolites in Table S2 for the GM model. Values of "1" indicate the existence of a path connecting a metabolite pair, while values of "0" indicates the absence of such a path.

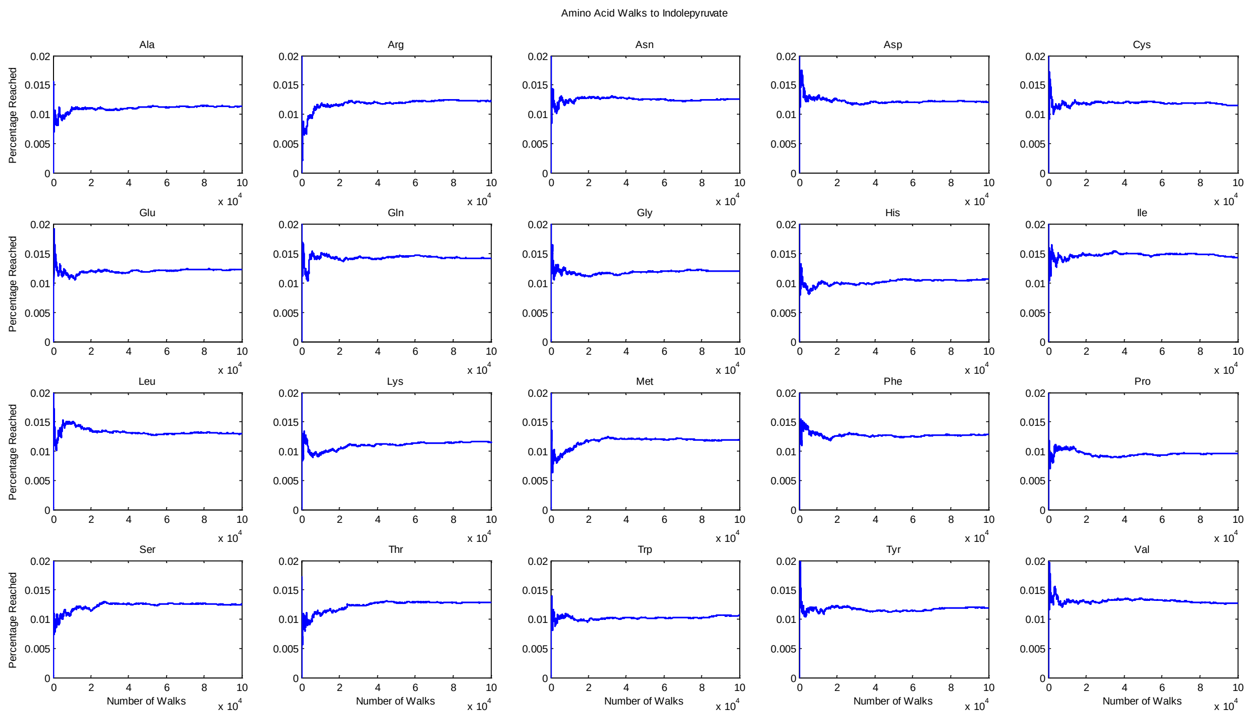


Figure S3. Plots of fraction of walks reached to indolepyruvate vs. the number of walks for all the amino acids.

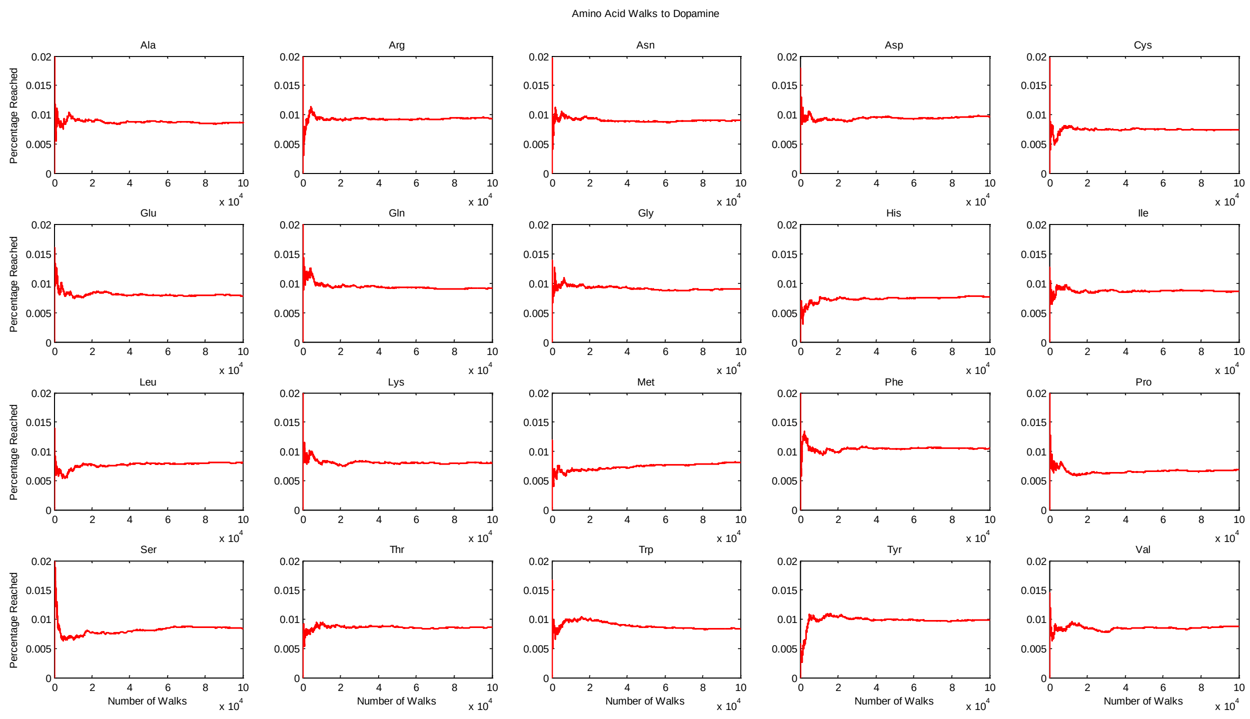


Figure S4. Plots of fraction of walks reached to indolepyruvate vs. the number of walks for all the amino acids.

