
Access from the University of Nottingham repository:

Copyright and reuse:

The Nottingham ePrints service makes this work by researchers of the University of Nottingham available open access under the following conditions.

This article is made available under the University of Nottingham End User licence and may be reused according to the conditions of the licence. For more details see:
http://eprints.nottingham.ac.uk/end_user_agreement.pdf

For more information, please contact eprints@nottingham.ac.uk
Services for Biological Network Feature Detection

Venkata Lakshmi Sirisha Gollapudi

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

June 2010
To Amma, Dad and Surekha
Abstract

The complex environment of a living cell contains many molecules interacting in a variety of ways. Examples include the physical interaction between two proteins, or the biochemical interaction between an enzyme and its substrate. A challenge of systems biology is to understand the network of interactions between biological molecules, derived experimentally or computationally. Sophisticated dynamic modelling approaches provide detailed knowledge about single processes or individual pathways. However such methods are far less tractable for holistic cellular models, which are instead represented at the level of network topology.

Current network analysis packages tend to be standalone desktop tools which rely on local resources and whose operations are not easily integrated with other software and databases. A key contribution of this thesis is an extensible toolkit of biological network construction and analysis operations, developed as web services. Web services are a distributed technology that enable machine-to-machine interaction over a network, and promote interoperability by allowing tools deployed on heterogeneous systems to interface. A conceptual framework has been created, which is realised practically through the proposal of a common graph format to standardise network data, and the investigation of open-source deployment technologies. Workflows are a graph of web services, allowing analyses to be carried out as part of a bigger software pipeline. They may be constructed using web services within the toolkit together with those from other providers, and can be saved, shared and reused, allowing biologists to construct their own complex queries over various tools and datasets, or execute pre-constructed workflows designed by expert bioinformaticians.

Biologically relevant results have been produced as a result of this approach. One very interesting hypothesis has been generated regarding the regulation of yeast glycolysis by a protein found to interact with seven glycolytic enzymes. This has implied a potentially novel regulatory mechanism whereby the protein in question binds these enzymes to form an ‘energy production unit’. Also of interest are workflows which identify termini (system inputs and outputs), and cycles, which are crucial for acquiring a physiological perspective on network behaviour.
Acknowledgements

I would like to express my sincerest gratitude to my supervisors, Professor Charlie Hodgman and Professor Chris Greenhalgh, for their encouragement, support and expert advice throughout my PhD.

Thanks also to everyone at MyCIB, in particular Mr John Veasey, Dr Tom Gallagher and Dr Michael Wilson for invaluable infrastructure support, and Mrs Elsie Fountain for her excellent administrative support and a shoulder to lean on.

Special thanks are due to Alex Marshall and Daniel Zadik, for making the last four years highly entertaining as well as educational.

I could not have reached the final stages of this work without the support of my fantastic family: my loving parents, my sister Surekha, my parents-in-law Pam and John and sisters-in-law Ally and Sam, and their families. I must also express my heartfelt thanks to Vicki Thurston, Sarah Capell and Daniel Snowdon, who are the best friends I could possibly ask for.

Lastly, thanks are most definitely due to my husband Chris for his belief in me, his seemingly never-ending supply of patience, and his ability to always help me to look on the bright side.
# Contents

1 Introduction .................................................. 1  
   1.1 The aims of this research ................................ 2  
   1.2 Key contributions ...................................... 2  
   1.3 Thesis structure ....................................... 3  

2 Background .................................................. 5  
   2.1 Biological networks .................................... 5  
      2.1.1 Complex networks and graph theory ............... 6  
      2.1.2 Types of biological network ...................... 7  
      2.1.3 Limitations on analyses ......................... 11  
   2.2 Existing network analysis software ................. 13  
      2.2.1 Standalone software .......................... 14  
      2.2.2 Web-based and other client-server tools ....... 18  
      2.2.3 Web services .................................. 19  
      2.2.4 Programming libraries .......................... 19  
   2.3 Distributed computing solutions in bioinformatics .. 20  
   2.4 Conclusion ........................................... 23  

3 A Framework for Network Analysis Software .......... 24  
   3.1 Conceptual framework .................................. 24  
      3.1.1 General conceptual framework ................. 24  
      3.1.2 Network-specific conceptual framework ....... 25  
   3.2 Technical framework for tasks ...................... 30  
      3.2.1 Web service protocols and standards .......... 33  
   3.3 Technical framework for workflows ................. 36  
   3.4 Conclusion ........................................... 40  

4 Technology Evaluation .................................... 45  
   4.1 Introduction .......................................... 45  
   4.2 Criteria and test cases for evaluation ............ 45  
   4.3 Evaluation ........................................... 46  
      4.3.1 SOAP::Lite .................................... 46  
      4.3.2 Apache Axis .................................... 49
## 5 Web Services Developed

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Web service design</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>Table of web services</td>
<td>71</td>
</tr>
<tr>
<td>5.4</td>
<td>Description of web services</td>
<td>73</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Group: analyse_directed</td>
<td>73</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Group: analyse_misc</td>
<td>81</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Group: analyse_undirected</td>
<td>82</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Group: format_output</td>
<td>92</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Group: retrieve</td>
<td>95</td>
</tr>
<tr>
<td>5.4.6</td>
<td>Group: transform</td>
<td>95</td>
</tr>
</tbody>
</table>

## 6 Biological Observations

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Holistic network analysis</td>
<td>99</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Motivation</td>
<td>99</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Workflow description</td>
<td>99</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Table of workflow processors</td>
<td>99</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Input</td>
<td>100</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Output interpretation</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>Cycle identification</td>
<td>102</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Motivation</td>
<td>102</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Workflow description</td>
<td>103</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Table of workflow processors</td>
<td>104</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Input</td>
<td>104</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Output interpretation</td>
<td>105</td>
</tr>
<tr>
<td>6.3</td>
<td>Local networks</td>
<td>105</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Motivation</td>
<td>105</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Workflow description</td>
<td>106</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Table of workflow processors</td>
<td>111</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Input</td>
<td>112</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Output interpretation</td>
<td>113</td>
</tr>
<tr>
<td>6.4</td>
<td>Source and sink metabolites in a network model of metabolism</td>
<td>117</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Motivation</td>
<td>117</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Workflow description</td>
<td>117</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Table of workflow processors</td>
<td>117</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Input</td>
<td>117</td>
</tr>
<tr>
<td>6.4.5</td>
<td>Output interpretation</td>
<td>118</td>
</tr>
<tr>
<td>6.5</td>
<td>Annotating metabolic pathways with PPIs</td>
<td>120</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Motivation</td>
<td>120</td>
</tr>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.2</td>
<td>Workflow description</td>
<td>120</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Table of workflow processors</td>
<td>121</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Input</td>
<td>124</td>
</tr>
<tr>
<td>6.5.5</td>
<td>Output interpretation</td>
<td>124</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusion</td>
<td>126</td>
</tr>
</tbody>
</table>

### 7 Conclusions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Limitations</td>
<td>134</td>
</tr>
<tr>
<td>7.2</td>
<td>Future Work</td>
<td>136</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Distributed workflows</td>
<td>136</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Mixed networks</td>
<td>137</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Extension of the common graph format</td>
<td>137</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Increasing exposure</td>
<td>138</td>
</tr>
<tr>
<td>7.3</td>
<td>Summary</td>
<td>138</td>
</tr>
</tbody>
</table>

References | 139

Appendices | 151

A Taverna tutorial | 152

B Technology Evaluation examples | 166

C Tutorial examples for web services developed | 177

D Detailed workflow outputs | 231

E Cyclic cores | 243
Chapter 1

Introduction

In the post-genomic era, biologists have increasingly turned to computational methods for the management, analysis and dissemination of vast quantities of experimental data. The field of network biology, which seeks to understand biological function through the reactions and interactions between molecules, is one source of such data, and is the focus of this work. Biological networks such as metabolic, gene regulatory, signalling and protein-protein interaction may be assembled using the results of wet-lab and dry-lab \textit{(in silico)} experimentation (d’Alché-Buc and Schachter, 2005). Such networks may be modelled and analysed using a \textit{graphical} data structure, where a graph consists of a set of nodes connected by a set of edges. Typically, in biological networks the nodes are molecules and the edges are the biochemical or physical events between them (Newman, 2003).

There are currently a number of software tools for the analysis of biological networks, which accept user-generated data, as well as data extracted from relevant public repositories (Xenarios and Eisenberg, 2001; Bauer-Mehren \textit{et al.}, 2009). While there are ongoing efforts to standardise this data, problems such as database redundancy, non-standard identifiers and formats, and lack of data provenance are serious issues (Birney and Clamp, 2004). This thesis therefore presents a novel set of tools developed as \textit{web services} to address some of these points. Web services differ from traditional standalone and client-server tools, in that they are designed to be accessed programmatically, and have a public interface which allows tools, deployed on heterogeneous systems, to interface. The benefits and drawbacks to this method of software delivery are discussed in detail in Chapter 3, however an important advantage is that web services lend themselves to the development of computational \textit{workflows} for the automated querying of large datasets. Bioinformaticians often carry out manual workflows which entail the
transfer of data between different online and offline tools by cutting and pasting: a brittle method with much scope for human error (Stein, 2002). The automation of in silico processes seeks to remedy these shortcomings by introducing a workflow language in which data, tools and the links between them can be shared and reused, thereby explicitly capturing experimental provenance.

Workflows can be thought of as structured questions, which in this work are used to investigate the properties and features of intra-cellular networks, with a view to extracting new biological insight into their structure and function.

1.1 The aims of this research

The overall aim of this work is to provide an extensible toolkit of tasks developed as composable web services to facilitate research into cellular networks, supporting recognised formats for pathway and network exchange, and enabling the automatic querying of large datasets to serve as a launch pad for generating testable hypotheses.

A specific aim of this research is to critically explore the current state of the field of biological network construction and analysis, by investigating software solutions designed to query those networks, identifying their various advantages and drawbacks, and exploring the public repositories that contain interaction data. Based on this evaluation, a further aim is to establish how an emerging software paradigm, web services, can be leveraged to not only emulate the success of existing software, but address the drawbacks, in order to create an open, freely-available toolkit. Making the relevant functionality available as web services is a key objective, as they are themselves the components of computational workflows. Another key objective is therefore the creation of workflows that are designed to answer specific questions about holistic networks, pathways or individual network entities, comprising services developed by the author as well as those created by other service providers. In doing so, the author aims to show how a combination of open-source technologies can be used in conjunction with freely-available relevant biological datasets to gain a novel perspective on complex molecular networks.

1.2 Key contributions

The key contributions of this thesis are as follows:

1. Development of a framework to support the construction and analysis of biological
networks. The framework proposes categories of relevant tasks which maximise opportunities for composition and reuse.

(a) Proposal of the *common graph format* as a standard within the framework.
(b) Evaluation of common application architectures to enable suitable technology recommendations.

2. Comparative evaluation of four service implementation technologies, with particular emphasis on how easily they are used by a bioinformatics service developer, and their compatibility with the Taverna workflow enactment engine.

3. Web service development

(a) A toolkit of 68 web services developed by the author.
(b) Comprehensive documentation for each web service, comprising a detailed description and example usage demonstrated via tutorial examples, to facilitate adoption of the services and to establish their place within the framework.

4. Creation of a set of computational workflows to illustrate how the framework guides the development of a range of queries (which demonstrate varying levels of complexity) over interaction datasets. The workflows illustrate the application of the framework to relevant biological problems.

**1.3 Thesis structure**

The thesis is organised as follows:

The current literature and background to the project are described in Chapter 2, which is divided into three parts to cover the separate aspects of this work. The first is a detailed review of the study of biological networks, the second reviews current software solutions for network analysis, while the third is a summary of distributed computing solutions in bioinformatics.

Chapter 3 covers one of the major contributions of this work, that is, the framework upon which the software is designed and built. The framework is initially conceptual, but is later expanded to give technical recommendations for implementation. These technical recommendations form the basis of the next three chapters.
1.3 Thesis structure

Chapter 4 recounts practical experience gained after deployment of a simple web service, using four deployment technologies. The evaluation is based on framework recommendations, and concludes with a justification for the technology chosen to develop web services for this work.

The web services developed for this work are catalogued in Chapter 5. The documentation is structured such that potential users are informed as to the exact nature of each service, that is, what it does, the motivation for using it, and the situations in which it is applicable, as well as example inputs and outputs and implementation details highlighting the author’s specific contribution. A minimal example demonstrating the usage of each service is given in Appendix C.

Chapter 6 details the in silico experiments which may be carried out using web services developed for this work, as well as those made available by external providers. The workflows in this Chapter are developed using the Taverna workbench (Oinn et al., 2004).

Chapter 7 describes conclusions and lessons learned throughout the course of the project, as well as possible future work to extend the usefulness of the software produced.
Chapter 2

Background

The scope of this thesis extends over several subject areas, so this chapter is divided into three sections to put the goals of the project into context. The first section relates to the study of biological networks of various types, including metabolic, gene regulatory and protein-protein interaction. The second section provides a summary of several network visualisation and analysis tools already available, including those provided as standalone desktop tools, web-based tools, web services and programming libraries. The final section is a discussion on distributed computing in bioinformatics, and the ways in which data and tools are managed effectively. This chapter only briefly introduces web services which are one of the fundamental technologies used in this work, as a more detailed discussion appears in Chapter 3.

2.1 Biological networks

A network is a collection of objects and the symmetric or asymmetric relationships between them. This concept can be applied to many real-world situations and networks can be used to model such varied systems as the links between pages in the WWW (World Wide Web), the physical connections between routers and computers that make up the Internet, relationships between people in social networks and affiliated authors in a citation network. With the increase in data generated from high-throughput experiments, biological systems are also increasingly represented as networks (Junker and Schreiber, 2008).
2.1 Biological networks

2.1.1 Complex networks and graph theory

The network approach to biology can be considered as being a branch of systems biology, which seeks to understand biological function through integrative rather than reductionist approaches, as it is recognised that observed cell behaviours are rarely attributed to one component acting alone. Though a wealth of information has been gathered about certain individual biological components such as genes, proteins and metabolites, it is the interactions between these components that serve to better characterise biological systems (Kitano, 2002; Oltvai and Barabási, 2002; Han, 2008). Such interactions are either established via experimental means, or may be computationally inferred. As the quantity of data increases, so too do the number and size of interaction and reaction databases which make these data publicly available (Galperin and Cochrane, 2009).

The mathematical field of Graph Theory has been used for many years to analyse various types of real-world networks (Newman et al., 2006). Graph theory provides a range of algorithms and data structures which may be applied to computational representations of networks, whose results can lead to greater understanding of part or all of the network’s functionality.

Formally, a graph \( G \) is a set of nodes connected by a set of edges \( (G = \{N, E\}) \). Generally the terms “graph” and “network” are used interchangeably, though a graph is more likely to refer to the abstract notion of a set as defined above, and a network is the real instantiation of that graph, so for example the WWW is a network which can be modelled using a graphical data structure. The nature of the nodes and edges and what they represent for a given graph depend on the type of network being modelled. In the case of biological networks, the nodes are usually cellular components such as genes, proteins and metabolites and the edges denote interactions or reactions between them. Edges may be directed or undirected to specify the direction, if any, of reactions (for example an irreversible metabolic reaction would have a directed edge from the substrate to the reaction identifier, and another directed edge to the product of that reaction). Both nodes and edges may have related metadata to describe their characteristics further.

Two properties which have been identified as being common to such varied networks as the WWW and biological networks are their small-world and scale-free characteristics (Newman, 2003). Within small-world networks, most nodes can be reached from other nodes by traversing a relatively small number of edges. This is a network property first established for social networks (Milgram, 1967), and for biological net-
2.1 Biological networks

works implies that local perturbations can reach other parts of the network very quickly. The degree or connectivity of a node is the number of edges adjacent to it, and for a long time it was assumed that real-world networks had an even degree distribution, which is the probability distribution of these degrees over the whole network (Gross and Yellen, 2003). However a study carried out on the WWW (Barabási and Albert, 1999) established that the degree distribution followed a power law, indicating a network topology where relatively few of the nodes had a very large degree, termed network hubs, while the vast majority of nodes had a low degree. The term scale-free refers to this degree distribution, and implies that there is not a typical node degree which characterises the whole network (Albert, 2005).

The position of a node in a biological network can therefore help to characterise its function and importance to the network both locally and globally, and can have applications such as drug discovery (Korcsmáros et al., 2007) and identification of functional motifs (Alon, 2007).

2.1.2 Types of biological network

2.1.2.1 Metabolic networks

The metabolism of an organism consists of a set of enzymatic steps involving the biosynthesis and breakdown of organic molecules. This system of interconnected pathways is known as a metabolic network. Such networks tend to be modelled using directed graphs, due to enzymatic reactions being either reversible or irreversible, with the nodes as metabolites and the edges representing reactions converting substrates into products (Choi, 2007). Some representations also include enzymes and/or reactions themselves as separate nodes. In the post-sequencing era, enzyme activity is commonly deduced via sequence comparison (Espadaler et al., 2008).

Statistical studies of the properties of large-scale metabolic networks have been carried out. Jeong et al. (2000) carried out a systematic analysis of the metabolic networks of 43 organisms, and found that the large-scale structure was identical for all 43, and classified them as robust and error-tolerant networks. They also concluded that the average path length for all the organisms was about the same. A path in a network is a sequence of nodes where there exists an edge connecting one node to the next, and the path length is the number of connecting edges. This work was then extended by Ma and Zeng (2003b) who carried out a similar study, this time on 80 fully sequenced genomes. Interestingly, though they removed hub metabolites and encoded reversibility information for each reaction in the networks, they still found the same
large-scale structure as identified by Jeong et al. However, they found a clear difference in average path length between eukaryotes and archaea as compared to bacteria, in that generally the average path length in bacterial networks was much shorter.

A large-scale study which focused on a single organism was carried out by Wagner and Fell (2001). They performed a graph-theoretic analysis of the metabolic network of the bacterium Escherichia coli, and found that this was a small-world graph whose degree distribution followed a power law.

Horne et al. (2004) provided an alternative view of metabolic networks by constructing a reaction graph using data from the ENZYME database, in which enzymes were nodes and metabolites were edges. The data from ENZYME were processed to resolve synonyms, as well as remove hubs, which were deemed to give a less biologically meaningful representation of metabolic connectivity. Analysis of the resulting network revealed that despite deletion of hubs, the main component remained intact. It was also found that certain components were not connected to this main component, owing to the presence of generic names for metabolites. Biological networks are often separated into disconnected components, owing to missing information or lack of synonym resolution of molecule names.

Work has also been carried out regarding the evolution of metabolic pathways and constituent enzymes. A study by Rison and Thornton (2002) found evidence to support the ‘patchwork’ model of pathway evolution through the co-analysis of phylogeny and metabolism. Another 2002 study, carried out by Alves, Chaleil and Sternberg, used a network approach to analyse the global metabolic networks of a variety of species. They found that the percentage of pairs of homologous enzymes less than three steps away from each other in the network was significantly higher than would be expected, had the network evolved randomly. More recently, Vitkup et al. (2006) found that the structure and function of the Saccharomyces cerevisiae metabolic network influences important evolutionary processes. For example, enzymes with a higher degree evolve more slowly than those with a lower degree, and genes encoding enzymes with high degree are more likely to retain duplicates in evolution.

2.1.2.2 Protein-protein interaction networks

Protein-protein interaction (PPI) networks are generally modelled as undirected graphs whose nodes are proteins and whose edges are the physical interactions between proteins (as opposed to chemical reactions between metabolites and enzymes) (Junker and Schreiber, 2008). PPIs are vital to a cell as they govern many important func-
2.1 Biological networks

In biological networks (Nooren and Thornton, 2003), for example, signal transduction is a process by which signals are transferred from the outside of the cell to the inside, and are then propagated through it, leading to a number of cellular responses including changes to metabolism, or activation or repression of transcription. The PPIs of signalling molecules are responsible for starting a signalling cascade which leads to these responses. Protein complexes are another important class of PPIs, as a protein may be activated or inhibited if and only if it is part of a functional complex. Such complexes may be stable over time, but a protein may also take part in much briefer interactions with other proteins in order to modify them, for example a protein kinase that phosphorylates another protein.

PPIs are commonly detected using the yeast two-hybrid method (Fields and Song, 1989). This technique is based on the modular organisation of many transcription factors (TFs). The DNA-binding domain (BD) of the TF is fused to a protein of interest (the ‘bait’). The activation domain of the TF is fused to another protein (the ‘prey’). When the genes are transformed into a cell and expressed, two hybrid proteins are produced. The bait binds to an upstream activating sequence (UAS) of a reporter gene, and the prey binds the remaining transcriptional machinery. If the two proteins bind each other, the transcriptional machinery will be brought into close proximity with the UAS, and the reporter gene will be expressed.

Two large-scale yeast two-hybrid studies carried out by Uetz et al. (2000) and Ito et al. (2001) identified 957 interactions between 1004 yeast proteins, and 4549 interactions between 3278 proteins respectively. There was surprisingly little overlap between the two datasets, so the Ito et al. study combined them to generate a single large interaction dataset which was queried for biologically interesting subnetworks. Eisenberg et al. (2000) built on this idea by proposing the notion of “functional protein networks”, with links between proteins predicted by both computational and experimental methods. The function of a protein has classically been derived by focussing on its individual action, however this view has been expanded to consider the protein’s function in the context of interactions with other proteins.