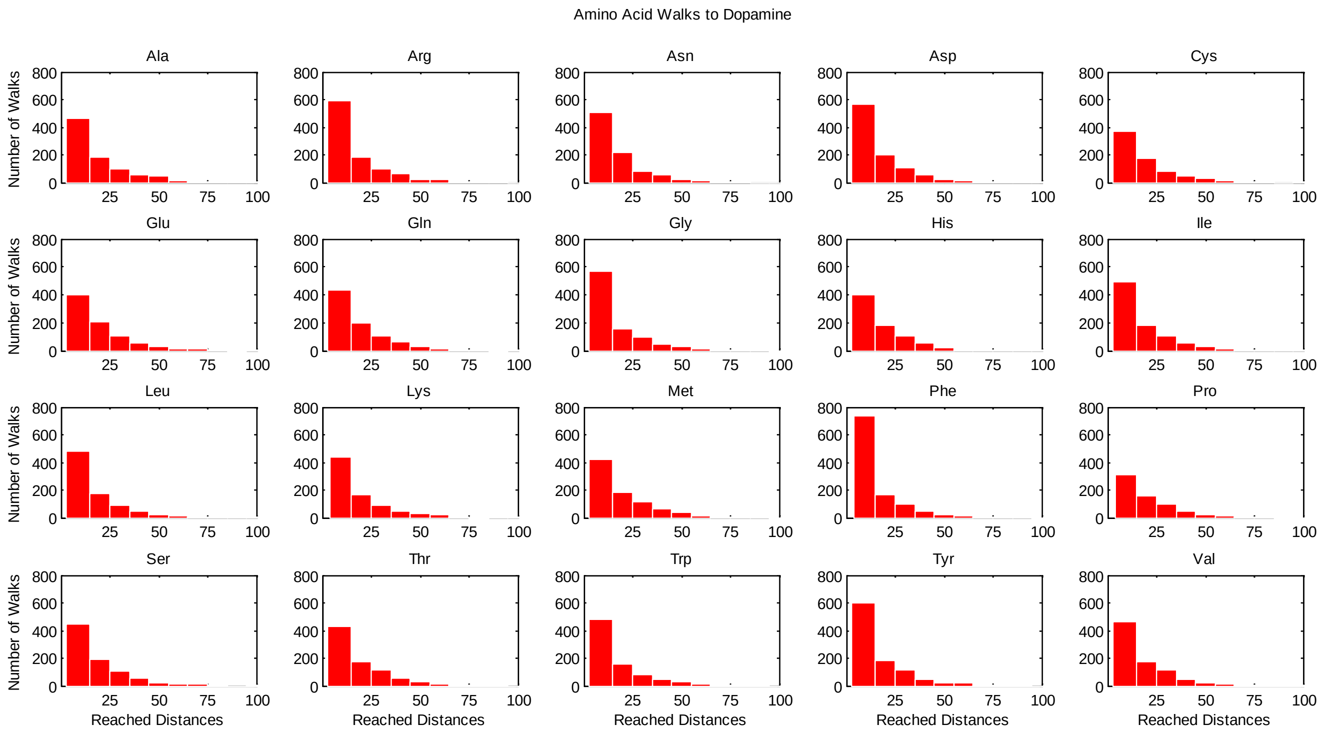


Figure S5. Histograms of path distances as defined in eq. (1) for the reached walks from the amino acids to dopamine on the GM graph. One hundred thousand walks walks were run from each amino acid.

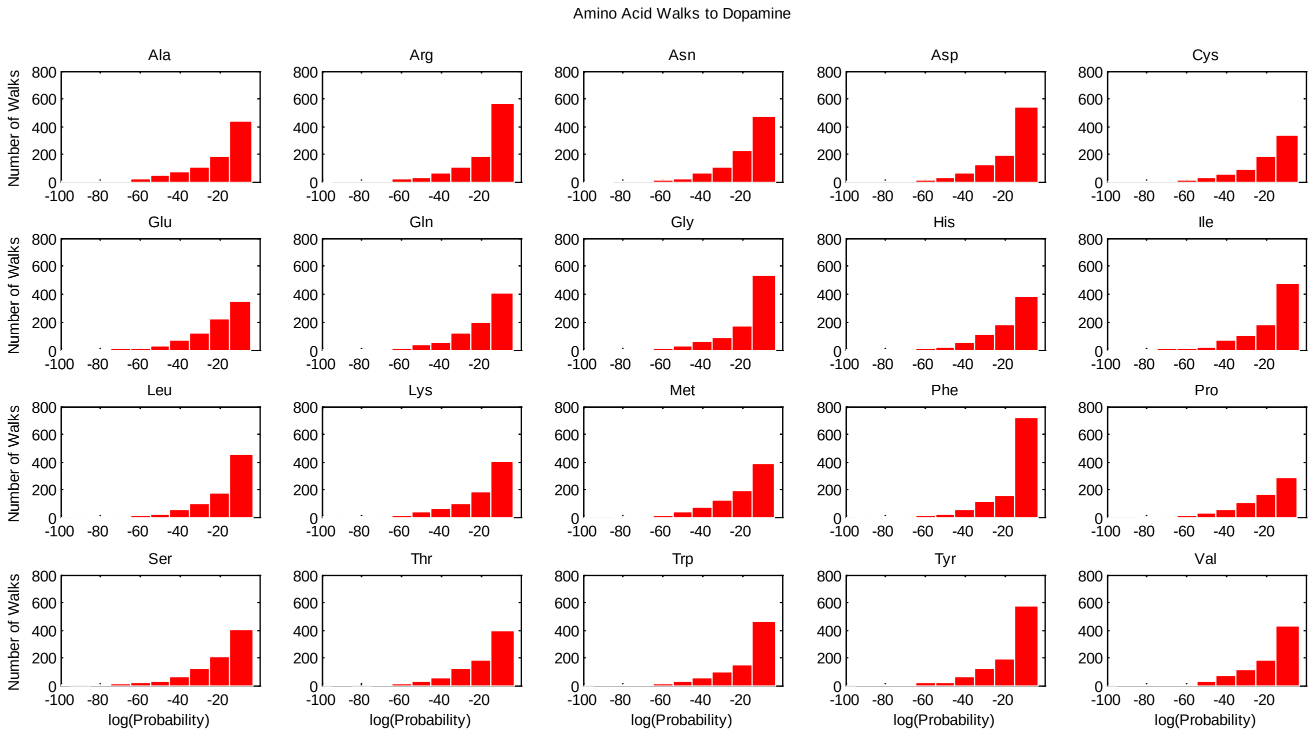


Figure S6. Histograms of the logarithm of path probabilities (eq. (2)) for the reached walks from the amino acids to dopamine on the GM graph. One hundred thousand walks were run from each amino acid.