Jeong et al. (2001) carried out the first large-scale graph-theoretic analysis of a PPI network by assembling PPI data from both the Uetz et al. study and the Database of Interacting Proteins (DIP, Salwinski et al., 2004). This analysis sought to establish a link between the essentiality of a protein and its position in the overall network, which in this instance was identified as having a scale-free topology. The network exhibited tolerance towards random errors, whereas selective removal of the proteins with the most number of connections increased the network diameter rapidly. It was then
established that these high-degree proteins with more than fifteen links are more likely to be essential than those with five links or fewer.

This study focussed on ranking proteins by their degree as a measure of their importance in the network, but in recent years, measurement of the betweenness centrality of nodes has emerged as a more accurate predictor of protein essentiality (Newman, 2003). For a given node, the betweenness centrality is the proportion of shortest paths between other nodes that it occurs on.

A study by Joy et al. (2005) assembled a yeast PPI network from data in DIP and the Munich Information Center for Protein Sequences (MIPS, Mewes et al., 2002). Analysis of this network uncovered the existence of proteins that exhibited high betweenness centrality values, and were also hubs, an intuitive result as there are many proteins connected to hubs, resulting in them appearing on many shortest paths between pairs of proteins. However lower degree nodes displayed a greater amount of variation in their betweenness scores, indicating that some of these may also be globally important, and it was found that the essentiality of a protein is at least as dependent on its betweenness centrality value as its degree. Gandhi et al. (2006) further refuted the findings of Jeong et al. by establishing, using a much more comprehensive dataset for yeast knockouts, that the lethality of a gene could not be confidently predicted on the basis of the degree of the gene alone. More recently, Bader and Madduri (2007) corroborated the finding that low-degree nodes show significant variation in betweenness values, for the human PPI network comprising around 44,000 interactions between 18,000 proteins.

### 2.1.2.3 Gene regulatory networks

Regulation of gene expression is another important cellular process which may be represented as a network. Gene expression is a multi-step process starting with the transcription of a gene to produce messenger RNA (mRNA). This mRNA is translated into a protein which may undergo post-translational modification - the attaching of various types of functional groups (such as phosphates, acetyl or methyl groups), or tertiary structural changes (Polevoda and Sherman, 2003). It is the first part of this process, transcriptional control, which is the most common means of gene regulation and is modelled as a transcriptional regulatory network or gene regulatory network (GRN). The other steps are assumed to occur and therefore do not require explicit representation.

The components of a GRN may be detected using a number of experimental means, however a commonly used technique is an electrophoretic mobility shift assay (EMSA,
Garner and Revzin, 1981), which identifies DNA-protein binding. If a protein binds to the promotor region of the DNA, the molecular weight will increase, which can then be detected when the sample is run on polyacrylamide or agarose gel, as the speed is determined by the size and charge of molecules. Transcriptomics data obtained from high-throughput microarray experiments may also be used, but often represents indirect effects and therefore should not be taken alone when determining the elements of a GRN (Needham et al., 2009).

Transcriptional control affects the selection of genes to be transcribed and the rate of transcription. A special class of protein known as transcription factors (TFs) bind to a specific regulatory sequence of DNA, which either inhibits or facilitates the binding of RNA polymerase to the regulatory sequence, known as the promoter (Ihmels et al., 2004). GRNs dynamically regulate the level of expression of each gene by various methods, often incorporating dynamic feedback loops which also provide regulation of the network architecture and output. GRNs are usually modelled using directed graphs, with edges representing interactions between TFs and the genes they regulate. Brazhnik, de la Fuente and Mendes (2002) describe GRNs as phenomenological models which are high-level starting points onto which details of proteins and metabolites can be added to expand the network.

Various statistical studies of the properties of GRNs have been carried out, in S. cerevisiae (Guelzim et al., 2002; Farkas et al., 2003) and E. coli (Shen-Orr et al., 2002). These studies revealed that transcriptional networks also exhibit a scale-free topology, and additionally contain certain network motifs (patterns of connected nodes) appearing at frequencies much higher than in random networks, suggesting they have specific functions in information processing.

### 2.1.3 Limitations on analyses

A number of parallels have been shown to exist between biological networks and other real-world networks with regard to their large-scale structure. It is important to bear in mind however that the amount of reaction and interaction data reported for organisms is by no means complete, and so any comparisons made between biological networks and, for example the Internet or social networks should not be pushed too far (?). The experimental methods which produce biological networks are error-prone and result in a high rate of false-positives, and so the results of any analyses must be examined closely to determine their biological relevance (Qi and Ge, 2006).
2.1 Biological networks

2.1.3.1 Integrated networks

The previous sections describe some types of biological network and discoveries made with regard to their global structure. Dividing molecules and interactions in this way, however, is a simplification of the real biological processes taking place in a cell. The actual scenario is more akin to a ‘network of networks’ (Barabási and Oltvai, 2004). Transcription factors activate genes, to produce proteins which participate in PPIs, are transcription factors themselves, or are enzymes which transform substrate metabolites into products, some of which alter transcription factor binding kinetics.

A number of studies have been carried out, which aim to reach more meaningful biological conclusions based on the analysis of integrated networks. A key issue is the identification of functional modules in such networks that are supported by interactions of different types (Sharan and Ideker, 2006).

Work carried out by the previously-mentioned Shen-Orr et al. study revealed motifs in networks comprising a single type of edge i.e. those between transcription factors and the operons they regulate. Yeger-Lotem et al. (2004) extended this by constructing an integrated S. cerevisiae network containing both transcriptional connections (a directed edge from the TF to its target gene) and PPIs (an undirected edge connecting two interacting proteins). Analysis of this network revealed several significant network motifs containing two, three and four proteins. These motifs contain a mixture of transcriptional edges and PPI edges. For example, one three-protein motif contains two interacting transcription factors that co-regulate a third gene. Of the 63 statistically significant network motifs made up of four proteins, only 6 could not be constructed from a three-protein motif in combination with an extra node or another three-protein motif. One particular four-protein motif of interest contains two transcription factors that co-regulate genes, which may indicate patterns of overlapping regulation.

Kelley and Ideker (2005) studied the combination of synthetic lethal genetic interactions (in which mutations in two non-essential genes are lethal when combined) and physical interactions among proteins. Two structures of interest were searched for: pairs of subnetworks of PPIs interconnected to each other by a dense pattern of genetic interactions, and clusters enriched for both physical and genetic interactions. It was found that the first structure was more prevalent, suggesting that genetic interactions tend to span multiple physical regions of the network rather than occurring between protein subunits within a single pathway.
2.2 Existing network analysis software

2.1.3.2 Temporal and spatial effects

Temporal and spatial factors also have important implications when analysing biological networks, as not every interaction or reaction occurs at the same time, or within the same cellular compartment.

The network of interacting genes and proteins is a dynamic system, evolving according to fundamental laws of reaction, diffusion and transport (Tyson, 2007). Sophisticated network modelling looks at the dynamic changes and quantitative effects of one component upon another, but simulations of cellular behaviour using holistic models require too much CPU time to be practical given current hardware. Whole-cell representations at the level of network topology are far simpler. The results of analyses on ‘static’ network representations must therefore be examined closely to determine their biological relevance. d’Alché-Buc and Schachter (2005) recognise that static topological analyses are useful when applied to genome-scale networks, as they aim to identify underlying biological mechanisms or design principles.

The view that quantitative network models yield more biologically accurate hypotheses regarding the structure and function of complex biological networks is supported in the literature (Strohman, 2002; Kharchenko et al., 2005; Blinov et al., 2008; Gopalacharyulu et al., 2009). It remains the case, however, that large-scale data on non-linear network dynamics are not yet available. The link between static structure and dynamic behaviour must be made carefully, to lead to the successful prediction of the functional characteristics of a network’s or pathway’s behaviour. Spatial information is more common, especially for model organisms, but remains incomplete.

The importance of temporal and spatial factors is illustrated by a study carried out by Han et al. (2004), in which two categories of protein hubs in the S. cerevisiae PPI network were found: ‘party hubs’, which interact with most of their partners simultaneously, and ‘date’ hubs, which bind their partners at different times or locations. Date hubs organise the network by connecting modules (functional groups) together, while party hubs tend to function inside these modules.

2.2 Existing network analysis software

A variety of software tools to carry out biological network analysis tasks are currently available. These tools can be divided into four categories: standalone or monolithic software, client-server software, programming libraries and web services. A discussion of some of the major tools in each category is presented here.
2.2 Existing network analysis software

2.2.1 Standalone software

Network visualisation and network analysis are two separate computational tasks that are often, unsurprisingly, implemented alongside each other in software packages (Saraiya et al., 2005). Bioinformatics applications benefit from attractive user interfaces that appeal to non-expert computer users, and the ease of utilising such software is increased by visually guiding the user through analyses (Tisdall, 2001). Network analysis tools that are currently available, with very few exceptions, contain a strong visualisation component (Suderman and Hallett, 2007). Holistic cellular models, in the main, tend to be very large with potentially hundreds of thousands of nodes, and visualisation is therefore a problematic issue in that a layout algorithm, however efficient, will usually produce complicated, densely-packed diagrams that are very difficult to interpret usefully by humans. Figure 2.1 gives an example of this, and shows a globally reconstructed human metabolic-network (Duarte et al., 2007) visualised using the yFiles organic layout in the network analysis and visualisation package, Cytoscape (Shannon et al., 2003).

Visualisation of such complex networks therefore presents significant challenges, and is the subject of much ongoing work (Becker and Rojas, 2001; Han and Ju, 2003; Li and Kurata, 2005; Kato et al., 2005). However as an effective layout cannot aid the understanding of the features of the network alone, analysis algorithms commonly borrowed from the field of Graph Theory should be applied to highlight features of interest, some of which are described in Section 2.1.

The number of network visualisation and analysis tools has grown considerably in the last few years, giving support to the notion that to better understand complex cellular machinery, studies of networks of interactions should be carried out, as well as those which seek to understand the function of individual cellular components. A review carried out by Saraiya, North and Duca (2005) identified 16 software tools for visualisation and analysis, though two years later Suderman and Hallett (2007) identified over 35. Saraiya et al. emphasised visualisation over analysis though the majority of the tools surveyed contained basic analysis functions as well, and this is also true of the Suderman and Hallett review. Table 1 gives details of seven standalone tools, comparing their visualisation and analysis capabilities, together with requirements, integration with biological resources and other parameters. Network Workbench has not yet been subject to peer review and is not specific to biological networks, but has been included as a stable version is available for download, and it provides a large number of graph-theoretic operations which can be applied to biological networks when
2.2 Existing network analysis software

Figure 2.1: Human metabolic network visualised using the yFiles organic layout in Cytoscape. The network has 6390 nodes and 14731 edges. (a) An enlargement of one part of the network. Diamond-shaped nodes are molecules either produced or consumed by reactions, which are denoted by circles. A green line connecting a molecule to a reaction indicates that the molecule is a substrate. A red line connecting a molecule to a reaction indicates the molecule is a product. (b) and (c) are components which are disconnected from the main subgraph. This may be due to gaps in the knowledge regarding reactions and reactants, or synonyms which have been resolved incorrectly, so the same molecule appears more than once under a different name. These issues may also affect the accuracy of the main component.

From the perspective of a user, certain parameters used to describe the tools are of particular importance or relevance. For example, the input and output formats accepted and produced by a piece of software in effect dictate to the user how their data should be represented if they wish to use the software to analyse or visualise their network data. For this reason, packages such as Cytoscape, Patika and ProViz all offer the advantage of accepting and producing standard file formats used to represent network and pathway data (e.g. SBML, BioPax and PSI-MI). Various data repositories offer the export of network data in some of these standard formats, allowing a user to
<table>
<thead>
<tr>
<th>Cytoscape</th>
<th>VisANT</th>
<th>Pajek</th>
<th>Piana</th>
<th>ProViz</th>
<th>Osprey</th>
<th>Network Workbench</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network analysis</strong></td>
<td>Via plugins. Clusters, topological parameters, shortest paths, expression activated subnets</td>
<td>Shortest paths, degree distribution, highly connected subgraphs, network motifs, cycles</td>
<td>Shortest paths, betweenness and closeness centralities, clusters</td>
<td>Interaction distances, clustering</td>
<td>Subgraphs, clustering</td>
<td>None</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>Java</td>
<td>Java</td>
<td>Delphi</td>
<td>Python</td>
<td>C++</td>
<td>Java</td>
</tr>
<tr>
<td><strong>Import format(s)</strong></td>
<td>SBML, BioPax, Cytoscape SIF, GML, XGMML, PSI-MI</td>
<td>VisANT format</td>
<td>Pajek format</td>
<td>PSI-MI</td>
<td>PSI-MI, Tulip format</td>
<td>Osprey format, custom format</td>
</tr>
<tr>
<td><strong>Export format(s)</strong></td>
<td>XGMML, GML, SIF, PSI-MI</td>
<td>VisANT format</td>
<td>Pajek format</td>
<td>Cytoscape SIF</td>
<td>PSI-MI</td>
<td>Osprey format</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Cross-platform</td>
<td>Linux, Windows</td>
<td>Windows</td>
<td>Linux, Mac OS X</td>
<td>Linux</td>
<td>Cross-platform (different download for each)</td>
</tr>
<tr>
<td><strong>Software requirements</strong></td>
<td>Java SE 5 or 6</td>
<td>may require installation of JRE</td>
<td>None</td>
<td>Python</td>
<td>Tulip, OpenGL, LibXML2, qt, glut, CURL</td>
<td>Java</td>
</tr>
<tr>
<td><strong>Hardware requirements</strong></td>
<td>For large networks - as fast a processor as possible, 2GB+ RAM, high-end graphics card</td>
<td>Not available</td>
<td>Not available</td>
<td>mySQL server requires minimum 6GB disk space</td>
<td>Not available</td>
<td>Pentium IV 450 MHz CPU, 256 MB of memory, 70 MB disk space</td>
</tr>
<tr>
<td><strong>Network types</strong></td>
<td>PPI, metabolic</td>
<td>Gene regulatory, metabolic</td>
<td>Biological, social, genealogies</td>
<td>PPI</td>
<td>PPI</td>
<td>Gene regulatory, PPI</td>
</tr>
<tr>
<td><strong>Database integration</strong></td>
<td>Data retrieval via web services from IntAct, NCBI and Biomart</td>
<td>Predictome (integrates KEGG, GO, MIPS, BOND &amp; HPRD)</td>
<td>None</td>
<td>Uniprot, NCBI, COG, SCOP, GO, DIP</td>
<td>None</td>
<td>BioGRID</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of common standalone network visualisation and analysis software packages
go straight from obtaining the data to submitting it to a software tool for analysis. Proprietary formats such as those used by Osprey, VisANT and Pajek create an extra step for the user as they must transform their data into the correct format in order to take advantage of any functions offered by these tools. It may be the case that the user has to write the conversion program themselves, introducing additional complexity. Only being able to save the results of an analysis in the proprietary format of a specific tool locks the user in further and limits them to the feature set of one tool.

One of the most important aspects of network software in the context of this project is the implementation of graph-theoretic operations for biological network analysis. As mentioned, visualisation is an important task but the size of networks means that it is not always the most useful way to extract significant network features. Of the tools surveyed, almost all provide analyses such as shortest-path calculations, centrality measures (degree, betweenness and closeness), clustering algorithms, cycle detection and network diameter. The most complete solution in terms of graph-theoretic operations is Network Workbench, however this suffers from the disadvantage of not being specifically designed for biological networks, being aimed at physicists and social network researchers as well, and therefore does not accept standard biological-network-exchange file-formats as input.

Cytoscape, though designed for biological networks, provides network analysis functionality via plugins. This functionality means that the user must install different plugins depending on what analyses they require, as no one plugin provides all possible graph-theoretic algorithms. Also, carrying out graph-theoretic analyses in Cytoscape usually causes the results to be captured within the visualisation of the network; for example the plugin ShortestPath allows for the selection of two nodes; if a shortest path is found between them, the nodes along it are highlighted on the network diagram itself. This is visually useful, and the highlighted nodes can be used to generate a sub-network that has all the properties of the whole network, and can be further analysed. Another plugin, NetworkAnalyzer (Assenov et al., 2008), provides a great many topological measures such as clustering coefficient, connected components, diameter, radius, shortest paths and degree distribution among others; however all of these parameters are calculated at once and there is no option to choose a single analysis at a time. This means that the calculation is very time-consuming for large networks.

Of the nine tools, seven are completely or partly written in Java, and as this is a platform-independent language, it allows users to execute the application on a variety of platforms. Despite the flexibility offered by Java, from the developer’s perspective, it may be necessary to provide a different version of the application for users on different
platforms, for example, Osprey. In contrast Pajek and ProViz offer less flexibility, as they may only be installed and executed on Windows and Linux respectively.

As well as the language a tool is written in, other requirements may need to be met for it to run successfully. These take the form of software or hardware prerequisites, and it is advantageous to have as few of these as possible, relieving a possibly non-expert computer user of the task of correctly installing them. An example of a tool with software prerequisites is PIANA, whose installation entails setting up a MySQL database (with client and server), installation of several external Python modules and creation and modification of environment variables. In contrast, Cytoscape does not have any such software prerequisites, apart from the correct version of Java. However, as a locally run standalone application the hardware requirements vary according to the network size; for manipulation of larger networks the processor should be “as fast as possible” as stated in the Cytoscape user manual, and there should be available a minimum of 2GB of RAM. Additionally a high-end graphics card is required for visualisation.

Data integration plays a very important role in the analysis of biological networks, and a tool that provides access to various repositories of interaction data will allow the user to analyse a more complete ‘picture’ of cellular interactions. VisANT for example is based on the Predictome database (Mellor et al., 2002), which is a database of predicted functional associations between genes and proteins in a variety of organisms. Interaction data is deduced via both experimental and computational techniques, and the user may access data from KEGG, GO, MIPS, BIND, HPRD and BIND. In contrast, applications such as Network Workbench and ProViz take network files as input and, once loaded, provide visualisation and analysis tasks and return results without linking to external data sources. Cytoscape has implemented a Web Service Client Manager from version 2.6.0, which enables the creation of plugins to access data programmatically from IntAct, Pathway Commons, NCBI Entrez Gene and Biomart.

2.2.2 Web-based and other client-server tools

A web-based client such as PatikaWEB (Dogrusoz et al., 2006) provides a fully-featured front-end in the form of a website, while the actual processing and analyses are carried out on a remote machine, which returns results to the user via the website user-interface. This application provides access to the Patika database, which integrates data from Entrez Gene, UniProt, PubChem, GO, IntAct, HPRD and Reactome, and supports analyses such as discovery of feedback loops and subgraphs. The client can support
import of data in BioPax format, and export in both BioPax and SBML.

Another method for delivering an application is via Java Web Start. This is a framework that allows Java software to be started directly from the Internet using a web browser, giving the advantage of automatically downloading and installing the appropriate Java Runtime Environment (JRE) if the user does not have it already, thereby overcoming compatibility problems caused by browser plugins and different versions of the Java Virtual Machine (JVM). VisANT may also be run in this way, as well as being available as an online Java applet. BiologicalNetworks (Baitaluk et al., 2006) is another tool delivered via this method. It provides graph-theoretic analyses such as cycle identification and topological measures such as average degree, average distance and network diameter. Supported formats for input and output include SBML and PSI-MI as well as its own BiologicalNetworks format. It can be used to study metabolic and gene regulatory networks and integrates data from KEGG, BIND, TRANSFAC and MIPS.

2.2.3 Web services

Recently a web services-based toolbox of network analysis functionality has been released by Brohée et al. (2008). The Network Analysis Tools (NeAT) software can be accessed from the website¹, and provides a selection of analysis tools, including shortest path analyses, topology measures such as degree, closeness and betweenness centralities and clustering algorithms. This toolkit is very similar in concept to the software provided through this work, however there are certain differences. The preferred client is the website which is specifically designed to access the services. Programmatic access to the tools is also provided through an interface description, which may be loaded into the workflow enactment software Taverna, however the emphasis is on web-based client access.

2.2.4 Programming libraries

Support for graph analyses is available through a number of programming libraries for a variety of languages. Using these libraries generally calls for a level of expertise much greater than that required for either standalone tools or web-based client-server applications. They are used programmatically and while they may provide powerful network-analysis functionality they are not a suitable option for non-expert progr-
2.3 Distributed computing solutions in bioinformatics

An ongoing issue in bioinformatics is that of data management: its storage, manipulation and integration. High-throughput experiments generate vast amounts of information which must be effectively handled if it is to yield new biological insights. Of particular relevance is the issue of data integration. There are over a thousand public and commercial databases (Galperin and Cochrane, 2009) which leads to one of the most fundamental problems facing in silico research, which is that heterogeneous data resources lack interoperability. Programmatic interfaces vary from resource to resource, and are sometimes not provided at all. In the case of the latter, it is necessary to ‘cut and paste’ data between web forms, or ‘screen-scrape’ - a brittle and unreliable method as described by Stein (2002). Even the simplest analyses require biologists to handle data from several repositories in various different formats. Ongoing standardisation issues mean there is no single accepted requirement for how biological information is stored, and tools and databases exist in many different database formats and are represented by various legacy flat-file formats.