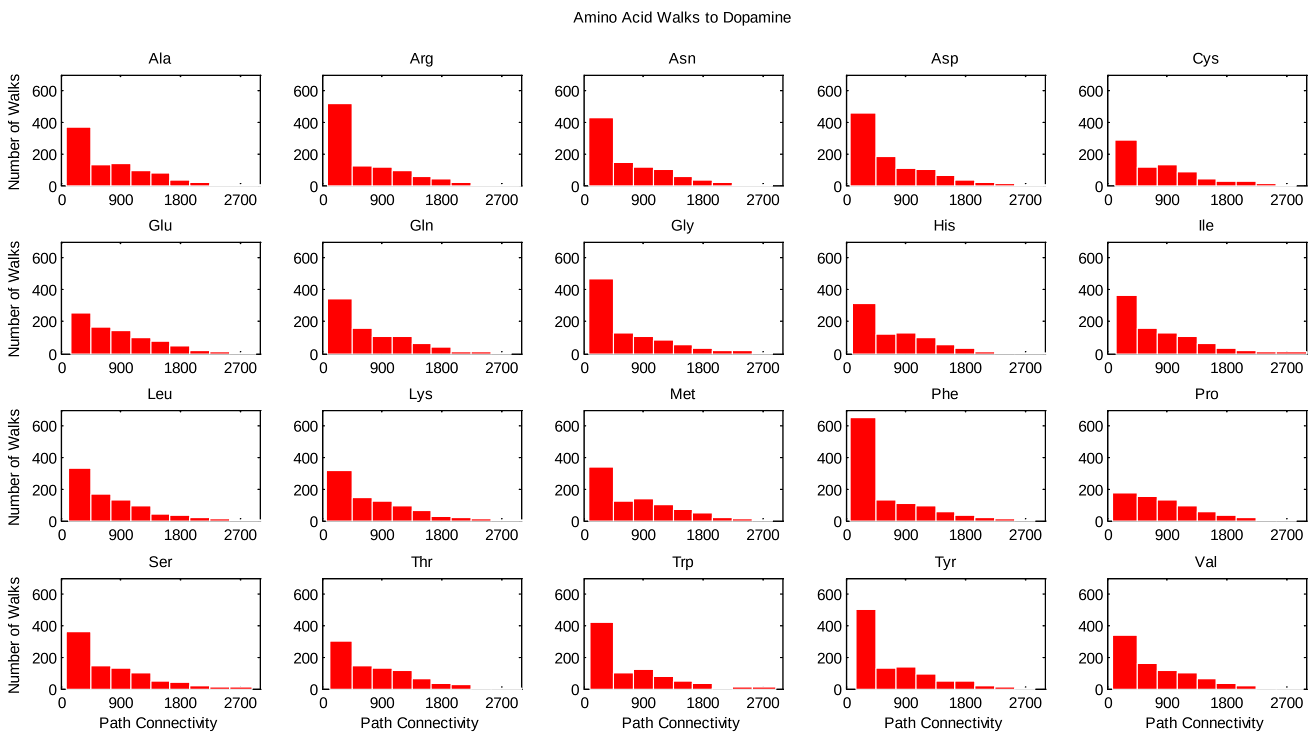


Figure S7. Path connectivity (eq. (3)) histograms for the reached walks from the amino acids to indolepyruvate on the GM graph. One hundred thousand walks were run from each amino acid.

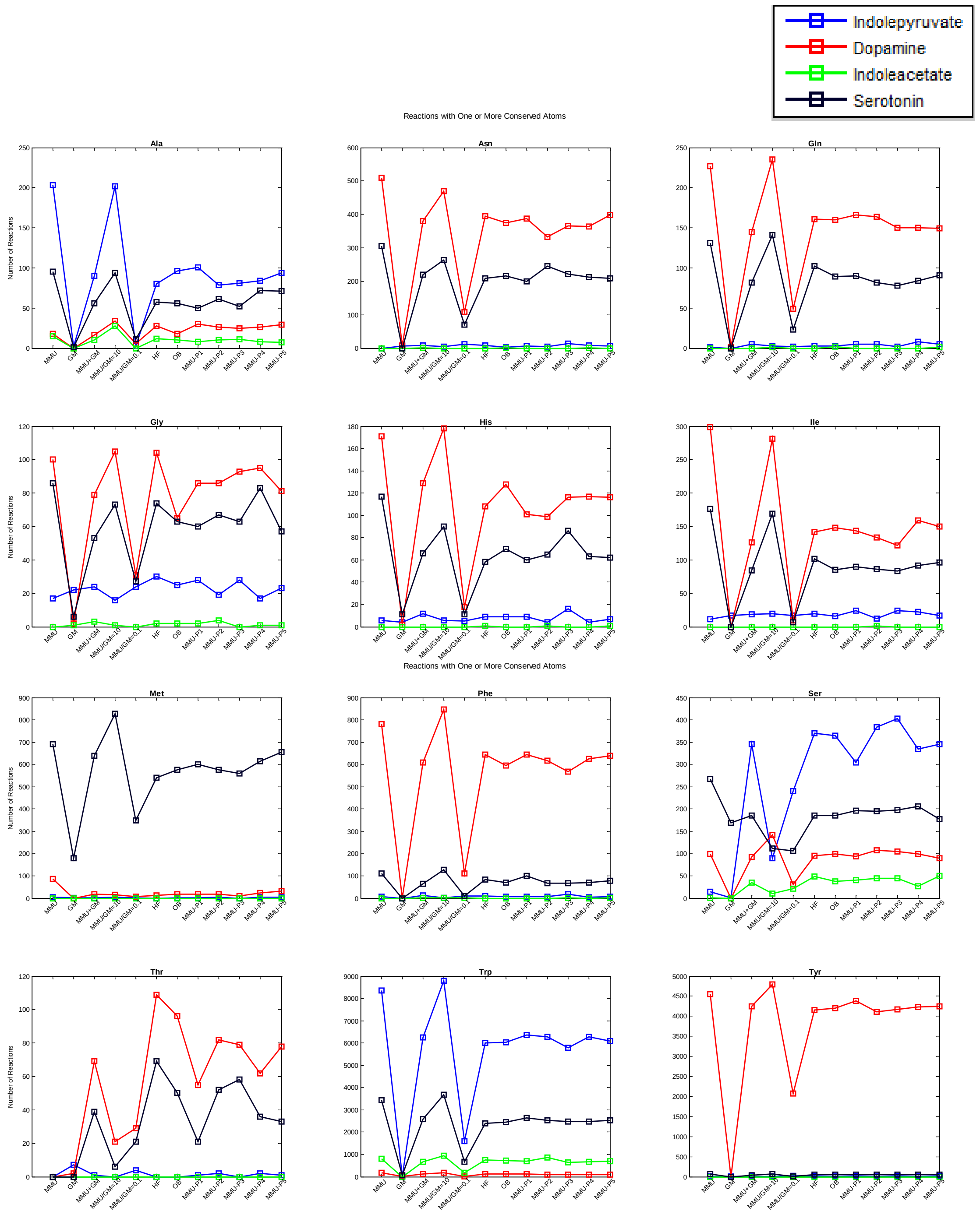


Figure S8. Plots of number of reached paths with conserved atoms for the amino acids with the most reached paths to indolepyruvate, dopamine, indoleacetate, and serotonin on all 12 models, out of 100,000 total random walks.

MATLAB Functions

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function AM = adjacency_matrix(S,reaction_weight)
% Input is a directed or undirected stoichiometrix matrix
% Creates adjacency matrix AM which contains the probability of going from
% from node i --> j
% AM is built assuming graph is undirected, i.e. AM(i,j) = AM(j,i)

AM = zeros(size(S,1));

for i = 1:size(S,2)
    for j = 1:size(S,1)
        if S(j,i) < 0
            [r,~] = find(S(:,i) > 0);
            if exist('reaction_weight','var')
                AM(j,r) = AM(j,r) + reaction_weight(i);
                AM(r,j) = AM(r,j) + reaction_weight(i);
            else
                AM(j,r) = AM(j,r) + 1;
                AM(r,j) = AM(r,j) + 1;
            end
        end
    end
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function [atom_tracing,reaction_path] =
atom_tracer(Reached_Walk,S,cmpds,rxns,num_atoms,Kegg_Reactions_Data-
base,Kegg_RPairs_Database,rxn_weights)
% atom_tracing = a row vector the length of the starting metabolite.
% Conserved atoms will have a nonzero entry corresponding to the final atom
% position at the end of the walk. Nonconserved atoms will have a zero
% entry.
% reaction_path = kegg reaction IDs of the reactions in the pathway
% Reached_Walk = a vector containing the metabolites in the path
% S = stoichiometric matrix
% rxns = vector or kegg reaction IDs corresponding to the stoichiometric
% matrix columns
% cmpds = vector of kegg compound IDs corresponding to the stoichiometrix
% matrix rows
% num_atoms = (optional)number of atoms in the beginning compound (optional
% argument)
% Kegg_Reactions_Database = (optional) Database of kegg reactions, atom
% tracer checks Kegg online if this argument is not present
% Kegg_RPairs_Database = (optional) Database of kegg rpairs, atom tracer
% checks Kegg online if this argument is not present
```

```
reaction_url = 'http://www.genome.jp/dbget-bin/www_bget?rn:R';
rpair_url = 'http://www.genome.jp/dbget-bin/www_bget?rp:';
starting_atom_url = 'http://www.genome.jp/dbget-bin/www_bget?cpd:';

% determine the number of atoms in the starting metabolite if its not
% specified as an argument
if exist('num_atoms','var') == 0
    Starting_Point = cmpds(Reached_Walk(1,1),1);
    first_metabolite = sprintf('C%05d',Starting_Point);
    first_metabolite_page = urlread(strcat(starting_atom_url,first_metabo-
lite));
    ATOM_index = strfind(first_metabolite_page,'ATOM');
    BOND_index = strfind(first_metabolite_page,'BOND');
    num_atoms = regexp(first_metabolite_page(ATOM_index:BOND_index),
'\d+','match');
    atom_tracing = 1:str2double(num_atoms{1});
else
    atom_tracing = 1:num_atoms;
end
Reached_Walk(Reached_Walk == 0) = [];
reaction_path = zeros(1,size(Reached_Walk,2)-1);

for m = 2:size(Reached_Walk,2)

    % The walk data contains the nodes of the walk; the reactin between the
    % nodes must be determined. If more than one possibility exists, a
    % reaction is chosen at random.

    [~,c1] = find(S(Reached_Walk(m-1),:) ~= 0);
    [~,c2] = find(S(Reached_Walk(m),:) ~= 0);
    [~,c3] = find(ismember(c1,c2) == 1);

    if exist('rxn_weights','var') && length(c3) > 1
        c3_order = datasample(c3,nnz(rxn_weights(c1(c3))),'re-
place',false,'weights',rxn_weights(c1(c3)));
        possible_reactions_order = zeros(length(c3),1);
        for i = 1:length(c3_order)
            ind = find(c3 == c3_order(i));
            possible_reactions_order = ind;
        end
    else
        possible_reactions_order = randperm(length(c3));
    end

    % This step ensures that in the reaction chosen one node is a reactant
    % and the other a product. This must be done because we assumed all
    % reactions in the S matrix to be reversible, thus when the adjacency
    % matrix was created it could not differentiate between reactants and
    % products.

    if exist('Kegg_Reactions_Database','var')
        for n = 1:length(possible_reactions_order)
            if S(Reached_Walk(m-1),c1(c3(possible_reactions_or-
der(n))))*S(Reached_Walk(m),c1(c3(possible_reactions_order(n)))) < 0
```

```
                kegg_rxn_id = rxns(c1(c3(possible_reactions_order(n))));
                reaction_path(1,m-1) = rxns(c1(c3(possible_reactions_or-
der(n))));
                break
            end
        end
        if n == length(possible_reactions_order) && S(Reached_Walk(m-
1),c1(c3(possible_reactions_order(n))))*S(Reached_Walk(m),c1(c3(possible_re-
actions_order(n)))) > 0
            reaction_path(1,m-1) = rxns(c1(c3(possible_reactions_order(n))));
            atom_tracing = atom_tracing*0;
            break
        end

        % Once the reaction is known, find the relevant RPair
        current_metabolite = cmpds(Reached_Walk(m-1),1);
        next_metabolite = cmpds(Reached_Walk(m),1);
        reaction_rpair_data = Kegg_Reactions_Database{kegg_rxn_id,4};
        row = and(any(reaction_rpair_data == current_metabolite,2),any(reac-
tion_rpair_data == next_metabolite,2));
        rpair_id = reaction_rpair_data(row == 1);
        if isempty(rpair_id)
            atom_tracing = atom_tracing*0;
            break;
        end

    else

        % Go online

        for n = 1:length(possible_reactions_order)
            if S(Reached_Walk(m-1),c1(c3(possible_reactions_or-
der(n))))*S(Reached_Walk(m),c1(c3(possible_reactions_order(n)))) < 0
                kegg_rxn_id = sprintf('%05d',rxns(c1(c3(possible_reac-
tions_order(n)))));
                reaction_path(1,m-1) = rxns(c1(c3(possible_reactions_or-
der(n))));
                break
            end
        end
        if n == length(possible_reactions_order) && S(Reached_Walk(m-
1),c1(c3(possible_reactions_order(n))))*S(Reached_Walk(m),c1(c3(possible_re-
actions_order(n)))) > 0
            reaction_path(1,m-1) = rxns(c1(c3(possible_reactions_order(n))));
            atom_tracing = atom_tracing*0;
            break
        end