One attempt to minimise the problems associated with these issues is to employ a web services model for data retrieval and analysis. A web service is defined formally by the W3C as ‘a software system designed to support interoperable machine to machine interaction over a network. It has an interface described in a machine-processable format’. Web services are a distributed technology, originally developed by the e-business community as a solution to interoperability issues regarding data exchange, and crucially were designed to be platform- and language-independent. They use standard Internet protocols, and offer the necessary architecture for flexible and expandable integration of diverse scientific tools.

The functionality of web services may be built on further to address the aforementioned difficulties of manually performing a sequence of analyses by creating a software pipeline, or computational workflow, which comprises a set of web services, chained

---

1 [http://search.cpan.org/~jhi/Graph-0.84/](http://search.cpan.org/~jhi/Graph-0.84/)
5 [http://www.w3.org/TR/ws-arch](http://www.w3.org/TR/ws-arch)
together to automate analyses. The output of one service becomes the input to the next, with no intervention from the user. Workflows in bioinformatics are therefore transformed into a fully automated graph of processes, starting with inputs provided by the user and transformed by a sequence of web services into an end result. Chapter 3 gives a more detailed description of the protocols underpinning the web services specification and the mechanics of workflow execution.

Web services are by no means the first distributed solution to address the problems of interoperability in bioinformatics. The Common Object Request Broker Architecture (CORBA)\(^1\), defined by the Object Management Group (OMG), has been an established standard since the early 1990s, and defines protocols which enable heterogeneous applications running on various platforms to interoperate, by ‘wrapping’ code into objects such that it can be accessed by clients across a network. An Interface Definition Language (IDL) describes the object’s interface in a language-independent fashion, allowing communication between wrapped objects written in different languages.

In much the same way that web services are now being adopted with increasing frequency by the bioinformatics community, in the mid- to late-nineties, CORBA was in a similar position, and was hailed as a solution for linking heterogeneous data resources (Barillot et al., 1999; Jungfer and Rodriguez-Tomé, 1998; Barillot et al., 1996). Stevens and Miller (2000) reviewed the advantages of utilising CORBA as a solution to the problems raised by the growing diversity of distributed resources, concluding that it was a viable option for researchers.

An interesting case study follows the development of the European Molecular Biology Laboratory (EMBL) nucleotide sequence database\(^2\) at the European Bioinformatics Institute (EBI). Wang et al. (2000) describe using the CORBA programming interfaces to the data stored in the EMBL database, as the existing use of flat-file formats for storage and returning query results was inflexible. As a follow-up to this (Wang et al., 2001), it was noted that CORBA was an incomplete solution to the problems that arose from the flat-file format, namely that it was too complex for biologists to use, and was often blocked by firewalls preventing effective use over the Internet. At the time, demand grew for EMBL to distribute data using XML, giving rise to the XEMBL project. The conclusion at this point was that CORBA together with XML was a successful method for the accessing and distribution of EBML data, with the added advantage that XML could be distributed over the Internet via SOAP, and pass through firewalls accordingly. Wang et al. (2002) released XEMBL as a web service, and eventually

\(^1\)http://www.corba.org
\(^2\)http://www.ebi.ac.uk/embl

2.3 Distributed computing solutions in bioinformatics
the CORBA interfaces were abandoned altogether, on the basis that communication through firewalls was difficult. The lightweight nature and flexibility of web services resulted in their being viewed as a superior solution (Pillai et al., 2005).

For completeness, alternative distributed technologies will also be discussed. An alternative to CORBA was Microsoft’s Distributed Component Object Model (DCOM), which has now been superseded by the Microsoft .NET framework. DCOM provided a set of interfaces for enabling client objects to request services from server objects on other computers in a network. The major disadvantage of these technologies is that they are limited to the Microsoft platform, though server components can be written in a variety of languages, for example Java or Visual Basic. As the trend in bioinformatics is to develop open-source tools, proprietary formats have not been embraced, and there has been little support for exposing code in such a manner.

Java RMI is yet another middleware solution which provides communication between clients and servers written in Java, though the platform-independent nature of the language means RMI-based applications are able to run on a wide variety of platforms. In an early comparison of the three technologies described, Gray (2004) highlights the fact that both Java RMI and CORBA have optimised connection protocols, which have detailed rules, in comparison to the universal technologies such as HTTP for transport and XML for textual representation of data. However, this increased model of interoperability also results in some performance issues, as generating and parsing XML documents is time-consuming and resource heavy, so sending and receiving SOAP messages may be slower than the analogous mechanisms in both Java RMI and CORBA. Despite this, Gray noted that SOAP has seen significant improvements to performance as compared to earlier studies carrying out similar comparisons.

A number of major data banks provide a web service interface to their data, for example EMBL-EBI as mentioned, the DNA Data Bank of Japan (DDBJ, Tateno et al., 2002), the Protein Data Bank of Japan (PDBJ, Standley et al., 2008) and the National Centre for Biotechnology Information (NCBI)1. Additionally some smaller databases such as the Kyoto Encyclopaedia of Genes and Genomes (KEGG, Kanehisa et al., 2008) and the Biomolecular Interaction Database (BIND, Alfarano et al., 2005) have created web services that access their data. A list of biological web-service providers is maintained on the myGrid web site.2

---
2http://www.mygrid.org.uk/wiki/Mygrid/BiologicalWebServices
such as BioGRID (Stark et al., 2006), DIP and the Molecular Interaction Database (MINT; Chatr-aryamontri et al., 2007). This is a particular disadvantage of what is a relatively new technology, and it is generally the case that a web-based front-end is provided to access and download records.

### 2.4 Conclusion

This chapter has demonstrated that biological network analysis is a useful and important approach in systems biology, to further our understanding of cellular networks and the roles played by constituent molecules. A number of software tools have already been proposed and developed which effectively handle such networks, and aim to provide biologists and bioinformaticians with a range of functionality, including graph-theoretic algorithms for analyses, and links to external databases. On the subject of data integration, it has been shown that the vast quantity of data generated from high-throughput experiments can be effectively managed by distributed solutions. With these conclusions in mind, the following chapter proposes a framework for a software system for the construction and analysis of biological networks, initially a conceptual overview, and then expanded to detail the practical realisation.
Chapter 3

A Framework for Network Analysis Software

The aim of this chapter is to formalise the framework upon which the software system for the construction and analysis of biological networks is built. The first section describes the conceptual framework, which outlines the fundamental elements of the system. The subsequent sections expand the conceptual framework to detail the practical realisation of framework elements, by providing suitable technical recommendations for implementation.

3.1 Conceptual framework

The description of the conceptual framework is divided into two parts. The general conceptual framework gives an overview in general terms of the elements in the system, and how they are related to each other. These elements are then expanded in the context of biological network construction and analysis.

3.1.1 General conceptual framework

The general conceptual framework consists of four elements:

- User
- Data
- Tasks
- Workflow
3.1 Conceptual framework

The four elements and the relationships between them are shown in Figure 3.1. A user with some data to process has access to a ‘pool’ or set of tasks. Each task manipulates data in some way, or may be combined with external data sources, and the user may wish to make use of several such tasks to achieve an end result. The data are passed through a particular user-defined subset of tasks, producing intermediate output which is consumed by a subsequent task or tasks, until the final output is returned. The collection of tasks to be executed is assembled into a workflow, that is, a graph of interconnected tasks with ordering constraints.

3.1.2 Network-specific conceptual framework

The general conceptual framework is applied to the domain of biological network construction and analysis, resulting in a network-specific conceptual framework. At this stage the framework is independent of implementation details, and expands the elements data and tasks.

A central issue regarding data as part of the framework is the representation of biological networks. There are a number of computational representations used to record interactions and reactions between biological components. For example, holistic models of metabolism for particular organisms may be represented using the XML-based SBML format (Hucka et al., 2003) or in BioPAX format (Luciano, 2005). PPI
data may be obtained in the standards proposed by the HUPO Proteomics Standard Initiative (HUPO-PSI), either in an XML-based or tab-delimited format (Hermjakob et al., 2004). Such formats contain a large amount of information which is surplus to requirements when considering a network in terms of carrying out some graph-theoretic analysis, for example the experimental origin of the data, or alternative molecule names.

The existence of various types of network data is a potential limitation when designing and realising tasks, as each task may require tailoring to be compatible with a particular format. A common graph format is therefore proposed, which describes a network as a list of tab-delimited interacting pairs, with one pair per line. A line may also contain a single node rather than a pair. The format is a minimal representation of networks as it captures only the nodes in the network and their connections to each other. For biological networks, nodes may be biological entities such as genes, proteins or metabolites, or concepts such as reactions or events. Connections, or edges, may represent physical bindings or biochemical interactions. The common graph format is proposed to standardise data from different sources, removing the need to design specific tasks to process heterogeneous data. A drawback to using the lowest common denominator network is that any related metadata do not appear in the graph itself, and may be lost altogether if not captured in a separate file.

Figure 3.2 demonstrates the common graph format representation of a toy network, together with two possible corresponding network diagrams to visualise the network. An important advantage of this format over those previously mentioned is a smaller file size, enabling more efficient processing. For example, the SBML version of the Palsson human metabolic-network discussed in Chapter 2 is 5.8 Mb, whereas the equivalent common graph version is 433 kb. Another advantage is that the format is particularly well suited for analysis using graph-theoretic algorithms, as to calculate topological metrics for the whole network, localised areas of interest and individual nodes, requires only the nodes and edges connecting them.

A drawback to this minimal approach, however, is that certain information about the interactions and reactions is lost, which may enhance the biological significance of results. To balance this, a task which converts a particular format into the common graph format also produces other relevant output containing more detailed information regarding the particular biological entities and their interconnections in the network. These data are specific to each computational representation, and may be used later in the workflow to improve the readability of results.

As well as data, the general framework describes a set of tasks available to the user. In the context of biological network construction and analysis, this set is divided into
3.1 Conceptual framework

Figure 3.2: Two toy networks to demonstrate the common graph format. (a) An undirected network is represented using tab-delimited pairs of nodes, where the order of the nodes is unimportant. This network contains four nodes and three edges. (b) A directed network is also represented using tab-delimited pairs of nodes, however as direction is encoded the order in which they appear on each line is important. There are four directed edges, two of which connect the same pair of nodes in opposite directions. Both networks contain a singleton node which appears in the common graph format on its own line, and is represented in the network diagrams without any incident edges.
3.1 Conceptual framework

four categories; data retrieval, data transformation, data analysis and output rendering:

- **Data retrieval** A data retrieval task accepts as input an identifier (e.g. for a particular database) and returns a record or collection of records representing a whole biological network, a subset (subgraph) of a network, or a pathway.

- **Data transformation** A data transformation task accepts as input network data in some format and returns the same network in another format. In the context of this framework, this produces data which is either suitable for analysis, or data which is suitable for rendering.

- **Data analysis** A data analysis task accepts as input the common graph format and returns the results of an analysis.

- **Output formatting** This category comprises two subcategories, output transformation and output rendering. An output transformation task accepts as input data intended as the final result of a workflow, and transforms it to a format that either makes it human-readable or suitable for further rendering. An optional output rendering task may then follow this, and applies the appropriate renderer to transformed data, again to make it easier to interpret.

The categories are proposed to facilitate the construction of workflows. Formalising categories in this way enhances interoperability, as when tasks are realised in practice, the correct order in which they should appear in a workflow is defined. This benefits the user by preventing nonsensical scenarios being created and executed. Defining the inputs and outputs to each type of task also enhances interoperability by indicating how new tasks may work with the existing set. The categories also maximise reuse, as any data analysis tasks need only be implemented once, and can be applied multiple times to different networks which were originally represented using heterogeneous formats.

Tasks in the four categories are therefore responsible for routing data through a workflow according to the order specified in this framework. This applies to tasks implemented by the author for this work. However there may be a requirement to make use of functionality offered by external developers, which also fits a framework category, for example, an external data retrieval step. When accessing such functionality, additional generic tasks acting as a logical wrapper or interface may be utilised, to enable relevant tasks to fit into the appropriate place in the framework. If an independently developed task adopts different idioms of data representation, a particular ‘shim’ will be required to facilitate the transfer of data between the categories (Hull et al., 2005).
3.1 Conceptual framework

Figure 3.3: Network-specific conceptual framework. Four categories are introduced to divide tasks within the context of biological network analysis, shown in green. The output from one category becomes the input to the next; inputs and outputs are shown in red. Generic wrappers are sometimes required if a framework task from an external source is used, and are necessary to ensure that data are routed correctly. The nature of these generic tasks depends on the exact specification of both input and output data for a particular framework task.

The examples in Chapter 6 contain instances of such generic wrappers used to achieve this data transfer throughout workflows. Figure 3.3 illustrates the expanded conceptual framework to reflect the introduced categories, the data they accept as input and produce as output as well as tasks for generic control.

There are two main classes of input which determine which framework categories will be used. The first are database identifiers, which are used to retrieve a record or collection of records, representing a biological network. The type of data retrieved depends on the nature of the database. For example, the identifier ‘9606’ as input to the IntAct PPI database retrieves all human protein interactions, the identifier ‘sce00010’ as input to the KEGG pathway database retrieves the yeast glycolysis/gluconeogenesis pathway, and the identifier ‘REACT_578.1’ as input to the Reactome pathway database retrieves the human apoptosis process. The second type of input is holistic network data, for example SBML or PSI-MI files as described previously. Such data may be produced by the user themselves, or downloaded from other sources (for example,
Figure 3.4: Example inputs to the framework and how they are transformed by tasks from the different categories, to create different workflows. The red arrow shows the direction of data flow. In some cases a particular category may not be used however the ordering of the categories remains consistent.

Reactome provides human reactions in SBML format. Figure 3.4 gives three possible workflows and the framework categories used.

### 3.2 Technical framework for tasks

There are several possible technologies within which tasks may be realised. Common approaches in bioinformatics include single-tier (or monolithic) tools, client-server architectures, and service-oriented architectures. Each approach influences how tasks are implemented and made available to the user, and also the effectiveness of the overall

---

3.2 Technical framework for tasks

system (Bass et al., 2003).

A task in a monolithic application is a piece of code to carry out some function, which has been tightly coupled with the graphical user interface (GUI). GUI elements such as drop-down menus and icons provide the user with the means to access and execute a particular task. This approach results in a single program which is self-contained and independent. The tools, and therefore the tasks, are run locally, and can only be accessed via the GUI.

Client-server implementations separate the application logic from the user interface, for example, in a web-based tool. Here, the tasks reside on a server, while the client is a website with a form, through which the user sends data to the server. The server can then respond to requests, sending a response back to the client. Unlike the single-tier model, the tasks and user interface are separated; however the tasks themselves are still only accessible via a prescribed client. Several examples of network analysis tools built both this way and as standalone tools were given in Chapter 2.

A service-oriented architecture, meanwhile, focuses on the development of tasks as services or methods which are made available to users through a standard interface. Services are created to be consumed by client applications whose only requirement is the ability to process the standard interface description, without requiring knowledge of the implementation details of the service (Arsanjani, 2004). Web services are one particular technology which may be used to populate a service-oriented architecture. The formal definition of a web service as given by the W3C is ‘a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format’ \(^1\). They are similar to the web-based client-server tools in that data are sent and received between a client application and a method residing elsewhere. The key difference is that web services have a public interface which enables access via a greater variety of externally developed clients. They can be thought of simply as callable routines made available over a network, which is the commonest style of web-service use, particularly in the bioinformatics domain (Pagni et al., 2008).

A service-oriented architecture, populated by web services, is the approach taken to realise the tasks described in the network-specific conceptual framework. There are various advantages offered by web services over monolithic and client-server systems. Web services are located and executed remotely, and so a user is not limited to locally available computing power. This also means the user is not responsible for installing, updating and patching service software. A service-oriented architecture is characterised

\(^1\)http://www.w3.org/TR/ws-gloss/
3.2 Technical framework for tasks

by loosely-coupled components, leading to a low level of interdependency between tasks. Tightly coupled components lead to a high level of interdependency, resulting in an architecture which is harder to maintain and reuse.

A potential disadvantage of implementing tasks as web services middleware is that they are designed to provide programmatic access to data and applications. This may dissuade biologists more accustomed to the traditional GUI elements included in the tools described in Chapter 2, from whom such middleware, if used, tends to be hidden. Using a web service may require more expert knowledge, for example awareness of a particular SOAP programming library. Remote invocation means that if a particular web service is altered, the user is forced to accept the change even if they do not wish to, and if a network connection should fail, for example due to faulty hardware, then this will result in a web service becoming unusable.

Despite these drawbacks, implementing tasks as web services offers a much greater degree of flexibility, as the public interface to a service enables a much wider range of client applications to be constructed. As they are platform- and language-independent, web services should theoretically be accessible by clients executed on heterogeneous platforms, without modifications to the service code, leading to a greater degree of code reuse. The biggest advantage offered over the single-tier and client-server approaches is that tasks as web services are much better suited for inclusion in automated computational workflows.

Figure 3.5 illustrates the technical framework for tasks developed in this project. At this stage, manual workflows are considered; automated workflows are discussed in Section 3.3. A single web service may be created such that it corresponds to one task, or may be composed of several tasks, depending on exactly how web services are deployed. Details of various web service deployment technologies and the corresponding implementation of tasks are discussed in greater detail in Chapter 4.

As per the network-specific framework proposed, web services can be grouped further into four categories based on the functionality offered. Each web service may be thought of as a component of a larger application, constructed by the user according to certain ordering constraints. Furthermore, web services may be grouped according to the individual or organisation responsible for creating and maintaining them. Figure 3.5 also depicts these organisational groupings. These groupings have various implications for a user of these services. Quality of service is an important consideration for organisations, if they wish to increase the popularity of their services. This includes ensuring that service discovery is maximised through suitable publication of service location, and that reliability is high through assured delivery of data being sent.
3.2 Technical framework for tasks

Figure 3.5: Technical framework for tasks. Tasks are implemented as web services. Each web service may correspond to a single task, or may contain several. Web services are grouped by service provider or organisation. Data are sent and received to and from a web service through web service clients. The details of the interaction between web service and client are discussed in Section 2.3.1.

and received by service users and providers (this is of particular relevance when web services operate over the Internet, which by its nature is dynamic and unpredictable). An organisation may also wish to enforce security measures on their web services, by only allowing access for a certain group of users, in which case authentication and data encryption should be implemented. Performance is another issue, particularly for web services which are designed to process large amounts of data. There may be multiple instances of a particular service from various organisations, and so the user may wish to find an instance which is geographically closer (though this does not necessarily guarantee the best performance).

3.2.1 Web service protocols and standards

Web services are defined by a set of specifications that are used to implement, describe and locate them. The core standards are the Service Oriented Architecture Protocol or Simple Object Access Protocol (SOAP, Gudgin et al., 2007), Web Services Definition Language (WSDL, Christensen et al., 2001) and Universal Description, Discovery and Integration (UDDI, Clement et al., 2004). All three protocols are based on eXtensible
3.2 Technical framework for tasks

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Description</th>
<th>Protocols Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Transport</td>
<td>Responsible for transporting messages between network applications</td>
<td>HTTP, SMTP, FTP</td>
</tr>
<tr>
<td>Messaging</td>
<td>Responsible for encoding messages in XML format</td>
<td>XML-RPC, SOAP</td>
</tr>
<tr>
<td>Service Description</td>
<td>Used to describe the public interface to a specific web service</td>
<td>WSDL</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>Centralises services into a common registry such that network web services can publish their location and description, and makes it easy to discover what services are available</td>
<td>UDDI</td>
</tr>
</tbody>
</table>

Table 3.1: The Web Service Protocol Stack

Markup Language (XML, Bray et al., 2006), which is a platform-independent, general purpose markup language with user-defined tags, designed to facilitate the sharing of data between heterogeneous systems.

The web services protocol stack in Table 3.1 shows the four layers and related protocols, while Figure 3.6 illustrates the relationship between the actors in the web service model.

The steps involved in publishing and consuming a web service are as follows:

1. A service provider describes a web service using WSDL. The WSDL defines the location of the service and the number of operations (i.e. tasks) exposed by the service. The definition is published to a registry of services, which could use UDDI or could be of another form.

2. A service requestor, i.e. the user (in this case a biologist or bioinformatician seeking to carry out some network-analysis tasks) issues a query to the registry to locate a web service, and the operations offered by it.

3. Part of the WSDL is passed to the service requestor, describing the requests expected by each operation and the responses returned from it.

4. The service requestor uses this information to send a request, to the appropriate operation.
3.2 Technical framework for tasks

Figure 3.6: Communication over a network between service provider, service consumer and registry in the web services model
5. The operation carries out the request and sends a response back to the user.

Web services and the associated technologies and protocols may be described in terms of the technical framework for tasks. WSDL is the public interface used to define tasks, and is processable by a suitable client application. Data are passed between a client and the task using SOAP; while this is an XML-based messaging protocol, data are sent in whatever format a task understands. For example, the common graph format is encapsulated in an XML message as text. While SOAP and WSDL have endured as core web services standards, UDDI (a business-oriented protocol for service discovery) is no longer used. In the bioinformatics domain, a more informal style of registry is used, for example, websites listing the location of WSDL files.