        % Once the reaction is known, find the relevant RPair

        kegg_reaction_url = strcat(reaction_url,kegg_rxn_id);
        rxn_page = urlread(kegg_reaction_url);
        current_metabolite = cmpds(Reached_Walk(m-1),1);
        current_metabolite = sprintf('C%05d',current_metabolite);
        next_metabolite = cmpds(Reached_Walk(m),1);
        next_metabolite = sprintf('C%05d',next_metabolite);
        rpair_string = strcat(current_metabolite,'_',next_metabolite);
        rpair_string = strcat('RP\d{5}.+',rpair_string); % makes the string
RP\d{5}.+Cxxxxx_Cxxxxx where "x"s are numbers
        rpair_find = cell2mat(regexp(rxn_page,rpair_string,'match')); % Looks
for the RPair. finds strings of the form RPxxxxx   Cxxxxx_Cxxxxx, ex: RP01885
C00331_C00637
        if isempty(rpair_find) % The case that the relevant reaction is the
reverse of the reaction in the page
            rpair_string = strcat(next_metabolite,'_',current_metabo-
lite); %switch the current and next metabolite order, search again
            rpair_string = strcat('RP\d{5}.+',rpair_string);
            rpair_find = cell2mat(regexp(rxn_page,rpair_string,'match'));
            rpair_index = strfind(rpair_find,'RP');
            if isempty(rpair_index) % the case that the two nodes are not di-
rectly involved in the reaction
                atom_tracing = atom_tracing*0;
                break;
            end
            rpair_index = rpair_index(end); %The case that the desired RPair
is not the first RPair listed on the reaction page
        else
            rpair_index = strfind(rpair_find,'RP');
            if isempty(rpair_index)
                atom_tracing = atom_tracing*0;
                break;
            end
            rpair_index = rpair_index(end); % The case desired RPair is not
the first RPair listed on the reaction page
        end
        if isempty(rpair_index)
            atom_tracing = atom_tracing*0;
            break;
        end
        rpair_id = rpair_find(rpair_index:rpair_index+6);
    end


    %RPair ID has been found, get RPair data

    if exist('Kegg_RPairs_Database','var')
        rpair_data = Kegg_RPairs_Database{rpair_id,1};
        [~,c] = find(rpair_data(1,:) == current_metabolite);
        current_atom_position = zeros(size(rpair_data,1)-1,1);
        next_atom_position = zeros(size(rpair_data,1)-1,1);
        if c == 1
            for p = 2:size(rpair_data,1)
                % create two vectors, one containing the current atom posi-
tion
                % of the aligned atoms, the other containing the new atom
                % positions of those atoms in the next metabolite
                current_atom_position(p-1,1) = rpair_data(p,1);
                next_atom_position(p-1,1) = rpair_data(p,2);
            end
        else
            for p = 2:size(rpair_data,1)
                current_atom_position(p-1,1) = rpair_data(p,2);
                next_atom_position(p-1,1) = rpair_data(p,1);
            end
        end
        conserved_atoms = zeros(1,length(atom_tracing));
        for q = 1:length(current_atom_position)
            [~,column]=find(atom_tracing == current_atom_position(q));
            conserved_atoms(1,column) = next_atom_position(q);
        end
        atom_tracing = conserved_atoms;
    else
        kegg_rpair_url = strcat(rpair_url,rpair_id);
        rpair_page = urlread(kegg_rpair_url);
        ALIGN_index = strfind(rpair_page,'ALIGN');
        ENTRY1_index = strfind(rpair_page,'ENTRY1');
        align_info = regexp(rpair_page(ALIGN_index:ENTRY1_in-
dex),'\r\n|\n|\r','split');
        Compound_index = strfind(rpair_page,'Compound');
        Type_index = strfind(rpair_page,'Type');
        align_metabolite_find = rpair_page(Compound_index:Type_index);
        align_metabolite_index = regexp(align_metabolite_find,'C\d{5}');
        align_metabolite = align_metabolite_find(align_metabolite_in-
dex(1):align_metabolite_index(1)+5);    % Determine which compound is being
aligned to the other
        current_atom_position = zeros(length(align_info)-3,1);
        next_atom_position = zeros(length(align_info)-3,1);
        num_aligned_atoms = regexp(char(align_info{1}),'\d+','match');
        num_aligned_atoms = str2double(cell2mat(num_aligned_atoms(1)));     %
Determine the number of aligned atoms
        if strcmp(align_metabolite,current_metabolite)  % The case that the
current metabolite in the walk is being aligned onto the next metabolite in
the walk
            for p = 1:num_aligned_atoms
                % create two vectors, one containing the current atom posi-
tion
                % of the aligned atoms, the other containing the new atom
                % positions of those atoms in the next metabolite
                align_info_p = regexp(align_info{p+1},'\d+:\w+','match');
                current_atom_position_cell = re-
gexp(align_info_p{1},'^\d+','match');
                current_atom_position(p,1) = str2double(cell2mat(cur-
rent_atom_position_cell));
                next_atom_position_cell = re-
gexp(align_info_p{2},'^\d+','match');
                next_atom_position(p,1) = str2double(cell2mat(next_atom_posi-
tion_cell));
            end
            conserved_atoms = zeros(1,length(atom_tracing));
            for q = 1:length(current_atom_position)
                [~,column]=find(atom_tracing == current_atom_position(q));
                conserved_atoms(1,column) = next_atom_position(q);
            end
            atom_tracing = conserved_atoms;
        else
            for p = 1:num_aligned_atoms % the case that the next metabolite
in the walk is being aligned onto the current metabolite
                align_info_p = regexp(align_info{p+1},'\d+:\w+','match');
                current_atom_position_cell = re-
gexp(align_info_p{2},'^\d+','match');
                current_atom_position(p,1) = str2double(cell2mat(cur-
rent_atom_position_cell));
                next_atom_position_cell = re-
gexp(align_info_p{1},'^\d+','match');
                next_atom_position(p,1) = str2double(cell2mat(next_atom_posi-
tion_cell));
            end
            conserved_atoms = zeros(1,length(atom_tracing));
            for q = 1:length(current_atom_position)
                [~,column]=find(atom_tracing == current_atom_position(q));
                conserved_atoms(1,column) = next_atom_position(q);
            end
            atom_tracing = conserved_atoms;
        end
    end
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function [rxns_MMU_GM, GM_reaction_weight,MMU_GM_reaction_weight] =
calc_rxn_weight(GM_composition,phyla_score,MMU_GM_ratio,rxns,rxns_MMU)

GM_composition = GM_composition./sum(GM_composition);
GM_reaction_weight = zeros(size(phyla_score,1),1);
for i = 1:size(phyla_score,1)
    for j = 1:size(phyla_score,2)
        GM_reaction_weight(i) = GM_reaction_weight(i) + GM_composi-
tion(j).*phyla_score(i,j);
    end
end

% Make combined MMU and GM matrix and combined MMU/GM reaction weights
common_reactions = ismember(rxns,rxns_MMU); %same size as rxns, has a 1 for
common reactions
rxns_MMU_GM = cat(1,rxns_MMU,rxns(common_reactions == 0)); %all MMU rxns fol-
lowed by unique GM rxns
GM_common_rxn_index = find(common_reactions == 1);
common_reaction_ids = rxns(common_reactions == 1);
MMU_reaction_weight = ones(size(rxns_MMU,1),1)*MMU_GM_ratio;
reaction_weight = cat(1,MMU_reaction_weight,GM_reaction_weight(common_reac-
tions == 0));

for i = 1:length(common_reaction_ids)
    MMU_common_rxn_index = find(rxns_MMU == common_reaction_ids(i));
    reaction_weight(MMU_common_rxn_index) = reaction_weight(MMU_com-
mon_rxn_index) + GM_reaction_weight(GM_common_rxn_index(i));
end
reaction_weight = reaction_weight./(1 + MMU_GM_ratio);

[rxns_MMU_GM,ind] = sort(rxns_MMU_GM);
```

```
MMU_GM_reaction_weight = zeros(length(ind),1);
for i = 1:length(ind)
    MMU_GM_reaction_weight(i) = reaction_weight(ind(i));
end
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function [SM,U,R] = comm_class(AM,S,cmpds,rxns,rxn_weights)
% Find the communication classes that make up the S matrix
% Create new S matrices with compounds and reaction lists for each
% communication classes
% The SM cell array contains the following information for each sub S
% matrix created
% SM{:,1} = S matrix
% SM{:,2} = cmpds
% SM{:,3} = rxns
% SM{:,4} = rxn_weight_adjusted
% U = communication classes
% R = reachability matrix

if issparse(AM) == 0
    AM = sparse(AM);
end

[nc,C] = graphconncomp(AM);

R = zeros(length(C));
for i =1:nc
    ind = find(C == i);
    R(ind,ind) = 1;
end

C = R & R';
U = unique(C,'rows');

SM = cell(size(U,1),4);
for i = 1:size(U,1)
    met_ind = find(U(i,:) == 1);
    [~,rxn_ind] = find(S(met_ind,:) ~= 0);
    SM{i,1} = S(met_ind,unique(rxn_ind));
    SM{i,2} = cmpds(met_ind);
    SM{i,3} = rxns(unique(rxn_ind));
    if exist('rxn_weights','var')
        rxn_weight_adjusted = zeros(size(SM{i,3},1),1);
        for j = 1:size(rxn_weight_adjusted,1)
            rxn_weight_adjusted(j) = rxn_weights(SM{i,3}(j) == rxns);
        end
        SM{i,4} = rxn_weight_adjusted;
    end
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function Kegg_Compounds_Database = kegg_cmpds_txt2cell(com-
pounds_txt_file,tot_num_cmpds)
% Creates an array whose indices are the Kegg compound ID numbers and the
% value of each entry corresponds to the number of atoms in that molecule.

clear fid
fid = fopen(compounds_txt_file);
line = fgetl(fid);
Kegg_Compounds_Database = zeros(tot_num_cmpds,1);
counter = 0;
while ischar(line)
    if length(line) < 4
        line = fgetl(fid);
        continue
    end
    counter = counter + 1;
    if isempty(regexp(line,'ENTRY.+C\d{5}','ONCE')) == 0;
        cmpd_id_index = regexp(line,'C\d{5}');
        cmpd_id = str2double(line(cmpd_id_index+1:cmpd_id_index+6));
        line = fgetl(fid);
        continue
    end
    if strcmp(line(1:4),'ATOM')
        num_atoms = regexp(line,'\d+','match');
        num_atoms = str2double(cell2mat(num_atoms(1)));
        Kegg_Compounds_Database(cmpd_id) = num_atoms;
        line = fgetl(fid);
        continue
    end
    line = fgetl(fid);
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function Kegg_RPairs_Database =
kegg_rpairs_txt2cell(rpairs_txt_file,tot_num_rpairs)
% Creates a cell array that contains the Kegg RPair alignment data
% Format of cell array:
%   - row number corresponds to RPAIR ID number
%   - each enter is a matrix; the first row contains the metabolites
%   involved in the RPAIR. The remaining rows contain the atom alignment
%   data
% rpairs_txt_file is the string name of the text file containing the rpair
% database

clear fid
fid = fopen(rpairs_txt_file);
line = fgetl(fid);
Kegg_RPairs_Database = cell(tot_num_rpairs,1);
```

```
while ischar(line)
    if length(line) < 5
        line = fgetl(fid);
        continue
    end
    if isempty(regexp(line,'ENTRY.+RP\d{5}','ONCE')) == 0;
        rpair_id_index = regexp(line,'RP\d{5}');
        rpair_id = str2double(line(rpair_id_index+2:rpair_id_index+6));
        line = fgetl(fid);
        metabolites = strsplit(char(re-
gexp(line,'C\d{5}_C\d{5}','match')),'_');
        x = metabolites{1};
        y = metabolites{2};
        x(1) = [];
        y(1) = [];
        metabolite1 = str2double(x);
        metabolite2 = str2double(y);
    end
    if strcmp(line(1:5),'ALIGN')
        num_aligned_atoms = regexp(line,'\d+','match');
        num_aligned_atoms = str2double(cell2mat(num_aligned_atoms(1)));
        RPair = zeros(num_aligned_atoms+1,2);
        RPair(1,1) = metabolite1;
        RPair(1,2) = metabolite2;
        for i = 1:num_aligned_atoms
            line = fgetl(fid);
            align_info = regexp(line,'\d+:\w+','match');
            RPair(i+1,1) = str2double(cell2mat(re-
gexp(align_info{1},'^\d+','match')));
            RPair(i+1,2) = str2double(cell2mat(re-
gexp(align_info{2},'^\d+','match')));
        end
        Kegg_RPairs_Database{rpair_id,1} = RPair;
    end
    line = fgetl(fid);
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function Kegg_Reactions_Database = kegg_rxns_txt2cell(rxns_txt_file)
% Creates Cell Array of Kegg Reactions Database.
% Columns Descriptions:
% 1 - Kegg reaction number
% 2 - Reactants
%       Column 1 - reactant stoichiometry
%       Column 2 - reactant number
% 3 - Products
%       Column 1 - reactant stoichiometry
%       Column 2 - reactant number
% 4 - RPairs
%       Column 1 - RPair number
%       Column 2 - RPair reactant
%       Column 3 - RPair product
%
```