3.3 Technical framework for workflows

An important element of the conceptual framework is the ability to construct workflows of tasks. In silico experimentation may involve the use of a number of computational tasks and access to databases made available by different providers. A record of the stages involved is analogous to a protocol in wet-lab research. It has been demonstrated in Section 3.2 that there are a number of ways that tasks may be made available to the user, which influences the way in which workflows are realised.

Tasks implemented within all three of the technologies discussed may be used to construct manual workflows. Manual workflows require constant interaction from the user, as once input data are submitted to a particular type of task, it must be monitored until the result is produced. The result is then exported and submitted to one or more other tasks, which are similarly monitored, until the desired final output is obtained. With the onus on the user to move data between different processing steps, such workflows are labour-intensive and error-prone, and every stage should be manually recorded if the in silico experiment is to be reproduced. Figure 3.7 shows an example of a manual workflow, involving tasks implemented within web-based client-server tools. Execution of the protocol involves not only the use of the tools themselves, but intermediate data export and transfer into external files to store results.

A much more efficient approach is to consider workflows as computational entities, created using tasks developed as part of a service-oriented architecture, which is the recommendation of the technical framework for tasks described previously. Computational workflows are desirable as they enable automatic software pipeline construction and execution, without the need for repeated user-intervention. One way this could
3.3 Technical framework for workflows

1) Enter sequence into Blast form

2) Copy the 5 top hits into a text file

3) Visit Gene Ontology website and enter each Swissprot accession one by one

4) Click link to retrieve GO term associations

Bcap31

5) Copy and paste GO terms...

6) ... into a new text file to collate results

Figure 3.7: An example of a manual in silico experiment, to help characterise a protein by finding similar sequences using BLAST, and their associated Gene Ontology terms.
be implemented in practice is to create a computer program written in a language with web-service support. This would contain a series of web-service calls, with extra code to handle data transfer and storage of intermediate results. A more sophisticated approach is the utilisation of workflow management software, to handle the discovery, invocation and execution of web services, and creation and enactment of workflows. The most effective workflow software must be able to facilitate discovery of resources and handle transfer of data between them.

Various open-source scientific workflow management packages are available, for example Kepler (Altintas et al., 2005), Triana (Majithia et al., 2004) and Wildfire (Tang et al., 2005). For this work, however, workflow design and enactment functionality are provided by the Taverna workbench (Oinn et al., 2000). Taverna is developed as part of the myGrid consortium (Stevens et al., 2003), an initiative that seeks to develop a loosely-coupled, service-based suite of open-source middleware for bioinformatics. Taverna was chosen for this project for a number of reasons: it is open-source and supported by a very active development team who encourage input and feedback from the user community. The Taverna user interface is designed to make the construction of workflows more accessible to those users who may not necessarily be expert programmers or be familiar with web services. A variety of web-service styles are supported in Taverna, from those described using the WSDL standard, to life-science-specific projects such as BioMoby and Soaplab. Also, scheduling of processor execution in Taverna is simple: processors are executed as soon as possible, relieving the user of the need to explicitly define the control flow that determines order of execution.

The Taverna workbench may be freely downloaded for Windows, Mac and Linux from the project homepage\(^1\). The version used throughout this work is 1.7, for the Linux platform. Workflows in Taverna are described using the Simplified Conceptual workflow language (Scufl), which is a high-level XML-based language. The components of a Scufl workflow can be described in terms of the framework:

- **Processors** A processor is an individual step in a workflow, and is analogous to a task as described in the network-specific conceptual framework, either belonging to one of the categories, or responsible for generic control. There are a number of different processor types which are used to present a common interface over heterogeneous interfaces, and a complete list of processor types is given in Oinn et al. (2004). Processors in Taverna are ‘scavenged’, for example, adding a WSDL scavenger into Taverna results in all the port types and operations in this WSDL

\(^1\)http://taverna.sourceforge.net/
becoming visible. Each operation can now be added into the workflow as a WSDL processor.

- **Inputs and outputs** Inputs and output data entities can be considered as being source and sink processors, respectively. A source processor makes an input value available on its virtual output port, and a sink processor receives a value from its virtual input port. The values of inputs and outputs are the data described in the network-specific conceptual framework.

- **Data links** A data link connects a source processor or output port of a processor to a sink processor or input port of a processor. Figure 3.4 in the network-specific conceptual framework illustrates the concept of linking the inputs and outputs of a set of tasks to form a workflow through which data are passed.

- **Coordination links** A coordination link is used to provide additional constraints on linked processors. For example, two processors may not have a data dependency, but should still execute in a particular order. While this concept is not explicitly defined in the framework, it is useful functionality which may be relevant for certain workflows.

Figure 3.8 shows the three components of the software: the Available Processors pane, the Advanced Model Explorer (AME) pane and the Workflow Diagram pane. The user may add processors to the Available Processors pane by entering the endpoint of a web service. As a workflow is constructed in the AME by adding processors, data links, co-ordination links and inputs and outputs, a pictorial representation is produced in the Workflow Diagram pane. The actual execution of a workflow is handled by the FreeFluo enactment engine. This is a Java workflow-orchestration tool which supports Scufl, though is language-independent. Figure 3.9 illustrates the technical framework for workflows. A detailed Taverna tutorial, which covers the setting up and installation of the software, as well as building and running a simple workflow, is available in Appendix A.

Taverna implements two mechanisms which facilitate effective workflow construction. One is implicit iteration. Where a processor expects a single input, but is passed a list of inputs, it iterates over each input, and produces a list of results, where each corresponds to an input from the list submitted. Another useful mechanism is the ability to insert nested workflows into a workflow. Nested workflows are beneficial as they enable commonly repeated sequences of processes to be saved and inserted into other workflows, as and when required, analogous to subroutines or methods in a program.
The network-specific conceptual framework described previously highlighted the need for a common graph format to abstract over the various heterogeneous formats for recording biological network data. An additional consideration relating to data as part of the framework, is the availability of biological network and interaction data from various public repositories. This may be through either a web service interface compatible with Taverna, or simply downloadable files in a format which is suitable for submission to the framework. Table 3.2 describes some of these databases.

3.4 Conclusion

The framework has been designed to promote reusability, extensibility and flexibility. Web services are highly reusable as they are self-contained computational routines which may be used repeatedly in different workflows. The proposal of different categories of web service also contributes to reusability, for example, services in the data-analysis category should be applied without modification to network data represented in the common graph format. The Taverna client ensures that workflows themselves are also highly reusable, as they may be saved, stored and distributed between interested scientists as required. The myExperiment initiative (De Roure and Goble, 2009) enables users to download pre-constructed, annotated workflows, shared globally or within research groups.

The latest version of Taverna to be released is 2.0, which introduces a redesign of the interface and certain improvements in performance over Taverna 1.7. However not all workflows developed in version 1.7 are compatible with the newer version, and there remain a number of unresolved problems\(^1\). Therefore all workflow functionality described in this chapter relates to version 1.7.

Extensibility is an important factor when designing software. The framework enables this by providing network analysis tasks as web services, rather than as part of a monolithic application or client-server tool: any interested individual may wish to add functionality by writing their own web services to be combined with those that exist already, in order to generate new workflows. Web services developed must therefore have well-defined inputs and outputs, to facilitate the creation of extensions.

The framework demonstrates flexibility by allowing users to access network analysis functionality through their preferred type of client, by making a public interface to the service available. The introduction of the sophisticated Taverna client extends

<table>
<thead>
<tr>
<th>Data source</th>
<th>Type of network data</th>
<th>Web service interface (type: location)</th>
<th>Export formats(s)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOND (Biomolecular Object Network Databank)</td>
<td>PPI</td>
<td>WSDL: <a href="http://soap.bind.ca/wSDL/bind.wSDL">http://soap.bind.ca/wSDL/bind.wSDL</a></td>
<td>SIF, PSI-MI 2.5</td>
<td>Alfarano et al. (2005)</td>
</tr>
<tr>
<td>IntAct</td>
<td>PPI</td>
<td>WSDL: <a href="http://www.ebi.ac.uk/intact/binary-search-ws/binarysearch?wSDL">http://www.ebi.ac.uk/intact/binary-search-ws/binarysearch?wSDL</a></td>
<td>PSI-MI 1.0, PSI-MI 2.5, PSI-MI TAB</td>
<td>Kerrien et al. (2007)</td>
</tr>
<tr>
<td>MINT (Molecular Interaction database)</td>
<td>PPI</td>
<td>Not available</td>
<td>PSI-MI 2.5, PSI-MI TAB, MINT tab-delimited</td>
<td>Chatr-aryamonti et al. (2007)</td>
</tr>
<tr>
<td>BioCyc</td>
<td>Metabolic</td>
<td>Not available</td>
<td>SBML, BioPax, BioCyc tab-delimited</td>
<td>Caspi et al. (2008)</td>
</tr>
<tr>
<td>MIPS MPact</td>
<td>PPI</td>
<td>WSDL: <a href="http://mips.gsf.de/proj/hobotws/services/PsimiService?wSDL">http://mips.gsf.de/proj/hobotws/services/PsimiService?wSDL</a></td>
<td>PSI-MI 2.5</td>
<td>Guldener et al. (2006)</td>
</tr>
<tr>
<td>HPRD (Human Protein Reference Database)</td>
<td>PPI</td>
<td>Not available</td>
<td>PSI-MI 2.5, HPRD tab-delimited</td>
<td>Peri et al. (2004)</td>
</tr>
<tr>
<td>DIP (Database of Interacting Proteins)</td>
<td>PPI</td>
<td>Not available</td>
<td>XIN, PSI-MI 2.5, PSI-MI TAB, DIP tab-delimited</td>
<td>Xenarios et al. (2000)</td>
</tr>
</tbody>
</table>

Table 3.2: A selection of biological network data repositories
this flexibility by enabling the construction of complex queries over relevant tools and databases. The interconnected tasks in a workflow may be available as local or remote web services, potentially developed on heterogeneous platforms in geographically diverse locations.

The framework is realised using two relatively new technologies, web services and computational workflows via Taverna. There are potential limitations to these recommendations: web service creation and access requires specialised programming knowledge, and the concept of workflow construction may not be immediately accessible for certain users. It has been suggested, however, that workflows are constructed by expert bioinformaticians who are familiar with the technologies involved, and enacted by less expert users (Egglestone et al., 2005). These users may be able to evaluate the biological significance of any results, and suggest modifications to the workflow accordingly.

This chapter has established how tasks and workflows will be practically realised. The following chapter recounts experience gained from deployment of tasks as web services using four open-source deployment technologies. As the framework specifies the exposing of computational tasks as web services, each technology will be evaluated in terms of how exactly this is achieved, and also how Taverna uses a processor plugin to make tasks deployed via each method available for use in automated workflows.
Figure 3.8: The three components of the Taverna workbench. (a) Right-clicking on Available Processors results in a drop-down menu in which the different scavengers are displayed. Selecting one of these prompts the user to enter a location, or end-point, of a web service, which if valid displays an expandable list of processors which can be added to a workflow. (b) The components of a Scufl workflow are listed in the Advanced Model Explorer; as they are added they appear in (c), the Diagram pane, in the appropriate position. Processors are different colours depending on their type, inputs and outputs are denoted by red and green triangles respectively, and data flow is shown with arrows connecting workflow components. In this example, the green processor denotes a standard SOAP service, and the purple processors are local Java operations representing generic functionality, in this case XML splitters which aggregate or split the inputs and outputs to a processor.
Figure 3.9: Technical framework for workflows. The Taverna workbench uses processor types to abstract over the different service interfaces, where each processor corresponds to a task. A Scufl definition of a workflow defines a set of processors between which data are passed, via the Freefluo engine. The user is therefore able to select from a pool of processors and construct computational workflows through the workflow client, which handles the discovery, orchestration and execution of the graph of tasks.
Chapter 4

Technology Evaluation

4.1 Introduction

The framework in Chapter 3 recommended the use of web services (which ultimately will be used in computational workflows), to implement tasks within the domain of biological network construction and analysis. As a service provider, there are various aspects of web service creation which influence the decision regarding which deployment method to use. The aim of this chapter is to present the practical experience gained from the evaluation of four web service deployment technologies. These technologies were evaluated against a number of criteria and test cases, used to justify the final choice used by the author to develop web services for this work.

All the technologies were evaluated under Linux (CentOS 5.3 64-bit).

4.2 Criteria and test cases for evaluation

The following criteria were applied to each deployment technology:

- **Availability** As web services themselves are based on open standards and protocols, open source deployment packages are considered preferable. A review by Stajich and Lapp (2006) concludes that there is a general trend in bioinformatics towards open-source technologies, producing tools which can be “continuously improved in their usefulness”, and that “freely available and modifiable open-source software can serve as the foundation for building important applications, analysis workflows and resources”.

- **Installation** While this procedure will usually only be performed once, the in-
stallation process should be clearly described, together with a complete list of
hardware and software prerequisites, and any configuration which should be car-
ried out within the development environment.

- **Support** The technology should be backed up by robust support mechanisms, for
eexample, active user mailing lists and example code.

- **Web service protocols supported** As described in the previous chapter, web ser-
  vices are defined by a set of open protocols. Adhering to these protocols ensures
  a greater degree of interoperability between services developed using different
  technologies.

The following test cases were applied to each deployment technology:

- **Implementation of tasks as web services** Two tasks were designed as part of a basic
  Calculator service: `add` and `subtract`, which calculate the sum and difference of
  two numbers respectively. This test case was designed to establish how tasks are
  implemented within each deployment technology, and to highlight any drawbacks
  and/or particular advantages.

- **Invoking the service - built-in client** Web service toolkits generally include support
  for client- as well as server-side SOAP. If this is the case for the technology under
  evaluation, the tasks in the Calculator service were first tested with the client
  library. This test case was designed to check that the web service is correctly
  handling and returning data.

- **Invoking the service - Taverna client** The framework specifies that the web ser-
  vices created for this work should be discoverable from within the Taverna work-
  bench, so that they may be used within automated workflows together with other
  services. As previously discussed, a task in the framework is analogous to a Tav-
  erna processor, therefore the tasks `add` and `subtract` must have a public interface
  which is compatible with one of the processor types used in Taverna (e.g. WSDL).

4.3 Evaluation

4.3.1 SOAP::Lite

SOAP::Lite is a Perl module designed to be an interface to SOAP on both the client
and server side. It is currently maintained by Martin Kutter. The version tested here
is 0.71.
Availability

The module is freely available from CPAN (http://search.cpan.org/~mkutter/SOAP-Lite-0.710.08/).

Installation

As a root user, the module may be installed using the CPAN module, which also takes care of the dependencies. If root access is unavailable, the module can be downloaded as a .tar.gz file and installed in a specified directory using the PREFIX keyword. Full details of this procedure are given in Appendix B.

A CGI-based SOAP server also requires the installation of a web server capable of running Perl-based CGI scripts, such as the freely available Apache (http://httpd.apache.org/).

Support

Links to mailing lists for developers and users are available from the module homepage, http://www.soaplite.com/. This page also hosts a ‘SOAP cookbook’, which is a comprehensive resource addressing various issues, and a user guide which is incomplete but does contain several useful examples. The module suffers from relatively poor documentation but the mailing list is reasonably active.

Web service protocols supported

SOAP and WSDL (though WSDL support is limited).

Implementation of tasks as web services

A SOAP::Lite CGI-based server consists of two components, the request handler and dispatcher. The request handler contains the core application logic exposed as a web service. It is simply a Perl module containing subroutines, each of which correspond to the tasks described in the framework. For the Calculator service, the module contains two subroutines, add and subtract.

The dispatcher is the part of the service directly exposed to the client invoking it. It binds the SOAP request to the class specified. Dispatch may take one of the following forms:

1. Static internal - the dispatcher and handler are located in the same script
2. Static external - the handler is located outside the server (dispatcher) code, and the Perl statement \texttt{use lib} points to the location of the module.

3. Dynamic - a directory is specified in the dispatcher rather than the module name, so that any module added to this directory becomes available for dispatch.

The dispatcher should be located in the \texttt{cgi-bin} directory on the web server. Permissions should be set to make it executable, or service unavailable (503) errors will be returned by the client. Once both the dispatcher and handler are placed in the correct location within the web server, the web service is ready to be invoked.

Example code for the dispatcher and handler for the Calculator service can be seen in Appendix B.

**Invoking the service - built-in client**

The SOAP::Lite toolkit has a client library. An example client for the Calculator service, which accesses the \texttt{add} task, is shown in Listing 4.1.

```perl
#!/usr/bin/perl -w
use SOAP::Lite;
my @values = (10,5);
print SOAP::Lite
  -> uri('http://behemoth.mycib.ac.uk/Calculator')
  -> proxy('http://behemoth.mycib.ac.uk/cgi/sirisha/calculator.cgi')
  -> add(@values)
  -> result;
```

Listing 4.1: client.pl

The \texttt{proxy} is the actual address of the SOAP server, or dispatcher. The \texttt{uri} refers to the namespace that the service responds to, and corresponds to the module name. Each SOAP server can offer multiple services through one proxy location, so the \texttt{uri} is used to identify a particular service. Values may then be passed to the individual web service operations, or tasks, to be processed.

**Invoking the service - Taverna client**

SOAP-based communication between clients and services which are both written in Perl is straightforward without the need for a structured definition of the service, as long as
the client is aware of which types to send and receive from the SOAP server. However, as Perl is dynamically typed, it is very hard to extract information from the code about the types, number of parameters and return values of methods, if access is required by a client written in a statically typed language. To enable communication between the Taverna client and the Calculator example in SOAP::Lite therefore requires the creation of a WSDL document describing the Calculator service, so that the operations \texttt{add} and \texttt{subtract} may be scavenged and added to workflows.

SOAP::Lite does not support automatic WSDL generation, so the service provider must generate this themselves. This may be done manually, but this is inadvisable owing to the complexity of WSDL. For a large number of services this quickly becomes very cumbersome. One solution is to use the Pod::WSDL module, freely available from CPAN\(^1\). The version used for this example is 0.05. The module was used to generate WSDL based on a Plain Old Documentation (POD) file which describes the subroutines that constitute the application logic of the web service. The POD should directly precede the subroutines. Appendix B contains a modified version of the original subroutine code for the Calculator service to demonstrate this, together with the generated WSDL file.

The WSDL file, once made available on the WWW, can be scavenged from within Taverna. Figure 4.1 shows the two new WSDL processors in the Available Processors list, then used in a simple workflow.

### 4.3.2 Apache Axis

The Apache Axis toolkit is a Java web service framework consisting of an implementation of a SOAP server, a client library, and various utilities and APIs for generating and deploying web services. The version tested here is 1.4.

**Availability**


**Installation**

Axis is installed within a servlet container, such as the freely available Apache Tomcat ([http://tomcat.apache.org/](http://tomcat.apache.org/)). Once downloaded and extracted, the \texttt{axis} directory from the distribution is copied into the \texttt{webapps} directory under Tomcat. Successful installation can be checked by navigating to the Axis homepage in a web browser.

\(^1\)[http://search.cpan.org/dist/Pod-WSDL/](http://search.cpan.org/dist/Pod-WSDL/)
4.3 Evaluation

Figure 4.1: WSDL for the SOAP::Lite Calculator service scavenged from within the Taverna workbench. The add operation is used to create a simple workflow.

The URL takes the form \texttt{http://<tomcat-host>:<tomcat-port>/axis/}. This webpage is likely to initially report errors regarding missing components. In particular one ‘required’ component, Activation API, should be downloaded separately\textsuperscript{1} and placed in the \texttt{lib} directory under the \texttt{axis} directory. Tomcat should be restarted for this change to be recognised.

Two ‘optional’ JAR files, Mail API\textsuperscript{2} and XML Security\textsuperscript{3} may not be immediately necessary, but may be downloaded and copied into \texttt{lib} as before, whenever required.

All the JAR files in \texttt{lib} must then be added to the \texttt{AXISCLASSPATH} environment variable to ensure Java can locate the necessary files when carrying out deployment activities and enabling client access.

Support

Support is available via a comprehensive user guide and active mailing list.

\textsuperscript{1}\url{http://java.sun.com/javase/technologies/desktop/javabeans/jaf/downloads/index.html}
\textsuperscript{2}\url{http://java.sun.com/products/javamail/}
\textsuperscript{3}\url{http://santuario.apache.org/}
4.3 Evaluation

Web service protocols supported

SOAP and WSDL.

Implementation of tasks as web services

The simplest approach to web service creation is to expose Java classes by placing Java source code in the root axis directory, changing the extension from .java to .jws. Axis automatically compiles the class and converts the SOAP calls correctly into Java invocations of the service class. Tasks are therefore public Java methods of a class, which are exposed as the operations of a web service. Appendix B contains the source code for a .jws version of the Calculator example.

Once the source code is placed in the root axis directory, it should be possible to navigate to the service location in a browser, using a URL which takes the form 


While JWS services are a very convenient and fast way to expose Java code as a web service, they are inflexible and offer limited functionality. Only source code can be used for deployment, which is not ideal if the service provider only has access to a compiled class. As the code is compiled at run-time, errors are not detected until after deployment. Also, packages are not supported.