```
% rxns_txt_file is the string name


kegg_reactions = importdata(rxns_txt_file);

ENTRY = strfind(kegg_reactions,'ENTRY');
EQUATION = strfind(kegg_reactions,'EQUATION');
ENTRY_indices = find(~cellfun('isempty',ENTRY));
EQUATION_indices = find(~cellfun('isempty',EQUATION));
SLASH = strfind(kegg_reactions,'///');
SLASH_indices = find(~cellfun('isempty',SLASH));

line = cell2mat(kegg_reactions(ENTRY_indices(end)));
max_rxn_num_index = regexp(line,'R\d{5}');
max_rxn_num = str2double(line(max_rxn_num_index+1:max_rxn_num_index+5));

Kegg_Reactions_Database = cell(max_rxn_num,4);

for i = 1:length(ENTRY_indices)
    % find reaction number
    line = cell2mat(kegg_reactions(ENTRY_indices(i)));
    rxn_num_index = regexp(line,'R\d{5}');
    rxn_num = str2double(line(rxn_num_index+1:rxn_num_index+5));
    Kegg_Reactions_Database{rxn_num,1} = rxn_num;
    rxn_line = cell2mat(kegg_reactions(EQUATION_indices(i)));

    % find reactants
    reactants_match = strtrim(strsplit(char(re-
gexp(rxn_line,'N.+<','match')),'+'));
    for j = 1:length(reactants_match)
        reactant_info = regexp(reactants_match{j},'\d+','match');
        if isempty(reactant_info)
            reactants_match{j} = [];
        end
    end
    reactants_match = reactants_match(~cellfun('isempty',reactants_match));
    reactants = zeros(length(reactants_match),2);
    for j = 1:length(reactants_match)
        reactant_info = regexp(reactants_match{j},'\d+','match');
        if length(reactant_info) == 1
            reactants(j,1) = 1;
            reactants(j,2) = str2double(reactant_info(1));
            else
            reactants(j,1) = str2double(reactant_info(1));
            reactants(j,2) = str2double(reactant_info(2));
        end
    end
    Kegg_Reactions_Database{rxn_num,2} = reactants;

    % find products
    products_match = strtrim(strsplit(char(re-
gexp(rxn_line,'>.+','match')),'+'));
    for j = 1:length(products_match)
        product_info = regexp(products_match{j},'\d+','match');
        if isempty(product_info)
```

```
            products_match{j} = [];
        end
    end
    products_match = products_match(~cellfun('isempty',products_match));
    products = zeros(length(products_match),2);
    for j = 1:length(products_match)
        product_info = regexp(products_match{j},'\d+','match');
        if length(product_info) == 1
            products(j,1) = 1;
            products(j,2) = str2double(product_info(1));
        else
            products(j,1) = str2double(product_info(1));
            products(j,2) = str2double(product_info(2));
        end
    end
    Kegg_Reactions_Database{rxn_num,3} = products;

    % find RPairs
    RPAIR = strfind(kegg_reactions(EQUATION_indices(i):SLASH_indi-
ces(i)),'RPAIR');
    if isempty(cell2mat(RPAIR))
        continue
    else
        RPAIR_index = find(~cellfun('isempty',RPAIR));
        ENZYME = strfind(kegg_reactions(EQUATION_indices(i):SLASH_indi-
ces(i)),'ENZYME');
        ENZYME_index = find(~cellfun('isempty',ENZYME));
        RPairs = zeros(ENZYME_index - RPAIR_index,3);
        for j = 1:size(RPairs,1)
            % find RPair number
            RPair_line = cell2mat(kegg_reactions(EQUATION_indices(i) +
RPAIR_index + j-2));
            rpair_id_index = regexp(RPair_line,'RP\d{5}');
            rpair_id = str2double(RPair_line(rpair_id_index+2:rpair_id_in-
dex+7));
            RPairs(j,1) = rpair_id;

            % find RPair metabolites
            rpair_metabolite_index = regexp(RPair_line,'C\d{5}');
            RPairs(j,2) = str2double(RPair_line(rpair_metabolite_in-
dex(1)+1:rpair_metabolite_index(1) + 5));
            RPairs(j,3) = str2double(RPair_line(rpair_metabolite_in-
dex(2)+1:rpair_metabolite_index(2) + 5));
        end
        Kegg_Reactions_Database{rxn_num,4} = RPairs;
    end
end
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function [T,cmpds_T] = Make_Atom_TM(cmpds,rxns,Kegg_Compounds_Data-
base,Kegg_Reactions_Database,Kegg_RPairs_Database,rxn_weights)
% Creates atomic level transition matrix
% T = transition matrix
% cmpds_T = compounds corresponding to the rows of T


tot_num_atoms = 0;
for i = 1:size(cmpds,1)
    num_atoms = Kegg_Compounds_Database(cmpds(i));
    if num_atoms == 0;
        num_atoms = 1;
    end
    tot_num_atoms = tot_num_atoms + num_atoms;
end

cmpds_T = zeros(tot_num_atoms,2);

for i = 1:size(cmpds,1)
    num_atoms = Kegg_Compounds_Database(cmpds(i));
    if num_atoms == 0
        num_atoms = 1;
    end
    [r,~] = find(cmpds_T(:,1) == 0);
    cmpds_T(r(1):r(1) + num_atoms - 1,1) = cmpds(i);
    cmpds_T(r(1):r(1) + num_atoms - 1,2) = (1:num_atoms)';
end