A more powerful approach that offers greater flexibility makes use of a Web Service Deployment Descriptor (WSDD). The deployment descriptor contains metadata about a web service that is to be made available to the Axis engine. An example WSDD for the Calculator service is shown in Listing 4.2.

```xml
<deployment xmlns="http://xml.apache.org/axis/wsdd/
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="Calculator" provider="java:RPC">
    <parameter name="className" value="Calculator"/>
    <parameter name="methodName" value="*"/>
  </service>
</deployment>
```

Listing 4.2: deploy.wsdd

To use a WSDD, the Java source must be compiled, and the resulting class is given as a parameter of the service (className) in the WSDD. Other parameters are available, for example allowedMethods which tells the Axis engine which methods (i.e. tasks) in the code to make available as operations of the service.
4.3 Evaluation

The deployment is carried out using the AdminClient which is packaged with Axis. The AdminClient is executed in a terminal as follows:

```
```

axis/services/AdminService deploy.wsdd

Successful deployment is indicated in the terminal with the `<Admin>Done processing</Admin>` message. This can be tested by clicking the ‘Available Services’ link displayed on the Axis homepage: the new Calculator service together with the available methods `add` and `subtract` will appear, as shown in Figure 4.2. It is important to note that if the class file of the service is placed outside of the `classes` directory under `axis`, then the appropriate package name must be specified in the Java source, otherwise the Axis engine will be unable to locate the class.

![Figure 4.2: The ‘Available Services’ links leads to a web page as shown in this figure. The new Calculator service is listed along with the associated methods.](image)

**Invoking the service - built-in client**

For services deployed using either the ‘drag and drop’ or deployment descriptor method, WSDL is generated automatically by appending `?wsdl` to the end of the unique service URL associated with that service. Client libraries within Axis can be used to construct clients that consume web services based on both their JWS endpoints or WSDL. Appendix B contains an example client to consume the WSDL for the Calculator service.
4.3 Evaluation

Invoking the service - Taverna client

As described, the automatic generation of WSDL means that the service provider does not need to do any further work to enable compatibility with Taverna. The WSDL can be scavenged, making the WSDL processors add and subtract available for use in workflows, as shown in Figure 4.3.

![Figure 4.3: WSDL for the Apache Axis Calculator service scavenged within the Taverna workbench. The subtract operation is used to create a simple workflow](image)

4.3.3 Soaplab1

Soaplab (Senger et al., 2003) has been developed by Martin Senger, and is a mechanism for web service development which wraps existing legacy applications as services. For this work, the original release (Soaplab1) was evaluated, though it has since been superseded by Soaplab2 (Senger et al., 2008).
4.3 Evaluation

Availability
Soaplab1 is freely available from Sourceforge\(^1\) as a zipped package.

Installation
The prerequisites for installation are Perl, Java and Apache Axis (installed within a servlet container such as Apache Tomcat, as previously described). The Soaplab1 package is unzipped to create the top-level directory `analysis-interfaces`. Installation is carried out by running a Perl script, `INSTALL.pl` which is located in this directory. This is an interactive installation process during which several environment variables are set, including the directory in which Tomcat is located, the URL used to access Tomcat and the location of the `lib` directory within Axis (so that Soaplab JAR files can be copied there). The installation script also gives the option of adding directories to the PATH, which is where applications should reside in order to be executed when invoked as web services.

Support
The project homepage at [http://soaplab.sourceforge.net/soaplab1/](http://soaplab.sourceforge.net/soaplab1/) contains a comprehensive user guide with example code.

Web service protocols supported
SOAP and WSDL.

Implementation of tasks as web services
Tasks in Soaplab are individual executable programs which are each wrapped as a web service, rather than operations collected together within a single web service. For the Calculator example, two executable Perl scripts were created, each encoding functionality for the `add` and `subtract` tasks.

Development of services using Soaplab1 is a multi-step process. The first stage involves the creation of metadata to describe the command line of the executable to be wrapped as a service. These metadata are described using the Ajax Command Definition (ACD) language, which originated as part of the EMBOSS project (Rice \textit{et al.}, 2000). Two ACD files corresponding to the `add` and `subtract` programs were

\(^1\)\url{http://sourceforge.net/project/showfiles.php?group_id=104834&package_id=112781&release_id=335757}
written and placed in the metadata directory, while the scripts themselves should be placed in a location specified in the PATH (established during installation). Appendix B shows the program code and corresponding ACD file for the add executable.

The ‘groups’ token in the ACD file is used to categorise and organise services. As add and subtract are different separate services rather than separate operations of a service, they are both assigned the group ‘Calculator’ to identify them.

The ACD is transformed into XML files used during deployment, by the acd2xml tool. This is executed from the top-level Soaplab1 directory as follows:

```
./generator/acd2xml -d add subtract -l Applications.xml
```

The .acd extension is not required when the ACD files are passed as arguments to the generator. The -l flag generates an XML file which lists all the executables to be deployed, and is used by the AppLab server. Applications.xml is the default name for this file, and is already specified in the run-AppLab-server script. If a different file name is passed after the -l flag, it must be changed accordingly in run-AppLab-server.

The AppLab server is a Java application accessible using a CORBA interface, and is responsible for communication with the underlying executables which are wrapped as web services. Before any services are deployed, the AppLab server should be started with the following command, executed from the top-level directory:

```
./run-AppLab-server
```

Execution of this command generates the files which are used by the Soaplab server to communicate with the AppLab server, which uses a launcher (a set of Perl scripts) to invoke the add and subtract executables.

The deployment itself may now proceed, by executing a script named deploy-web-services. An example command to deploy services is as follows, and can be customised using command line options:

```
./ws/deploy-web-services -a -d -j derived.jar
```

The -a flag ensures only AppLab services are deployed (rather than Gowlab services, a sub-project of Soaplab that wraps websites as web services). The -d flag is used to generate derived services which have strongly-typed methods, useful if the WSDL is needed to access the service. The -j flag is always used in conjunction with -d, and is followed by the name of a JAR file containing the generated derived services. During execution of deploy-web-services, the service provider is prompted to restart Tomcat. Successful completion of the process is indicated via messages in the terminal.
4.3 Evaluation

Invoking the service - built-in client

Soaplab1 does not provide a client library such as those provided by Axis and SOAP::Lite. However a powerful Java command-line client, run-analysis-client, is included with the distribution. This has a large number of command line options when invoking a particular service. The client is used as follows:

```
./run/run-analysis-client <find-arguments> [options] [inputs] [results]
```

A detailed description of the <find-arguments>, [options], [inputs] and [results] flags are given by executing:

```
./run/run-analysis-client -h
```

An example execution of the client for the add service is as follows:

```
./run/run-analysis-client -e http://compute1.mycib.ac.uk:8080/axis/services -name calculator.add int1 10 int2 5 -w -r
```

The -e flag specifies the location of the installation, -name the service name (which takes the form [group.name]), and the argument names int1 and int2 correspond to those specified in the ACD file. The -w flag specifies that the job be created, started and run until completion (either successfully or unsuccessfully). The -r flag indicates that results should be returned.

Invoking the service - Taverna client

As Soaplab1 services are deployed within the Apache Axis engine, it is possible to navigate to the service location in a browser and view the automatically generated WSDL, which is discoverable from within Taverna as previously demonstrated. However support for Soaplab1 is provided via the Soaplab1 scavenger, which enables the discovery of a Soaplab installation via the URL of the Soaplab1 services. This takes the form

```
http://<tomcat-host>:<tomcat-port>/soaplab/services
```

Each Soaplab1 service is available as a processor which may be used in a workflow.

4.3.4 Soaplab2

The premise of Soaplab2 remains the same as for Soaplab1, that is, the wrapping of legacy applications to expose them as web services. Despite an almost complete internal re-write, from the perspective of service users, interaction is very similar. There are some key differences for service providers however, which are explained here.
4.3 Evaluation

Availability

The software is freely available from Sourceforge\(^1\).

Installation

Installation and building (as well as deployment) is handled by the Apache Ant\(^2\) tool, which should be version 1.6.5 or later. A servlet container such as Tomcat is also required. Soaplab2 supports two protocols for deploying web services: the original Apache Axis protocol (there is no need to separately download Axis as it is bundled with Soaplab2), and the new protocol which uses Java API for XML Web Services (JAX-WS)\(^3\).

Soaplab2 is downloaded as a zipped package, which when extracted creates the top-level directory soaplab2. Before building, there are some configuration steps which should be carried out, which are explained in detail in Appendix B.

Ant may now be used to build Soaplab2. If behind a firewall, the environment variable `ANT_OPTS` should be set as follows (when using bash):

```bash
export ANT_OPTS="-Dhttp.proxyHost=<proxy-host> -Dhttp.proxyPort=<proxy-port>
```

The command `ant install` should be executed from within the top-level directory to start the build process. Successful completion is indicated by a terminal message. To test the build, executing `ant jaxdeploy` deploys a few testing services.

Support

The project homepage, [http://soaplab.sourceforge.net/soaplab2/](http://soaplab.sourceforge.net/soaplab2/) has a comprehensive user guide.

Web service protocols supported

SOAP and WSDL.

Implementation of tasks as web services

As for Soaplab1, deployment of an executable as a web service requires an ACD file describing the command line of the executable. The same scripts and ACD files from Soaplab1 can be used to deploy the `add` and `subtract` services using Soaplab2, following these steps:

\(^1\)http://soaplab.cvs.sourceforge.net/soaplab/soaplab2/  
\(^2\)http://ant.apache.org/  
\(^3\)https://jax-ws.dev.java.net/
4.3 Evaluation

- The ACD files are copied to the \texttt{src/etc/acd/sowa} directory under the \texttt{soaplab2} directory
- The executables are copied to the \texttt{run} directory under the \texttt{soaplab2} directory
- \texttt{ant gen} is executed to create \texttt{Soaplab2} run-time XML metadata from the ACD files
- \texttt{ant jaxdeploy} is executed to deploy the services to Tomcat using the JAX-WS protocol. This copies all the relevant files (executables and metadata) to the Tomcat container
- \texttt{ant axis1deploy} is executed to deploy the services to Tomcat using the Axis protocol

**Invoking the service - built-in client**

\texttt{Soaplab2} is packaged with a Java client program \texttt{run-cmdline-client} which is very similar to the \texttt{run-analysis-client} program in \texttt{Soaplab1}. This may be used to invoke a service to check if the deployment was successful. The client is executed as follows, to call the \texttt{add} service:

```
./build/run/run-cmdline-client -name calculator.add int1 4 int2 5 -w -r sum
```

where \texttt{-name} and \texttt{-w} are as before, whereas the \texttt{-r} flag is followed by the output name \texttt{sum}, so that only the output is returned instead of the full report:

```
Job ID: [calculator.add]_3c2efc54.11eee414f72._7ff5
sum
---
4
```

Another included client is the web-based Spinet tool which enables the discovery and execution of web services through a web form. This is a very convenient way to test the successful deployment of \texttt{Soaplab2} services. Figure 4.4 shows the Spinet client with the input form for the \texttt{add} service. The URL for the Spinet client takes the form \texttt{http://<tomcat-host>:<tomcat-port>/soaplab2/}, and for each service automatically generates a form labelled with the appropriate input, and a button to click which starts execution of the service. The results are then viewable in another web page (as specified during the configuration steps prior to building).
4.3 Evaluation

Figure 4.4: The Spinet web client. Services are grouped according their category, and the form listing inputs is revealed by clicking on the service name.

**Invoking the service - Taverna client**

Soaplab2 services deployed using the Axis protocol can be scavenged from within Taverna and used in workflows in exactly the same way as Soaplab1 services. Those deployed using the JAX-WS protocol use a different scavenger, which is installed from a plugin site as follows:

- Select Tools -> Plugin Manager -> Find New Plugins -> Add Plugin Site
- Enter the following in the site URL and give an appropriate name (e.g. soaplab2 plugin): http://soaplab.sourceforge.net/taverna-plugin/
- Close the Plugin Manager
- Restart Taverna
- After restart, right-clicking Available Processors will show a new option ‘Add Soaplab scavenger (version 2)’
The above steps apply to Taverna version 1.7. To ensure the plugin works correctly, Java should be at least update 4 of version 6, and the Taverna2 plugin should be disabled. The Soaplab2 services are scavenged and added to a workflow as shown in Figure 4.5

Figure 4.5: The Soaplab2 installation is scavenged from within Taverna, and the services in the ‘calculator’ group are available to be used as workflow processors. The `add` service is used in a simple workflow.

4.3.5 BioMoby

BioMoby (Wilkinson and Links, 2002) is an open-source research project that aims to generate an architecture of discovery and distribution of biological data through web services. Data and services are decentralised, but the availability of these resources and instructions for interacting with them are all located in a central registry called MOBY Central.
4.3 Evaluation

Availability

BioMoby is freely available for download from the project website, http://www.biomoby.org/. There are two implementations, in Java and Perl, so prerequisites depend on the language chosen. Both make use of technologies already described: the Java version requires Apache Tomcat and Apache Axis, while the Perl version uses the SOAP::Lite module. The Perl version is tested here, the code for which may be downloaded from CPAN¹.

For deployment, the module MOSES-MOBY should also be downloaded² This Perl extension enables the automatic generation of BioMoby web services.

A CGI-based SOAP server also requires the installation of a web server capable of running Perl-based CGI scripts.

Installation

The MOBY and MOSES-MOBY modules are installed using CPAN, however during installation a package management utility (for example, yum) must be used in tandem, as CPAN is unable to correctly handle development headers and so reports errors when installing certain module dependencies. This is not indicated in the MOBY documentation. Once the MOSES-MOBY module is installed, it is configured by executing the interactive moses-install script.

Support

Comprehensive user support is available from the project website, which includes a number of examples and various tools for testing and exploration of the various constituent parts of MOBY Central.

Web service protocols supported

SOAP and WSDL.

Implementation of tasks as web services

BioMoby comprises an Object Ontology, a Namespace Ontology and a Service Ontology. Objects that a service can consume and produce are lightweight XML documents that conform to BioMoby object descriptions. The XML representing an Object contains

¹http://search.cpan.org/dist/MOBY/
²http://search.cpan.org/dist/MOSES-MOBY/
three piece of information: the namespace, the ID within that namespace and the data itself. An example of the simplest type of Object is as follows:

```
<Object namespace = 'NCBI_gi' id = '163483'/>
```

Namespaces are domains of ID numbers. In the above example Genbank is identified by the ‘NCBI_gi’ namespace, and 163483 is the value representing an instance of this namespace. The Object ontology enables the construction of more complex Objects, containing three types of relationship: ISA (an inheritance relationship, indicating all properties of the parent are present in the child), HASA (a container relationship with cardinality 1) and HAS (a container relationship with cardinality 1 or more). The root of the Object ontology is called ‘Object’, the base from which all BioMoby data must inherit from. Also organised in a hierarchical fashion is the Service Ontology. The only relationship currently supported is ISA (inheritance).

Each BioMoby web service corresponds to one task as described in the framework; in this example the add task is deployed. The first deployment step uses a script to register the service with the desired service registry, an example of which is given in Appendix B.

MOSES-MOBY is then used to generate the skeleton code of the service, and is executed as follows:

```
moses-generate-services.pl -v -c mycib.ac.uk simpleCalculatorAdd
```

The -v flag indicates verbose mode, and the -c flag generates a service implementation and a CGI dispatcher script. As the Perl implementation is based on SOAP::Lite, the service implementation (request handler) and dispatcher work together as previously described. For the simpleCalculatorAdd service, these two generated files are labelled simpleCalculatorAdd.pm and simpleCalculatorAdd.cgi respectively.

The CGI dispatcher is ready to be placed in the appropriate directory of the web server, but the handler must be edited, to implement the application logic of the service. Both of these scripts are given in Appendix B.

**Invoking the service - built-in client**

BioMoby contains a client library (MOBY-Client) including methods for communicating with MOBY services. For convenience, MOSES-MOBY also provides a testing script which is executed as follows:

```
moses-testing-service.pl Service::simpleCalculatorAdd input.xml
```

‘input.xml’ contains MOBY XML data:

```
<moby:MOBY xmlns:moby="http://www.biomoby.org/moby">
```
The output is as follows, showing the service has been implemented correctly and is returning the expected data:

```xml
<MOMBY xmlns:moby="http://www.biomoby.org/moby">
  <moby:mobyContent moby:authority="mycib.ac.uk">
    <moby:mobyData moby:queryID="job_0">
      <moby:Simple moby:articleName="sum">
      </moby:Simple>
    </moby:mobyData>
  </moby:mobyContent>
</MOMBY>
```

**Invoking the service - Taverna client**

Support for BioMoby is provided through the BioMoby scavenger, which enables a user to specify the location of a BioMoby central registry. Services and datatypes are then available as workflow processors in the Available Services window, and are organised according to their registration authorities. Figure 4.6 shows the simpleCalculatorAdd service discovered and used in a Taverna workflow.

### 4.4 Conclusion

Table 4.1 summarises the criteria and test cases applied to each deployment technology.

SOAP::Lite is a powerful module, implementing tasks as web service operations through the creation of subroutines which are exposed using a dispatcher. The lack of
<table>
<thead>
<tr>
<th></th>
<th>SOAP::Lite</th>
<th>Apache Axis</th>
<th>Soaplab1</th>
<th>Soaplab2</th>
<th>BioMoby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>Freely available</td>
<td>Freely available</td>
<td>Freely available</td>
<td>Freely available</td>
<td>Freely available</td>
</tr>
<tr>
<td>Installation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prerequisites</td>
<td>Perl, web server to run CGI scripts</td>
<td>Java, servlet container</td>
<td>Perl, Java, Apache Axis and a servlet container</td>
<td>Perl, Java, Apache Ant and a servlet container</td>
<td>(Perl version) Perl, web server to run CGI scripts, BioMoby modules from CPAN</td>
</tr>
<tr>
<td>Procedure</td>
<td>Module dependencies handled automatically by CPAN module, otherwise require manual installation</td>
<td>Copy axis directory to Tomcat, add JARs to AXISCLASSPATH environment variable</td>
<td>Interactive installation script</td>
<td>Ant handles build and installation tasks</td>
<td>Some BioMoby dependencies handled automatically by CPAN module, also require package manager</td>
</tr>
<tr>
<td>Support</td>
<td>Incomplete documentation, responsive mailing list</td>
<td>Comprehensive documentation, responsive mailing list</td>
<td>Fairly comprehensive documentation, responsive mailing list</td>
<td>Comprehensive documentation, responsive mailing list</td>
<td>Incomplete documentation, responsive mailing list</td>
</tr>
<tr>
<td>Web service protocols supported</td>
<td>SOAP, WSDL (limited)</td>
<td>SOAP, WSDL</td>
<td>SOAP, WSDL</td>
<td>SOAP, WSDL</td>
<td>SOAP, WSDL</td>
</tr>
<tr>
<td>Implementation of tasks in a web service</td>
<td>Task = Perl subroutine exposed as a web service operation</td>
<td>Task = Java method exposed as a web service operation</td>
<td>Task = executable wrapped as a web service</td>
<td>Task = executable wrapped as a web service</td>
<td>Task = Perl subroutine exposed as a web service</td>
</tr>
<tr>
<td>Invoking the service - built-in client</td>
<td>Client library available</td>
<td>Client library available</td>
<td>No client library, but command-line client included</td>
<td>No client library, but command-line client included</td>
<td>Client library available</td>
</tr>
<tr>
<td>Invoking the service - Taverna client</td>
<td>Extra work required: generation of WSDL using Pod::WSDL module, WSDL scavenger built into Taverna</td>
<td>No further work required: WSDL automatically generated, WSDL scavenger built into Taverna</td>
<td>No further work required: Soaplab1 scavenger built into Taverna</td>
<td>No further work required: Soaplab2 scavenger built into Taverna</td>
<td>No further work required: BioMoby scavenger built into Taverna</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of web service deployment technology evaluation
Figure 4.6: BioMoby services are used in Taverna workflows slightly differently to the examples already shown. The inputs to the workflow, in this case labelled ‘int1’ and ‘int2’ are used to provide a value for each ‘String’ object. The ‘Parse_Moby_Data_String’ processor is used to parse the output of the ‘simpleCalculatorAdd’ service, to retrieve the desired part of the output and direct it to the ‘sum’ output of the workflow.

direct support from within Taverna renders this an impractical solution for this work, however, as there is an extra step required to generate WSDL, and services can only be developed in Perl.

Apache Axis services provide a solution to this issue through automatic WSDL generation, however services can only be written in Java, limiting its usefulness when wrapping methods from existing network analysis libraries, a limitation which can also be applied to SOAP::Lite.

Soaplab and BioMoby are composite frameworks, as they build on pre-existing technologies. Soaplab1 uses Apache Axis and Tomcat, while Soaplab2 introduces JAX-WS. The Perl version of BioMoby uses SOAP::Lite to handle the sending and receiving of SOAP messages. Both of these therefore offer a richer functionality. BioMoby is a relevant package as it focuses on biological and bioinformatics services, and very successfully implements an object-driven registry query system through the use of the
object and service ontologies. This enables the traversal of a diverse set of data and tools, where each possible step is based on the data which the user is currently investigating. The same drawback exists however as for Apache Axis and SOAP::Lite in that services may only be implemented in a single language, either Java or Perl depending on the version chosen.

Soaplab is therefore the preferred choice for web service development. The main advantages are the flexibility it offers regarding the language the service is written in, and the availability of a Soaplab scavenger in Taverna. While the initial set of network construction and analysis services were deployed using Soaplab1, the release of Soaplab2 offers further advantages, such as the introduction of the Spinet web-based client, and the use of Apache Ant to build, install, generate metadata and deploy services. The introduction of Ant results in a much smoother development process. However the creation of ACD files is still relatively laborious and time-intensive. Each service requires an individually written ACD descriptor, and the process is not currently automated.

One particular advantage offered by Soaplab over the other technologies stems from the use of ACD to describe the command-line of an executable which is to be wrapped as a service: the datatypes “infile” and “outfile”. Inclusion of these datatypes to describe input and output data for a web service results in two differently named arguments. For example, the input ‘int1’ generates the arguments ‘int1_direct data’ and ‘int1_url’. The former is equivalent to “pass by value” and the latter to “pass by reference”. Supporting pass by reference in web services is an important requirement, as for large data, processing time and overheads are significantly reduced, and ‘out of memory’ errors may be avoided. This is particularly relevant with regard to the framework, which specifies that holistic network data can be submitted to web services for processing. Sending such data via SOAP may incur significant performance issues.

An advantage when using BioMoby is that the ontologies ensure that all the services that can consume a particular piece of data are presented to the user. For example, if they wish to query an NCBI gene identifier, then only the services which consume this object type are accessible. This semantic discovery of resources facilitates the ‘wandering through large data sets in a manner similar to the thought processes of biologists’ (Wilkinson and Links, 2002). While Soaplab cannot match this semantic functionality, the development of the framework introduces categories of services aimed at helping the user construct meaningful queries. A further advantage to Soaplab is that scripts may be developed as ‘normal’ and are only wrapped as web services at the final stage. This not only benefits a bioinformatics service provider, but enables other
members of a research group to contribute to the total set of web services developed, simply by giving an executable to the service provider, who can generate metadata and deploy it as a service. Development of BioMoby services is a more intensive process: objects and the services that consume them should be designed according to a rigorous API. To overcome this issue, MOSES-MOBY offers a way to generate skeleton code, thereby automating parts of the process.

Apart from BioMoby, none of the technologies surveyed implement a registry for service discovery. As previously stated, bioinformatics services tend to be made available in an informal fashion, rather than adhering to a formal protocol such as UDDI. Two recent initiatives however seek to address the problem of how to make such web services discoverable by the research community. The EMBRACE Registry (Pettifer et al., 2009) is motivated by the fact that while services are becoming common, the mechanisms for publishing them are less mature. It enables users to rank and annotate services, which are monitored automatically according to test scripts supplied by service providers. This allows other users to select the most appropriate service for their particular task. BioCatalogue (Goble et al., 2009), currently in beta, seeks to solve similar issues and also enables users to search, register, and annotate biological web services. A key advantage however is the provision of APIs which can be used by Taverna to programmatically access the registry of services.
Chapter 5

Web Services Developed

5.1 Introduction

The aim of this chapter is to present comprehensive documentation of the web services developed by the author. The services are listed in alphabetical order within their assigned Soaplab2 groups. This is the order in which they appear when scavenged and displayed in the Available Processors pane within the Taverna workbench. The documentation is divided into three parts:

- The first part of the documentation is a summary table of all the web services developed, with the following headings:

  **Soaplab2 group** The Soaplab2 group names are assigned during development.
  
  **Service name**
  
  **Framework category** A web service belongs to one of the four framework categories described in Chapter 3: data retrieval, data transformation, data analysis or output rendering. The category is important when considering the order in which services are connected when creating workflows.
  
  **Implementation details** All the web services documented in this chapter are developed using Soaplab2 as described in Chapter 4. Two components are required for each web service: the underlying executable program which is the application logic of the web service, and the custom ACD file used to describe the command line of the executable and generate the interface description. The implementation of a web service may therefore be described using one of the following:
5.2 Web service design

The reasoning behind the design of analysis web services (in the groups analyse\_directed and analyse\_undirected) was based on the literature on biological network construction and analysis, and the operations carried out on holistic datasets that have resulted in biologically meaningful observations. As such, graph-theoretic libraries in Perl and Python were identified as providing a large number of algorithms between them, which are applicable to the field, and therefore appropriate for inclusion in the toolkit of web services. Data transformation web services were each designed specifically to process a

\[\text{http://search.cpan.org/~jhi/Graph/}\]

\[\text{http://search.cpan.org/dist/XML-DOM/}\]
particular network representation standard. Data retrieval web services were designed to access local copies of publically available interaction datasets, and therefore are an example of how a relevant dataset may be leveraged by an expert bioinformatician. Output rendering web services were designed based on the type and format of data produced by analysis operations.
### 5.3 Table of web services

<table>
<thead>
<tr>
<th>Soaplab2 group</th>
<th>Service name</th>
<th>Framework category</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyse_directed</td>
<td>add_edges_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_network_diameter_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_network_radius_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_sink_nodes_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_source_nodes_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_subgraph_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>largest_strongly_connected_component_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>list_strongly_connected_components_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>node_degree_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>node_in_degree_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>node_out_degree_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>query_adjacency_matrix_directed</td>
<td>Data analysis</td>
<td>(B)**</td>
</tr>
<tr>
<td></td>
<td>rank_betweenness_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_closeness_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_degrees_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_secondary_degrees_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>remove_edges_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>shortest_path_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>shortest_path_length_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>size_distribution_strongly_connected_components_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>total_edges_directed</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td>analyse_misc</td>
<td>compare_two_rankings</td>
<td>Data analysis</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>reverse_adjacency_list</td>
<td>Data analysis</td>
<td>(A)</td>
</tr>
<tr>
<td>analyse_undirected</td>
<td>add_edges_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>cliques_containing_node_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>find_cliques_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_average_clustering_coefficient_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_bridges_undirected</td>
<td>Data analysis</td>
<td>(B)**</td>
</tr>
<tr>
<td></td>
<td>get_clique_by_size_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_cut_nodes_undirected</td>
<td>Data analysis</td>
<td>(B)**</td>
</tr>
<tr>
<td></td>
<td>get_cyclic_core</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_network_diameter_undirected</td>
<td>Data analysis</td>
<td>(B)**</td>
</tr>
<tr>
<td></td>
<td>get_network_radius_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>get_subgraph_radius_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
</tbody>
</table>

71
5.3 Table of web services

<table>
<thead>
<tr>
<th>Soaplab2 group</th>
<th>Service name (cont.)</th>
<th>Framework category (cont.)</th>
<th>Implementation (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>largest_connected_component_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>list_connected_components_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>node_clustering_coefficient_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>node_degree_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>query_adjacency_matrix_undirected</td>
<td>Data analysis</td>
<td>(B)**</td>
</tr>
<tr>
<td></td>
<td>rank_betweenness_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_closeness_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_clustering_coefficients_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_degrees_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>rank_secondary_degrees_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>remove_edges_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>remove_nodes</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>remove_self_loops_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>remove_singleton_nodes</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>size_distribution_connected_components_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>size_of_largest_clique_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>total_edges_undirected</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td></td>
<td>total_nodes</td>
<td>Data analysis</td>
<td>(B)*</td>
</tr>
<tr>
<td>render_output</td>
<td>common_graph_to_dot_directed</td>
<td>Output transformation</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>common_graph_to_dot_undirected</td>
<td>Output transformation</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>dot</td>
<td>Output rendering</td>
<td>(C)</td>
</tr>
<tr>
<td></td>
<td>format_psi25_id_list</td>
<td>Output transformation</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>format_sbml_id_list</td>
<td>Output transformation</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>neato</td>
<td>Output rendering</td>
<td>(C)</td>
</tr>
<tr>
<td>retrieve</td>
<td>query_atpid</td>
<td>Data retrieval</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>query_inferred</td>
<td>Data retrieval</td>
<td>(A)</td>
</tr>
<tr>
<td>transform</td>
<td>common_graph_to_sif</td>
<td>Data transformation</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>psi25_to_common_graph</td>
<td>Data transformation</td>
<td>(B)***</td>
</tr>
</tbody>
</table>
Table 5.1: List of the network construction and analysis web services deployed

### 5.4 Description of web services

#### 5.4.1 Group: analyse_directed

##### 5.4.1.1 add_edges_directed

*Intent* The intention of this web service is to add a specified set of edges to a directed network, and return the new network.

*Motivation* Edges in biological networks represent interactions or reactions between biological entities. Such relationships are established via experimental methods, or may be computationally inferred. This web service enables the addition of novel edges by the user, following manual inspection and curation, thus improving the overall quality of the network model. The addition of edges which are known to be biologically accurate ensures that any further analyses are more likely to return significant result.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network, and a list of edges also represented using the common graph format. N.B. as directionality is encoded, the order of a pair of nodes which constitutes an edge is important. Therefore if the network contains a directed edge $E_1$ connecting $A$ to node $B$ but the specified edge for addition is $E_2$ connecting node $B$ to node $A$, then edge $E_2$ is added as a new edge.

##### 5.4.1.2 get_network_diameter_directed

*Intent* The intention of this web service is to calculate the diameter of a directed network. The diameter is defined as the maximum eccentricity of any node in the network, where the eccentricity of each node $n$ is the greatest distance (path length, or number of edges) between $n$ and any other node. The diameter is therefore the distance between
the two nodes which are furthest away from each other in the network.

**Motivation** Network diameter may be used as a measure of robustness, that is, how resilient a network is when faced with nodes or edges are removed. In biological networks this could refer to the mutation or removal of genes due to disease. As such networks tend to be scale-free (as discussed in Chapter 2), deletion of random nodes are tolerated relatively well (i.e. diameter remains characteristically small), whereas targeted removal of hub nodes results in a greatly increased network diameter.

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network. The web service can only return the diameter if the directed network is **strongly connected**, that is, if there is path between every pair of nodes in the network. This web service may be applied to a network which is not strongly connected, however an error will be reported.

### 5.4.1.3 get network_radius_directed

**Intent** The intention of this web service is to calculate the radius of a directed network. The radius is defined as the minimum eccentricity of any node in the network, where the eccentricity of each node $n$ is the greatest distance (path length, or number of edges) between $n$ and any other node.

**Motivation** The network radius may be used as a measure of robustness in the same way as the network diameter (see: get network_diameter_directed)

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network. The web service can only return the radius if the directed network is strongly connected. This web service may be applied to a network which is not strongly connected, however an error will be reported.

### 5.4.1.4 get sink_nodes_directed

**Intent** The intention of this web service is to calculate the list of sink nodes in a directed network. A sink node is one which only has predecessors and no successors.

**Motivation** Obtaining a list of sink nodes for a directed biological network such as a metabolic network indicates which molecules are the final product of a sequence of reactions. This information can be used to check the completeness of a holistic model; molecules may be present in the list which should themselves be part of further reactions, but this information is missing from the network model.
**5.4 Description of web services**

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network. (N.B. if the network contains singleton nodes, then these are returned as sink nodes. If the user wishes to ignore these, they should apply the web service `remove_singleton_nodes` first).

### 5.4.1.5 get_source_nodes_directed

*Intent* The intention of this web service is to calculate the list of source nodes in a directed network. A source node is one which only has successors and no predecessors.

*Motivation* Obtaining a list of source nodes for a directed biological network such as a metabolic indicates which molecules are the initial substrates of a sequence of reactions. This information can be used to check the completeness of a holistic model; molecules may be present in the list which should themselves be part of further reactions, but this information is missing from the network model.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network. (N.B. if the network contains singleton nodes, then these are returned as source nodes. If the user wishes to ignore these, they should apply the web service `remove_singleton_nodes` first).

### 5.4.1.6 get_subgraph_directed

*Intent* The intention of this web service is to extract a subgraph from a directed network given a list of nodes in that network, where the subgraph comprises the nodes in the list and any edges connecting them, in the common graph format.

*Motivation* A subgraph of a holistic biological network may represent a clique, motif or component and so it is useful to isolate this subgraph in order to carry out further analyses, which could help to determine the biological function a particular element of the subgraph, or the function of the subgraph as a whole. Isolation of the subgraph in the common graph format also enables the user to visualise the subgraph using services from the `render_output` group of web services.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network, and a list of nodes which are part of the network.
5.4 Description of web services

5.4.1.7 largest_strongly_connected_component_directed

**Intent** The intention of this web service is to return a list of the nodes in the largest strongly connected component of a directed network. A strongly connected component is a subgraph which there exists a path between every pair of nodes.

**Motivation** This web service is motivated by work carried out on strongly connected components in metabolic networks by Ma and Zeng (2003a), and further expanded (Csete and Doyle, 2004; Kitano, 2004; Palumbo et al., 2005). Ma and Zeng established the biological significance of strongly connected components by showing that, for the metabolic networks of 65 fully-sequenced organisms, there exists a ‘bow-tie’ structure. This consists of one giant strong component (GSC), a product subset of metabolites, a substrate subset of metabolites and an isolated subset. This GSC is the largest of the strongly connected components, which Ma and Zeng established as being the core of a metabolic network, and the most complex part. Isolation of this part of the network is therefore useful for further study.

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.8 list_strongly_connected_components_directed

**Intent** The intention of this web service is to calculate a list of the nodes in each strongly connected component of a directed network.

**Motivation** The significance of strongly connected components in directed biological networks is described above. A list of the strongly connected components of a network may be used to establish which part of the network metabolites and reactions appear in.

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.9 node_degree_directed

**Intent** The intention of this web service is to calculate the degree of a node of interest in a directed network. The degree is the number of edges incident to the node, both directed towards the node and away from it.

**Motivation** As discussed in Chapter 2, the degree of a node in a biological network can indicate its global importance, as those nodes with a higher degree, commonly termed network ‘hubs’ are involved in more interactions and/or reactions, and so their removal may disturb the network architecture more significantly than those with a lower degree.
5.4 Description of web services

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.10 node_in_degree_directed

Intent The intention of this web service is to calculate the in-degree of a node of interest in a directed network. The in-degree is the number of incoming edges to the node.

Motivation Metabolic networks are frequently represented as directed networks, so the in-degree is a useful calculation for a given reaction identifier, to ascertain how many metabolites are substrates of the reaction. The in-degree of a metabolite denotes how many reactions produce it.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.11 node_out_degree_directed

Intent The intention of this web service is to calculate the out-degree of a node of interest in a directed network. The in-degree is the number of outgoing edges from the node.

Motivation Metabolic networks are frequently represented as directed networks, so the out-degree is a useful calculation for a given reaction identifier, to ascertain how many metabolites are products of the reaction. The out-degree of a metabolite denotes how many reactions consume it.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.12 query_adjacency_matrix_directed

Intent The intention of this web service is to create the adjacency matrix of a directed network, and query it with a node of interest. The rows and columns of an adjacency matrix are labelled by nodes. Position \((n_1, n_2)\) is a 1 if \(n_1\) and \(n_2\) are adjacent, or 0 if they are not. If there is a directed edge going from \(n_1\) to \(n_2\), then \(n_1\) is adjacent to \(n_2\), but \(n_2\) is not adjacent to \(n_1\) (i.e. the matrix is not symmetric).

Motivation The construction of an adjacency matrix to represent a network is useful as it enables the discovery of which nodes are direct neighbours of a node of interest. While calculation of the in- and out-degree of a metabolite in a directed metabolic network shows how many enzymes and/or reactions it is a substrate or product of, the adjacency matrix shows which elements it is connected to.
5.4 Description of web services

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.13 rank_betweenness_directed

Intent The intention of this web service is to generate a list of all the nodes in the network, ranked according to their betweenness centrality values, from highest to lowest. For a given node, the betweenness centrality is the proportion of shortest paths between other nodes that it occurs on.

Motivation As discussed in Chapter 2, betweenness centrality is a useful measure of the global importance of a node in a biological network. The ranking of all the nodes in the network can therefore be used to select potentially significant molecules in the network for further study.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.14 rank_closeness_directed

Intent The intention of this web service is to generate a list of all the nodes in the network, ranked according to their closeness centrality values, from highest to lowest. The closeness centrality of a given node is the mean shortest path length between the node and all other nodes reachable from it.

Motivation Closeness centrality is an indication of how quickly information can be transferred from a given node to all others in a network, and so in biological networks a ranking of closeness centralities can show which nodes are important with regard to transferring the effects of perturbations.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.15 rank_degrees_directed

Intent The intention of this web service is to generate a list of all the nodes in the network, ranked according to their degree values (in-degree plus out-degree), from highest to lowest.

Motivation As discussed in Chapter 2, the degree of a node in a biological network can indicate its global importance, as those nodes with a higher degree (‘hubs’) are involved in more interactions and/or reactions, and so their removal may disturb the network architecture more significantly than those with low degree. A ranking of all the nodes
in the network according to their degree can be used to select potentially significant molecules for further study.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.16 rank secondary degrees directed

Intent The intention of this web service is to generate a list of all the nodes in the network, ranked according to their secondary degree values from highest to lowest. The secondary degree of a node is defined as the number of nodes reachable from it which are two edges away.

Motivation If a node in a network has a high degree, then it may be possible to infer that the node is globally important. However a node may have a low degree, but have a high secondary degree, i.e. it is connected to a small number of immediate neighbours, but they in turn are connected a large number of immediate neighbours. There may therefore be a high impact on the network if such a node is removed. In a directed network such as one representing metabolism, this could be an enzyme which catalyses a reaction where there is exactly one substrate and one product, but these are the products and substrates respectively of a large number of reactions.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.17 remove edges directed

Intent The intention of this web service is to remove a specified set of edges from a directed network, and return the new network.

Motivation Edges in biological networks represent interactions or reactions between biological entities. Such relationships are established via experimental methods, or may be computationally inferred. This web service enables the removal of those edges which a user deems to be inaccurate, thus improving the overall quality of the network model. The removal of edges which are known to be biologically accurate simulates the effect of a mutation or deletion, and the overall effect on the network’s architecture and integrity.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network, and a list of edges also represented using the common graph format. NB as directionality is encoded, the order of a pair of nodes which constitutes an edge is important. Therefore
if the network contains a directed edge $E_1$ connecting $A$ to node $B$ but the specified edge for removal is $E_2$ connecting node $B$ to node $A$ then edge $E_1$ will not be removed.

5.4.1.18 shortest path directed

*Intent* The intention of this web service is to calculate the nodes which appear on the shortest path between two nodes of interest.

*Motivation* There are a number of established pathways in biological networks, for example the metabolic pathways which when assembled for a particular organism result a metabolic network. Such pathways convert a particular substrate into a final product, and both ends of the pathway may be produced and consumed, or be intermediaries in, other pathways. A shortest path analysis can therefore be used to establish if there is redundancy built into such pathways, by identifying an alternative route between two metabolites of interest. In directed networks, it is also useful to determine if the shortest path from $n_1$ to $n_2$ also exists in the other direction, between $n_2$ and $n_1$.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.19 shortest path length directed

*Intent* The intention of this web service is to calculate the length of the shortest path (i.e. number of edges) between two nodes of interest.

*Motivation* In a metabolic network, the length of the shortest path between two metabolites denotes how many reactions separate them.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.20 size distribution strongly connected components directed

*Intent* The intention of this web services is to calculate the distribution of sizes (number of nodes) of strongly connected components in a directed network.

*Motivation* As discussed, strongly connected components in directed biological networks such as those representing metabolism have been shown to conform to a ‘bow-tie’ structure, that is, there is one giant strong component, containing the core of the network, and many smaller strong components. This web service is therefore useful to establish whether or not a GSC may exist, before further analyses relating to strongly connected components are carried out.
5.4 Description of web services

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.1.21 total_edges_directed

Intent The intention of this web service is to calculate the total number of edges in a directed network.

Motivation The total number of edges in a network indicates the number of interactions and/or reactions in a biological network, and so is a useful statistical measure especially when taken in conjunction with the total number of nodes in the network, as a comparison of the two reveals if the network is sparsely or densely connected.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network. This web service will count all the edges in the network, i.e. in a network where there is a directed edge connecting node A to node B and another directed edge connecting B to A then the total edges is 2. The user may apply the web service total_edges_undirected if they wish to count such a relationship as one edge.

5.4.2 Group: analyse_misc

5.4.2.1 compare_two_rankings

Intent The intention of this web service is to compare two lists of nodes, where each is a ranked according to some topological metric. The number of nodes to compare from each list is specified by the user, and the web service calculates those nodes which appear on only one of the lists, and those which appear in both.

Motivation This web service is motivated by research carried out by Joy et al. (2005), Gandhi et al. (2006) and Bader and Madduri (2007), discussed in Chapter 2. These studies established that globally important molecules may not be characterised by their high degree alone, and so a direct comparison between rankings of, for example, degree and betweenness, may help to reveal which molecules play a significant role in the network.

Applicability This web service may be applied to two generated rankings of nodes for the same network.

5.4.2.2 reverse_adjacency_list

Intent An adjacency list is a (usually unordered) list of nodes in a network, where each node is itself followed by a list of the nodes adjacent to it. The intention of this web
service is to process an adjacency list representation of a set of interactions (commonly PPIs), and ‘reverse’ it, so that the nodes in each adjacency list themselves are listed, and followed by a list of adjacent nodes.