Row = zeros(30*tot_num_atoms,1);
Column = zeros(30*tot_num_atoms,1);
Value = zeros(30*tot_num_atoms,1);

for i = 1:size(rxns,1)
    rpair_info = Kegg_Reactions_Database{rxns(i),4};
    reactants = Kegg_Reactions_Database{rxns(i),2}(:,2);
    products = Kegg_Reactions_Database{rxns(i),3}(:,2);
    % reactions that do not have rpair data
    if isempty(rpair_info)
        for j = 1:length(reactants)
            [r,~] = find(cmpds_T(:,1) == reactants(j));
            next_spot = find(Row == 0);
            Row(next_spot(1):next_spot(1)+length(products)-1) = r(1);
            Column(next_spot(1):next_spot(1)+ length(products)-1) = products;
            if exist('rxn_weights','var')
                Value(next_spot(1):next_spot(1) + length(products)-1) =
rxn_weights(i);
            else
                Value(next_spot(1):next_spot(1) + length(products)-1) = 1;
            end
        end
    else
        % reactions that have rpair data
        % metabolites in the reaction that do not have rpair data
        no_rpair_reactants = reactants(ismember(reactants,rpair_info(:,2:3))
== 0);
        no_rpair_products = products(ismember(products,rpair_info(:,2:3)) ==
0);
        r_products = zeros(length(no_rpair_products),1);
        for j = 1:length(no_rpair_products)
            [r,~] = find(cmpds_T == no_rpair_products(j));
            r_products(j) = r(1);
        end
        for j = 1:length(no_rpair_reactants)
            [r_reactant,~] = find(cmpds_T(:,1) == no_rpair_reactants(j));
            next_spot = find(Row == 0);
            Row(next_spot(1):next_spot(1)+length(no_rpair_products)-1) = 
r_reactant(1);
            Column(next_spot(1):next_spot(1)+length(no_rpair_products)-1) = 
r_products;
            if exist('rxn_weights','var')
                Value(next_spot(1):next_spot(1) + length(no_rpair_products)-
1) = rxn_weights(i);
            else
                Value(next_spot(1):next_spot(1) + length(no_rpair_products)-
1) = 1;
            end
        end
        % metabolites in the reaction that do have rpair data
        for j = 1:size(rpair_info,1)
            rp = Kegg_RPairs_Database{rpair_info(j,1)};
            next_spot = find(Row == 0);
            counter = next_spot(1);
            [r_reactant,~] = find(cmpds_T(:,1) == rp(1,1));
            [r_product,~] = find(cmpds_T(:,1) == rp(1,2));
            for k = 2:size(rp,1)
                [r,~] = find(cmpds_T(r_reactant(1):r_reactant(end),2) == 
rp(k,1));
                % this if statement is included because some compounds in
                % the model are generic compounds (e.g. generic amine)
                % which don't have an atom number listed, but they still
                % have RPairs in reactions
                if  isempty(r) && cmpds_T(r_reactant,2) == 1
                    num_atoms = max(rp(2:end,1));
                    cmpds_T = [cmpds_T(1:r_reactant,1) cmpds_T(1:r_react-
nat,2); ones(num_atoms-1,1)*rp(1,1) (2:num_atoms)'; cmpds_T(r_reac-
tant+1:end,1) cmpds_T(r_reactant+1:end,2)];
                    [r_reactant,~] = find(cmpds_T(:,1) == rp(1,1));
                    [r,~] = find(cmpds_T(r_reactant(1):r_reactant(end),2) == 
rp(k,1));
                end
                Row(counter) = r_reactant(r);
                [r,~] = find(cmpds_T(r_product(1):r_product(end),2) == 
rp(k,2));

                if isempty(r) && cmpds_T(r_product,2) == 1
                    num_atoms = max(rp(2:end,2));
                    cmpds_T = [cmpds_T(1:r_product,1) cmpds_T(1:r_product,2); 
ones(num_atoms-1,1)*rp(1,2) (2:num_atoms)'; cmpds_T(r_product+1:end,1) 
cmpds_T(r_product+1:end,2)];
                    [r_product,~] = find(cmpds_T(:,1) == rp(1,2));
                    [r,~] = find(cmpds_T(r_product(1):r_product(end),2) == 
rp(k,2));
                end
                Column(counter) = r_product(r);
                if exist('rxn_weights','var')
                    Value(counter) = rxn_weights(i);
                else
                    Value(counter) = 1;
                end
                counter = counter + 1;
            end
        end
    end
end

ind = find(Value == 0);
Row(ind) = [];
Column(ind) = [];
Value(ind) = [];

T = sparse(Row,Column,Value,length(cmpds_T),length(cmpds_T));

T = T + T';
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function [S,cmpds] = Make_S_Matrix(reaction_data,tot_num_kegg_cmpds,Kegg_Re-
actions_Database)
% Make stoichiometric matrix - each row is a metabolite, each column is a
% reaction
% S = stoichiometrix matrix
% cmpds = list of metabolite IDs in the S matrix
% reaction_data = a list of reaction IDs to be included in this S matrix
% tot_num_kegg_rxns = the total number of reactions in the Kegg database
% Kegg_Reactions_Database = database of Kegg reactions

S = zeros(tot_num_kegg_cmpds,length(reaction_data));

for i = 1:size(reaction_data,1)
    reactants = Kegg_Reactions_Database{reaction_data(i,1),2};
    products = Kegg_Reactions_Database{reaction_data(i,1),3};
    for j = 1:size(reactants,1)
        S(reactants(j,2),i) = -reactants(j,1);
    end
    for j = 1:size(products,1)
        S(products(j,2),i) = products(j,1);
    end
end

r = any(S,2);
[cmpds,~] = find(r == 1);
[empty_rows,~] = find(r == 0);
S(empty_rows,:) = [];
```

```
% CREATED BY BASSEL GHADDAR
% LAST EDITED 5/6/15

function [paths,steps] = random_walks(IM,CL,A,B,num_walks,maxSteps)
% S = stoichiometric matrix
% IM = incidence matrix
% CL = cofactors list, or list of nodes at which to terminate walks
% A = beginning node
% B = end node
% maxSteps: maximum # of steps to take on the walk.
% iterate the walk until a deadend is reached, or maxSteps exceeded, or B
% is reached.
% terminate a walk if a cofactor is reached

paths = zeros(num_walks,maxSteps+1);
paths(:,1) = A;
steps = zeros(num_walks,1);

for i = 1:num_walks
    BNotReached = 0;
    currentNode = A;
    while (steps(i) < maxSteps && ~BNotReached)
       % decide where to go:
       possibleDestinations = find(IM(currentNode,:)> 0);
       if (possibleDestinations <=0)
           % reached an external metaoblite;
           break;
       end
       next_dest = datasample(possibleDestinations,length(possibleDestina-
tions),'replace',false,'weights',IM(currentNode,possibleDestinations));
       for j = 1:length(possibleDestinations)
           currentNode = next_dest(j);
           if any(paths(i,:) == currentNode) || any(CL == currentNode)
               continue
           else
               break
           end
       end
       if any(paths(i,:) == currentNode)
           break
       else
           BNotReached = B == currentNode;
           steps(i) = steps(i)+1;
           paths(i,steps(i)+1) = currentNode;
       end
    end
end
end
```