Motivation Having established the interacting partners of a set of proteins of interest, the reversal of the adjacency list representation reveals which of the interacting partners interact with two or more of the proteins. For example, a set of enzymes along a metabolic pathway may interact with a number of other proteins, including other enzymes. The reversal of these interactions therefore shows which of these other proteins interact with multiple enzymes, having a potential regulatory effect on the pathway.

Applicability This web service may be applied to an adjacency list representation of a set of interactions, where each line is a node, followed by the interacting partners of that node, separated by any whitespace character.

5.4.3 Group: analyse_undirected

5.4.3.1 add_edges_undirected

Intent The intention of this web service is to add a specified set of edges to an undirected network, and return the new network.

Motivation Edges in biological networks represent interactions or reactions between biological entities. Such relationships are established via experimental methods, or may be computationally inferred. This web service enables the addition of novel edges by the user, following manual inspection and curation, thus improving the overall quality of the network model. The addition of edges which are known to be biologically accurate ensures that any further analyses are more likely to return significant result.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network, and a list of edges also represented using the common graph format. NB as directionality is not encoded, the order of a pair of nodes which constitutes an edge is unimportant. Therefore if the network contains an edge $E_1$ connecting $A$ to node $B$ and the specified edge for addition is $E_2$ connecting node $B$ to node $A$, then $E_2 \equiv E_1$ and $E_2$ is not added as a new edge.

5.4.3.2 cliques_containing_node_undirected

Intent The intention of this web service is to generate a list of cliques containing a node of interest. A clique is a subset of nodes in a network such that there is an edge connecting all pairs of nodes in the subset.
Motivation In biological networks, a clique may be considered analogous to a (possibly functional) complex, for example in a PPI network. The cliques that a particular protein belongs to may be studied further to determine if all members of the clique bind at the same time, or under the same environmental conditions.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.3 **find_cliques_undirected**

*Intent* The intention of this web service is to generate a list of all the cliques in a network.

*Motivation* As discussed, cliques may be analogous to (possibly functional) complexes. For a holistic network such one containing all known PPIs in an organism, this is a lengthy calculation, but provides a list which may be examined to reveal potentially novel complexes. This is also useful if there is no particular protein of interest under study, and the user wishes to query the whole network.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.4 **get_average_clustering_coefficient_undirected**

*Intent* The intention of this web service is to calculate the value of the average clustering coefficient. The clustering coefficient of a single node quantifies how close the node’s neighbours are to being a clique, and the average is calculated all the nodes in the network.

*Motivation* The measure was introduced by Watts and Strogatz (1998) to determine if a network exhibited the small-world property. Small-world networks exhibit a clustering coefficient significantly higher than expected by random chance.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.5 **get_bridges_undirected**

*Intent* The intention of this web service is to return a list of bridges for an undirected network.

*Motivation* A bridge is an edge in an undirected network, whose removal increases the number of connected components. In biological networks, an edge is analogous to a reaction or interaction between two entities, and so identification of bridges reveals which
are the interactions whose repression or removal altogether would fracture the network into components and result in possibly limiting communication between different parts of the network, or have a fatal effect, equivalent to cell death.  

_Applicability_ This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

### 5.4.3.6 get_clique_by_size_undirected

**Intent** The intention of this web service is to generate a list of cliques of a given size (i.e. containing a particular number of nodes) for a network.  

**Motivation** In a large PPI network, there may be a great many smaller cliques containing one, two or three proteins. Such cliques may not have any functional relevance, so by specifying a clique size the user may investigate larger, more significant cliques systematically.  

_Applicability_ The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

### 5.4.3.7 get_cut_nodes_undirected

**Intent** The intention of this web service is to return a list of cut nodes (also known as articulation points) for an undirected network.  

**Motivation** A cut node is a node in an undirected network, whose removal increases the number of connected components. In biological networks, cut nodes are elements of the network whose repression (e.g. of gene expression) or removal altogether (e.g. gene deletion) would fracture the network into components, and result in possibly limiting communication between different parts of the network, or have a fatal effect, equivalent to cell death.  

_Applicability_ This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

### 5.4.3.8 get_cyclic_core

**Intent** The intention of this web service is to calculate for either a directed or undirected network, the list of nodes which make up the cyclic core. The cyclic core is defined as a subgraph of the network where every node is part of a cycle, where a cycle is a path where the source and destination nodes are the same.  

**Motivation** Identification of cycles, particularly those involving regulatory steps, is crucial for acquiring a physiological perspective on network behaviour. This analysis
returns all the cycles in a network, however these must be studied further to establish novel feedback behaviour.

Applicability This web service may be applied to a network represented in the common graph format, which the user wishes to interpret as a directed or undirected network.

5.4.3.9 get_network_diameter_undirected

Intent The intention of this web service is to calculate the diameter of a network. The diameter is defined as the maximum eccentricity of any node in the network, where the eccentricity of each node \( n \) is the greatest distance (path length, or number of edges) between \( n \) and any other node. The diameter is therefore the distance between the two nodes which are furthest away from each other in the network.

Motivation Network diameter may be used as a measure of robustness, that is, how resilient a network is when faced with nodes or edges are removed. In biological networks this could refer to the mutation or removal of genes due to disease. As such networks tend to be scale-free (as discussed in Chapter 2), deletion of random nodes are tolerated relatively well (i.e. diameter remains characteristically small), whereas targeted removal of hub nodes results in a greatly increased network diameter.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.10 get_network_radius_undirected

Intent The intention of this web service is to calculate the radius of network. The radius is defined as the minimum eccentricity of any node in the network, where the eccentricity of each node \( n \) is the greatest distance (path length, or number of edges) between \( n \) and any other node.

Motivation The network radius may be used as a measure of robustness in the same way as the network diameter (see: get_network_diameter_undirected)

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.11 get_subgraph_undirected

Intent The intention of this web service is to extract a subgraph from an undirected network given a list of nodes in that network, where the subgraph comprises the nodes in the list and any edges connecting them.

Motivation A subgraph of a holistic biological network may represent a clique, motif
or component and so it is useful to isolate this subgraph in order to carry out further analyses, which could help to determine the biological function a particular element of the subgraph, or the function of the subgraph as a whole. Isolation of the subgraph in the common graph format also enables the user to visualise the subgraph using services from the render_output group of web services.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network, and a list of nodes which are part of the network.

5.4.3.12 largest_connected_component_undirected

Intent The intention of this web service is to calculate the largest connected component of an undirected network. A connected component is a subgraph in which there exists a path between all pairs of nodes in the network.

Motivation In undirected biological networks such as PPI networks, it is commonly the case that there exist a number of connected components, some of which may be very small. These are isolated from each other, i.e. there are no interactions (edges) connecting the different components, which may be due to a number of reasons, such as incomplete knowledge, or experimental or human errors. Calculations such as path lengths, diameter and radius cannot be carried out on a network that consists of connected components, so the largest of these, when isolated, is useful for further study.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.13 list_connected_components_undirected

Intent The intention of this web service is to calculate a list of the nodes in each connected component of an undirected network.

Motivation While the largest component of a network is useful to carry out certain graph-theoretic analyses, a complete list of connected components enables the user to pick out the parts which do not connect to the larger subgraphs, and potentially suggest ways to improve the accuracy of the network by investigating why the smaller components exist, and modifying the network model if real biological events are deemed to be missing.

Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.
5.4 Description of web services

5.4.3.14 node_clustering_coefficient_undirected

*Intent* The intention of this web service is to calculate the clustering coefficient of a particular node of interest, in an undirected network.

*Motivation* The clustering coefficient of a single node may be used to determine how close the neighbourhood of the node is to forming a clique. In a PPI network, if a protein has a high clustering coefficient then it is more likely to be part of a complex of interacting proteins.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.15 node_degree_undirected

*Intent* The intention of this web service is to calculate the degree of a node of interest in an undirected network. The degree is the number of edges incident to the node.

*Motivation* As discussed in Chapter 2, the degree of a node in a biological network can indicate its global importance, as those nodes with a higher degree are involved in more interactions and/or reactions, and so their removal may disturb the network architecture more significantly than those with a lower degree.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.16 query_adjacency_matrix_undirected

*Intent* The intention of this web service is to create the adjacency matrix of a directed network, and query it with a node of interest. The rows and columns of an adjacency matrix are labelled by nodes. Position \((n_1,n_2)\) is a 1 if \(n_1\) and \(n_2\) are adjacent, or 0 if they are not. If there is an undirected edge going from \(n_1\) to \(n_2\), then \(n_1\) is adjacent to \(n_2\), and \(n_2\) is also adjacent to \(n_1\) (i.e. the matrix is symmetric).

*Motivation* The construction of an adjacency matrix representation of a network is useful as it enables the discovery of which nodes are direct neighbours of a node of interest. In a PPI network it is often the case that proteins share a similar function to their interaction partners.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.
5.4 Description of web services

5.4.3.17 rank_betweenness_undirected

*Intent* The intention of this web service is to generate a list of all the nodes in the network, ranked according to their betweenness centrality values, from highest to lowest. For a given node, the betweenness centrality is the proportion of shortest paths between other nodes that it occurs on.

*Motivation* As discussed in Chapter 2, betweenness centrality is a useful measure of the global importance of a node in a biological network. The ranking of all the nodes in the network can therefore be used to select potentially significant molecules in the network for further study.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.18 rank_closeness_undirected

*Intent* The intention of this web service is to generate a list of all the nodes in the network, ranked according to their closeness centrality values, from highest to lowest. The closeness centrality of a given node is the mean shortest path length between the node and all other nodes reachable from it.

*Motivation* Closeness centrality is an indication of how quickly information can be transferred from a given node to all others in a network, and so in biological networks a ranking of closeness centralities can show which nodes are important with regard to transferring the effects of perturbations.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.19 rank_clustering_coefficients_undirected

*Intent* The intention of this web service is to generate a list of all the nodes in the network, ranked according to their clustering coefficients, from highest to lowest.

*Motivation* By ranking all the clustering coefficients in, for example, a PPI network, it is possible to establish which proteins show the greatest tendency to be part of cliques. A protein with a high clustering coefficient is therefore more likely to be part of a (possibly functional) complex.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.
5.4.3.20 rank_degrees Undirected

*Intent* The intention of this web service is to generate a list of all the nodes in the network, ranked according to their degree values, from highest to lowest.

*Motivation* As discussed in Chapter 2, the degree of a node in a biological network can indicate its global importance, as those nodes with a higher degree ('hubs') are involved in more interactions and/or reactions, and so their removal may disturb the network architecture more significantly than those with low degree.

*Applicability* The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.21 rank_secondary_degrees Undirected

*Intent* The intention of this web service is to generate a list of all the nodes in the network, ranked according to their secondary degree values from highest to lowest.

*Motivation* If a node in a network has a high degree, then it may be possible to infer that the node is globally important. However a node may have a low degree, but have a high secondary degree, i.e. it is connected to a small number of immediate neighbours, but they in turn are connected to a large number of immediate neighbours. There may therefore be a high impact on the network if such a node is removed. In an undirected network such as a PPI network, such a protein may connect two hubs which in turn are highly connected to other proteins, so removal affects the communication between these two highly connected areas of the network.

*Applicability* This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.22 remove_edges Undirected

*Intent* The intention of this web service is to remove a specified set of edges from an undirected network, and return the new network.

*Motivation* Edges in biological networks represent interactions or reactions between biological entities. Such relationships are established via experimental methods, or may be computationally inferred. This web service enables the removal of those edges which a user deems to be inaccurate, thus improving the overall quality of the network model. The removal of edges which are known to be biologically accurate simulates the effect of a mutation or deletion, and the overall effect on the network’s architecture and integrity.

*Applicability* This web service may be applied to a network represented using the com-
mon graph format, which the user wishes to interpret as an undirected network, and a list of edges also represented using the common graph format. NB as directionality is not encoded, the order of a pair of nodes which constitutes an edge is unimportant. Therefore if the network contains an edge $E_1$ connecting $A$ to node $B$ and the specified edge for removal connects node $B$ to node $A$ then then $E_2 \equiv E_1$ and $E_2$ is removed from the network.

5.4.3.23 remove_nodes

**Intent** The intention of this web service is to remove a set of nodes from a directed or undirected network, and return the new network.

**Motivation** Nodes in biological networks can represent molecules such as genes, proteins or metabolites, or events such as reactions. Unlike edges (reactions and interactions) the presence of a node in a biological network is likely to be accurate, however providing the facility to remove nodes enables the user to manipulate a network under study to their liking. Removal of a node can represent a mutation or deletion and thus can enhance understanding of the role played by specific nodes in the network.

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed or undirected network.

5.4.3.24 remove_self_loops_undirected

**Intent** The intention of this web service is to remove the self loops (also known as self edges) from an undirected network, and return the network and a list of the nodes (if any) which are connected to themselves.

**Motivation** A self loop in an undirected network such as a PPI network implies a protein which interacts with itself, i.e. forms a dimer. While it is biologically accurate to represent such an interaction, the presence of self loops can skew the results of some graph-theoretic analyses. For example, a self edge increases the degree of a protein by one.

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.25 remove_singleton_nodes

**Intent** The intention of this web service is to remove singleton nodes from a network, and return the new network, as well as a list of singletons removed. A singleton node
is defined in both directed and undirected graphs as having a degree of zero, that is, there are no edges incident to the node.

**Motivation** Singleton nodes are uncommon in networks such as PPI and metabolic, as by their nature such networks define relationships between molecules, and a protein, gene or metabolite is rarely experimentally recorded as existing in isolation, rather they undergo interactions and reactions with other molecules. Nevertheless, singletons may arise as a result of certain analysis operations such as the removal of nodes and edges, or from transforming network data from one format to another, where a particular network entity is not labelled with the database identifier used throughout. Their presence may skew certain analyses, for example, singleton nodes appear in lists of both source and sink nodes, and the subgraph which consists of a single node is both a clique and a connected components.

**Applicability** The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed or undirected network.

### 5.4.3.26 size_distribution_connected_components_undirected

**Intent** The intention of this web services is to calculate the distribution of sizes (number of nodes) of connected components in an undirected network.

**Motivation** As discussed, undirected biological networks such as PPIs may contain subgraphs that do not connect to each other. The distribution of component size shows how many nodes belong to components of different sizes, and can prompt the user to investigate the biological reasons for the existence of such components.

**Applicability** This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

### 5.4.3.27 size_of_largest_clique_undirected

**Intent** The intention of this web service is to calculate the size (i.e. number of nodes) in the largest clique of an undirected network.

**Motivation** Cliques are subgraphs of the network in which all nodes are connected to all other nodes. In an undirected graph which represents a PPI network, the largest clique may be biologically significant. A recent study (Lin et al., 2009) hypothesised that since highly connected proteins are globally important, cliques may also be associated with essentiality. They found that the maximum (largest) clique in the yeast PPI network contained an extremely high number of essential proteins (90%).
Applicability This web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.28 total_edges_undirected

Intent The intention of this web service is to calculate the total number of edges in an undirected network.

Motivation The total number of edges in a network indicates the number of interactions and/or reactions in a biological network, and so is a useful statistical measure especially when taken in conjunction with the total number of nodes in the network, as a comparison of the two can reveal if the network is sparsely or densely connected.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.3.29 total_nodes

Intent The intention of this web service is to calculate the total number of nodes in a directed or undirected network.

Motivation The total number of nodes in a network indicates the number of molecules, for example proteins, genes or metabolites, in the network. For certain representations of metabolic networks the total also includes the number of reactions. The total number of nodes is a useful statistical measure when taken in conjunction with the number of edges, as a comparison of the two can reveal if the network is sparsely or densely connected.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed or undirected network.

5.4.4 Group: format_output

5.4.4.1 common_graph_to_dot_directed

Intent The intention of this service is to prepare a directed network for visualisation, using the DOT language.

Motivation The results of certain analyses produce subgraphs containing nodes of interest (e.g. shortest_path_directed). Such subgraphs may more helpfully viewed as network diagrams in order to better appreciate their connectivity, and to aid biological interpretations. The dot and neato web services developed are tools which generate
layout diagrams of networks, but can only process networks represented using the DOT language.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as a directed network.

5.4.4.2 common_graph_to_dot_undirected

Intent The intention of this service is to prepare an undirected network for visualisation, using the DOT language.

Motivation The results of certain analyses produce subgraphs containing nodes of interest (e.g. cliques containing node undirected). Such subgraphs may more helpfully viewed as network diagrams in order to better appreciate their connectivity, and to aid biological interpretations. The dot and neato web services developed are tools which generate layout diagrams of networks, but can only process networks represented using the DOT language.

Applicability The web service may be applied to a network represented using the common graph format, which the user wishes to interpret as an undirected network.

5.4.4.3 dot

Intent The intention of this web service is to process a directed or undirected network represented using the DOT language, and generate a hierarchical layout diagram of the network.

Motivation Visualisation of networks can be a useful aid to interpreting results. The DOT language is highly configurable with many options for node and edge (for example, different colours and styles) which can clarify and highlight certain parts of the network. The dot executable processes DOT files to generate layout diagrams.

Applicability This web service can be applied to a network represented using the DOT language. It is particularly useful when applied to biological pathways, as the hierarchical layout ensures edges are displayed in the same direction (e.g. from left to right, or top to bottom).

5.4.4.4 format_psi25_id_list

Intent The intention of this web service is to convert a list of PSI-MI 2.5 protein identifiers into a list of corresponding descriptive names.

Motivation The ‘id’ field of a PSI-MI protein is smaller than the ‘fullName’ field and so more useful computationally when working with graph data structures and algorithms.
However the ‘fullName’ field is more descriptive and so it is useful to format a list containing an ‘id’ for each protein with its name, to give more information to a user studying a list of results.

*Applicability* This web service may be applied to a list of protein identifiers from a PSI-MI 2.5 network, as long as the user has access to a proteins file, which is generated using the *psi25_to_common_graph* web service.

### 5.4.4.5 format sbml id list

*Intent* The intention of this web service is to convert a list of SBML identifiers into a list of corresponding SBML names, for ‘species’ in the network (i.e. molecules such as metabolites and proteins).

*Motivation* The ‘id’ field of an SBML species is smaller than the ‘name’ field and so more useful computationally when working with graph data structures and algorithms. However the ‘name’ field is more descriptive and so it is useful to format a list containing an ‘id’ for each molecule with its name, to give more information to a user studying a list of results.

*Applicability* This web service may be applied to a list of identifiers from the SBML ‘id’ field, as long as the user has access to a species file, which is generated using the *sbml_to_common_graph* web service.

### 5.4.4.6 neato

*Intent* The intention of this web service is to process a directed or undirected network represented using the DOT language, and generate a spring-embedded layout diagram of the network.

*Motivation* Visualisation of networks can be a useful aid to interpreting results. The DOT language is highly configurable with many options for node and edge (for example, different colours and styles) which can clarify and highlight certain parts of the network. The neato executable processes DOT files to generate layout diagrams.

*Applicability* This web service can be applied to a network represented using the DOT language. It is particularly useful when applied to subgraphs such as cliques or connected components, as the spring-embedded layout ensures nodes and edges are at an optimum distance from each other, with minimal crossing of edges.
5.4 Description of web services

5.4.5 Group: retrieve

5.4.5.1 query_atpid

*Intent* The intention of this web service is to query the *Arabidopsis thaliana* Protein Interaction Database (AtPID, Cui et al., 2008) with an AGI code of a gene of interest, to retrieve any interacting proteins.

*Motivation* AtPID contains around 24,500 PPIs, obtained by integrating several prediction methods for protein-protein interactions. Seven computational methods are used, including identifying orthologous interactions in other organisms, identifying proteins with shared biological function (i.e. their Gene Ontology annotations) as being more likely to interact, and co-expression matrices, which are used to identify interacting proteins on the basis of their similar gene expression patterns. The provision of a web service interface to this data enables users to query for interacting proteins, which may then be used in further study, for example topological metrics and protein locality within the network.

*Applicability* The web service may be applied to an *A. thaliana* AGI identifier.

5.4.5.2 query_inferred

*Intent* The intention of this web service is to query an inferred interactome for *A. thaliana*, developed by Geisler-Lee et al. (2007).

*Motivation* The inferred dataset contains almost 20,000 interactions between 3,617 *A. thaliana* proteins, predicted from interacting orthologues in *H. sapiens, C. melanogaster, C. elegans* and *S. cerevisiae*. The provision of a web service interface to this data enables users to query for interacting proteins, which may then be used in further study, for example topological metrics and protein locality within the network.

*Applicability* The web service may be applied to an *A. thaliana* AGI identifier.

5.4.6 Group: transform

5.4.6.1 common_graph_to_sif

*Intent* The intention of this web service is to convert a network represented using the common graph format into the Simple Interaction Format (SIF).

*Motivation* The common graph format is suitable for submission to the data analysis operations described in this chapter, however SIF is a format compatible with the software package Cytoscape. Conversion to SIF therefore enables the user to take advantage of functionality provided by Cytoscape that is not available as part of this
web services toolkit, such as visualisation of a whole network in order to manipulate individual node colours and other attributes.

*Applicability* The web service is applicable when the user wishes to analyse a network represented using the common graph format.

### 5.4.6.2 psi25_to_common_graph

*Intent* The intention of this web service is to convert a network represented using the PSI-MI Level 2.5 XML-based format to the common graph format, retaining all the interactions and/or reactions in the network required for network analysis.

*Motivation* PSI-MI Level 2.5 format is used to represent PPI networks. The conversion of this format to the common graph format prepares a network for submission to the data analysis web services, and avoids the situation in which a custom set of analysis tasks must be created for each format. A number of databases provide PPI to download in this format, for example BIND, HPRD and DIP. The PSI-MI XML format is very large compared to the common graph representation, and so the reduced file size enables faster data transfer and analysis.

*Applicability* The web service is applicable when the user wishes to analyse a network represented using the PSI-MI Level 2 format.

### 5.4.6.3 psitab_to_common_graph

*Intent* The intention of this web service is to convert a network represented using the PSI-MI tab-delimited format to the common graph format, retaining all the interactions and/or reactions in the network required for network analysis.

*Motivation* PSI-MI tab-delimited format is used to represent PPI networks. The conversion of this format to the common graph format prepares a network for submission to the data analysis web services, and avoids the situation in which a custom set of analysis tasks must be created for each format. A number of databases provide PPI to download in this format, for example IntAct and MINT. The PSI-MI tab-delimited format is very large compared to the common graph representation, and so the reduced file size enables faster data transfer and analysis.

*Applicability* The web service is applicable when the user wishes to analyse a network represented using the PSI-MI tab-delimited format.
5.4.6.4  sbml_to_common_graph

Intent  The intention of this web service is to convert a network represented using the SBML Level 2 XML-based format to the common graph format, retaining all the interactions and/or reactions in the network required for network analysis.

Motivation  SBML Level 2 format is one of a number of formats used to represent biological networks, commonly metabolic networks. The conversion of this format to the common graph format prepares a network for submission to a number of data analysis web services, and avoids the situation where a custom set of analysis tasks must be created for each format. The SBML format is very large compared to the common graph representation, and so the reduced file size enables faster data transfer and analysis.

Applicability  The web service is applicable when the user wishes to analyse a network represented using the SBML Level 2 format.

5.4.6.5  sif_to_common_graph

Intent  The intention of this web service is to convert a network represented using SIF to the common graph format, retaining all the interactions and/or reactions in the network required for network analysis.

Motivation  SIF is one of a number of formats used to represent biological networks. The conversion of this format to the common graph format prepares a network for submission to a number of data analysis web services, and avoids the situation where a custom set of analysis tasks must be created for each format.

Applicability  The web service is applicable when the user wishes to analyse a network represented using SIF.
Chapter 6

Biological Observations

The aim of this chapter is to describe computational workflows that are relevant to biological research, created within the Taverna workbench. They use web services developed for this work as well as those developed by external providers.

As discussed in Chapter 3, Taverna uses processor types to abstract over the different service interfaces, meaning that workflows define a set of processors between which data are passed. The framework defines categories which are used to guide workflow creation, by specifying an order in which processors are linked together. Within the context of network construction and analysis, the processors perform specific functions, that is, one of either data retrieval, data transformation, data analysis, or output formatting. However there is sometimes also a need for generic control processors to be included in a workflow, to ensure that data are routed correctly.

The construction of each workflow in this Chapter is therefore guided by the framework developed in Chapter 3. For each workflow, the following are documented:

- The biological motivation for developing the workflow.
- The workflow description, which comprises a summary of the processors used and the Taverna workflow diagram.
- A table of processors used in the workflow, giving the name of each processor and it’s framework category, type and detailed description.
- An example input to the workflow.
- The interpretation of the result obtained.
6.1 Holistic network analysis

6.1.1 Motivation

Various topological properties may be calculated for holistic networks. These properties characterise the global structure of the network, and may be used as a starting point for further investigation. As described in previously-mentioned studies, biological networks of various types demonstrate a scale-free topology, and have a hierarchical structure, leading to properties such as modularity, local clustering and a heterogeneous degree distribution. This workflow may therefore be used to establish if a real network of interactions derived through experimental means possesses these properties, and the implications this has for its global structure.

6.1.2 Workflow description

The workflow begins with a data transformation step, which parses the human PPI interactions obtained from the MINT database in the tab-delimited PSI-MI format, and returns the same interactions in the common graph format. The resulting network is then passed to 13 data analysis processors. These are all for undirected graphs, as PPI networks encode binary interactions where directionality is not recorded. Most of the results of these analysis operations are returned directly, however two output formatting steps are applied to enable the visualisation of one result, the largest clique in the network, as a layout diagram. The workflow diagram is shown in Figure 6.1

6.1.3 Table of workflow processors

The names of some Soaplab processors in this workflow have been contracted to make the workflow diagram less cluttered. The full names of the processors used are given in brackets.

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>psi_tab_to_common_graph</td>
<td>Data transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>remove_singleton_nodes</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>list_CC (list_connected_components _undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>DEG (rank_degrees_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>FC (find_cliques_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
</tbody>
</table>
6.1 Holistic network analysis

<table>
<thead>
<tr>
<th>Processor name (cont.)</th>
<th>Framework category (cont.)</th>
<th>Type (cont.)</th>
<th>Details (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sizeCC (size_distribution_connected_components_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>LCC (largest_connected_component_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>subLCC (get_subgraph_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>DIA (get_network_diameter_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>EDGES (total_edges_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>NODES (total_nodes)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>SLC (size_largest_clique_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>CBS (get_clique_by_size_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>subLClique (get_subgraph_undirected)</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>cg2dot (common_graph_to_dot_undirected)</td>
<td>Output transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>neato</td>
<td>Output rendering</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
</tbody>
</table>

6.1.4 Input

Human PPIs from MINT in PSI-TAB format. The latest version is available from the MINT FTP site at ftp://mint.bio.uniroma2.it/pub/release/MITAB/current. This workflow was executed using the file “2009-04-14-mint-human.mitab25.txt” (April 2009 version).

6.1.5 Output interpretation

There are 10 outputs produced as a result of executing this workflow:

- list_cc A list of the connected components in the network, separated by newline characters
- diameter The diameter of the largest connected component
- cliques A list of the cliques in the network, separated by newline characters
6.1 Holistic network analysis

Figure 6.1: Topological metrics for a holistic PPI network

- **deg_ranking** A list of all the proteins in the network, ranked by their degrees, from highest to lowest
- **largest_clique** The largest clique in the network rendered as a diagram
- **total_nodes** The total number of proteins in the network
- **total_edges** The total number of interactions in the network
- **size_CC** The size distribution of the connected components in the network
- **size_LClique** The number of proteins in the largest clique
- **singles_from_psitab** A list of singleton proteins in the network

The data transformation step converts the PSI-MI tab-delimited format to the common graph format. If an interaction is recorded where at least one participant is not represented using a Uniprot identifier, this protein is removed from the network. This leads to a number of proteins existing in the network as singleton nodes, which are removed as they have the potential to skew the results of downstream analyses. The removed proteins are recorded in a separate workflow output.

The resulting network consists of 7170 proteins and 16110 interactions. The largest connected component consists of 6592 proteins. Despite this relatively large size, the
6.2 Cycle identification

diameter of this component is 14, and as this is the greatest distance between all pairs of proteins in the network, it implies the existence of proteins which are highly connected. The list of connected components shows the isolated subgraphs in the network. While the largest has been selected for further analysis here, any of the others may be examined further to establish why they do not join up to the main component, or indeed if they should at all. The list is useful when checking the biological completeness of the PPI network.

The degree ranking reveals that the top twenty degrees range from 311 down to 89, a sharp drop-off. In fact, as expected for a large-scale network, there are a large number of low-degree proteins, versus a relatively lower number of high-degree proteins. The degree ranking may be used to investigate so-called protein hubs, or those proteins which bind a large number of other proteins and therefore may be globally important.

The number of proteins in the largest clique is ten. While this clique was selected for visualisation, the list of cliques generated may be used to select other potentially interesting clique sizes. The largest clique has been rendered as a layout diagram (see Figure 6.2). A clique in a PPI network is a set of proteins that all bind each other, and a clique of this size may have a biologically significant role. The significance of this set of bindings may depend on temporal factors, that is, whether the proteins bind each other all at the same time, or if different proteins in the clique come together at different times, perhaps under different conditions. To investigate this further, the annotations for each protein were retrieved, and can be seen in Table 6.2 This is a biologically meaningful clique, as it represents RNA polymerase II, a multi-protein complex whose constituent proteins bind simultaneously, and attach to DNA in order to initiate transcription and produce complementary RNA chains. Of the two non-RNA polymerase proteins, Q92830 (general control of amino acid synthesis) is described in Uniprot as functioning as histone acetyltransferase to promote transcriptional activity. Q9Y4A5 (transformation/transcription domain-associated protein) is an adapter protein which is associated with histone acetyltransferase activity.

6.2 Cycle identification

6.2.1 Motivation

Identification of cycles in biological networks is crucial to acquire a physiological perspective on network behaviour. Cycles are sequences of interactions or reactions in a cell which are used to reinforce certain cell-scale mechanisms: homeostasis, oscillation,
6.2 Cycle identification

Figure 6.2: Largest clique in the human PPI network retrieved from the MINT database, rendered as a network layout diagram.

<table>
<thead>
<tr>
<th>Uniprot identifier</th>
<th>Recommended name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q9UHV7</td>
<td>Mediator of RNA polymerase II transcription subunit 13</td>
</tr>
<tr>
<td>O75448</td>
<td>Mediator of RNA polymerase II transcription subunit 24</td>
</tr>
<tr>
<td>Q92830</td>
<td>General control of amino acid synthesis protein 5-like 2</td>
</tr>
<tr>
<td>Q9Y2X0</td>
<td>Mediator of RNA polymerase II transcription subunit 16</td>
</tr>
<tr>
<td>Q93074</td>
<td>Mediator of RNA polymerase II transcription subunit 12</td>
</tr>
<tr>
<td>Q9Y4A5</td>
<td>Transformation/transcription domain-associated protein</td>
</tr>
<tr>
<td>Q15648</td>
<td>Mediator of RNA polymerase II transcription subunit 1</td>
</tr>
<tr>
<td>Q9NVC6</td>
<td>Mediator of RNA polymerase II transcription subunit 17</td>
</tr>
<tr>
<td>O60244</td>
<td>Mediator of RNA polymerase II transcription subunit 14</td>
</tr>
<tr>
<td>Q9ULK4</td>
<td>Mediator of RNA polymerase II transcription subunit 23</td>
</tr>
</tbody>
</table>

Table 6.2: Annotations for members of the largest clique in the human PPI network retrieved from MINT.

stress response and developmental growth. By reducing a network to its constituent cycles, it is possible to deduce which, if any, cycles are regulating these cell-scale behaviours, and whether they are involved in positive or negative feedback.

6.2.2 Workflow description

The workflow commences with a data transformation step, to prepare the network for analysis. In this example, an SBML model is transformed into a common graph representation. Two analysis steps for directed networks are then applied, to obtain the subgraph of the cyclic core of the network, again represented using the common
6.2 Cycle identification

graph format. The output formatting step `format_sbm1_id_list` is preceded and followed by a number of generic shims to ensure that the SBML identifiers are converted to the corresponding descriptive names. The two final output formatting processors are applied to render the cyclic core of the network as a layout diagram. The workflow diagram is shown in Figure 6.3.

6.2.3 Table of workflow processors

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbml_to_common_graph</td>
<td>Data transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>get_cyclic_core</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>get_subgraph_directed</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>split</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the string output from get_subgraph_directed into a list of strings</td>
</tr>
<tr>
<td>split1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the string output from split into a list of strings</td>
</tr>
<tr>
<td>format_sbm1_id_list</td>
<td>Output transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>join_tab</td>
<td>n/a - generic control</td>
<td>Beanshell processor</td>
<td>Beanshell script which joins the string list output from format_sbm1_id_list using a tab character, to form a string</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from join_tab to a string</td>
</tr>
<tr>
<td>common_graph_dot</td>
<td>Output transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>neato</td>
<td>Output rendering</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
</tbody>
</table>

6.2.4 Input

SBML model of cholesterol metabolism from GeneNet (Ananko et al., 2005), downloaded from http://www.bionet.nsc.ru/mgs/gnw/gn_model/List_of_Models_SBML.shtml. The version used was model number six, “Cholesterol_MODEL”.

104
6.2.5 Output interpretation

The reduction of the model of cholesterol metabolism to its cyclic core, and subsequent pictorial depiction enables the visualisation of cycles in the network. The layout diagram generated as a result of running this workflow with the specified input is shown in Figure 6.4. This example is a test case as the cycles in cholesterol metabolism are well characterised, and owing to the relatively small size of the network, are suitable for visualisation.

The regulatory edges and nodes involved are highlighted in the layout diagram. The presence of cholesterol affects the formation of the sterol regulatory element binding protein (SREBP1 in the diagram), through reaction 1568. SREBP1 acts as a transcription factor to stimulate the transcription of many genes, including HMG-CoA reductase (Hs:HMGCR in the diagram), through reaction 319. This is the rate-controlling enzyme of the mevalonate pathway, and converts HMG-CoA to mevalonic acid. Cholesterol is the final product of the pathway, and therefore regulates its own synthesis. In the presence of cholesterol, SREBP1 forms a complex with two other proteins: SREBP-cleavage activating protein (SCAP) and Insig1. When cholesterol levels fall, Insig1 dissociates from the SREBP-SCAP complex, which then migrates to the Golgi apparatus. SREBP is then cleaved by two enzymes, and migrates to the nucleus, activating the expression of HMG-CoA reductase, thereby initiating the production of more cholesterol. This is an example of a negative feedback cycle leading to homeostasis.

6.3 Local networks

6.3.1 Motivation

Given a whole PPI network, and a particular protein of interest in that network, it is often useful to isolate the local network around that protein. In this context the local network refers to the set of proteins up to a specified number of edges away from that network, as well as any interconnections between proteins in that set. A protein is more likely to interact with another protein whose role it shares (Nabieva et al., 2005; Sharan et al., 2007). Viewing a protein in the context of its local network, as well as examining the topology of the local network, may shed further light on its metabolic or regulatory role.
6.3.2 Workflow description

A data retrieval step is called, either once, twice or three times, depending on the size of local network required. In this example, query atpid is called first on the query protein, then twice more on the subsequent interacting proteins, to find all the proteins up to three edges away from the query. A set of processors for generic control (Beanshells and Local Processors) are used to manipulate the lists of proteins identified as being in the local network, to ensure that all proteins are retained in the final list. Another data retrieval step is a nested workflow which retrieves the GO terms for each protein. The data are then passed through an analysis step, to return the subgraph comprising the proteins in the local network. Two output formatting steps are applied to render the local network as a layout diagram. The layout diagram is directed into one output, while the list of proteins in the local network, and their associated GO terms are directed into another output.

The main workflow diagram is shown in Figure 6.5. The workflow contains four nested workflows, first_edge, second_edge and third_edge which are all the same workflow, and get_GO_for AGI. These workflows are shown in Figure 6.6.
Figure 6.3: Identification of cycles in a metabolic network model
Figure 6.4: Cycle identification workflow result for the model of cholesterol metabolism.

The regulatory edges and corresponding nodes are highlighted in red. Cholesterol affects the formation of SREBP1 which activates the expression of HMGCR, and therefore regulates its own synthesis through negative feedback, to achieve homeostasis.
Figure 6.5: Generation of the local network around a protein of interest in the AtPID dataset
6.3 Local networks

(a) Query the AtPID database to retrieve interacting proteins for a given query protein

(b) Retrieve Gene Ontology terms for *A. thaliana* proteins using a BioMoby web service

Figure 6.6: Nested workflows used in a workflow to generate a local network around an *A. thaliana* protein in AtPID
### Table of workflow processors

Table of processors for main workflow:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_edge</td>
<td>Data retrieval</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>second_edge</td>
<td>Data retrieval</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>third_edge</td>
<td>Data retrieval</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>get_GO_for_AGI</td>
<td>Data retrieval</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>merge_first_interactors</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from first_edge by one level</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from second_edge by one level</td>
</tr>
<tr>
<td>merge_second_interactors</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from merge by one level</td>
</tr>
<tr>
<td>concat1</td>
<td>n/a - generic control</td>
<td>Beanshell</td>
<td>Concatenates two lists of proteins, merge_first_interactors and merge_second_interactors</td>
</tr>
<tr>
<td>merge1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from third_edge by one level</td>
</tr>
<tr>
<td>merge2</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from merge1 by one level</td>
</tr>
<tr>
<td>merge_third_interactors</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from merge2 by one level</td>
</tr>
<tr>
<td>concat2</td>
<td>n/a - generic control</td>
<td>Beanshell</td>
<td>Concatenates two lists of proteins, concat1 and merge_third_interactors</td>
</tr>
<tr>
<td>split</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the string output from concat2 into a list of strings</td>
</tr>
<tr>
<td>remove_dups</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Removes duplicate strings from the list output of split</td>
</tr>
<tr>
<td>merge3</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from remove_dups by one level</td>
</tr>
</tbody>
</table>
Table of processors for data retrieval workflow **first_edge** (second_edge and third_edge are identical, so the corresponding tables are not shown):

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_atpid</td>
<td>Data retrieval</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>split</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the string output from query_atpid into a list of strings</td>
</tr>
</tbody>
</table>

Table of processors for data retrieval workflow **get_GO_for_AGI**:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>n/a - generic control</td>
<td>BioMoby Object Ontology node</td>
<td>Takes as input a “namespace” and “id” to produce mobyData</td>
</tr>
<tr>
<td>Locus2GOIDs</td>
<td>Data retrieval</td>
<td>BioMoby processor</td>
<td>BioMoby web service provided by arabidopsis.org, accepts output of Object and returns corresponding GO terms</td>
</tr>
<tr>
<td>Parse_Moby_Data_GO_Term</td>
<td>n/a - generic control</td>
<td>BioMoby parser</td>
<td>Parses the output from Locus2GOIDs to enable extraction of the required content</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local Processor</td>
<td>Flattens the string list output from Parse_Moby_Data.GO.Term by one level</td>
</tr>
<tr>
<td>merge1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from merge by one level</td>
</tr>
<tr>
<td>join</td>
<td>n/a - generic control</td>
<td>Beanshell processor</td>
<td>Beanshell script to prefix the query protein identifier to the list of corresponding GO terms</td>
</tr>
</tbody>
</table>

### 6.3.4 Input

An AGI identifier corresponding to a protein of interest. For the example, the identifier AT4G03460 is used. This protein contains ankyrin repeat domains. The GO molecular
function is identified as ‘protein binding’, but the GO biological process and GO cellular component are both unknown. The maximum number of edges away from the query protein is set to 3. In practice, generating a network any larger than this is impractical, as the small-world nature of the global network means that most of the network would be returned.

### 6.3.5 Output interpretation

The two outputs of the workflow are shown in Figure 6.7 (layout diagram of local network) and Table 6.7 (list of proteins in the local network, with associated GO terms). As mentioned, there are no associated GO terms for the query AT4G03460, along with nine other proteins in the local network. AT4G03460 interacts with nine proteins directly. The GO terms for some of these proteins include ‘sulfate transport’, ‘membrane’, ‘integral to membrane’ and ‘protein amino acid phosphorylation’. One protein that interacts with the query, ATG312520, is involved in a clique with 5 other proteins. The GO terms associated with proteins in this clique strongly suggest a functional module relating to sulfate transport across membranes, so the fact the query is connected to this clique strengthens the assumption that its role is related.

Another area of interest in the local network is a second clique, not directly connected to the query, but instead two edges from it, via AT1G03670. This clique contains seven proteins, but there is only one protein with any annotation: the protein AT5G50390 is associated with ‘chloroplast’. Also connected to the query via AT1G03670 is an area of the local network which is not a clique, but is richly annotated. The annotations are not as unifying as the first clique discussed above, in fact the only term that appears more than once is ‘protein amino acid phosphorylation’. This is surprising, as across these eight interacting proteins, there are 22 terms in total, with 21 distinct terms, implying a rather heterogeneous grouping of proteins. Another collection of interacting proteins connected to the query indirectly however share many functional terms despite not being a clique. The gateway to this part of the network is AT3G53810, and these proteins are involved in ‘protein amino acid phosphorylation’, ‘endomembrane system’ and ‘plasma membrane’. In fact so many of these proteins share similar GO annotations that there may be some interactions occurring which have not been computationally inferred or experimentally derived, but do in fact exist.
### 6.3 Local networks

<table>
<thead>
<tr>
<th>AGI identifier</th>
<th>Associated GO terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT1G03670</td>
<td>None</td>
</tr>
<tr>
<td>AT5G26710</td>
<td>cytoplasm; tRNA aminoacylation for protein translation; glutamyl-tRNA aminoacylation; translation</td>
</tr>
<tr>
<td>AT5G60300</td>
<td>plasma membrane</td>
</tr>
<tr>
<td>AT3G45430</td>
<td>protein amino acid phosphorylation</td>
</tr>
<tr>
<td>AT5G39350</td>
<td>mitochondrion</td>
</tr>
<tr>
<td>AT3G12520</td>
<td>sulfate transport; transport; membrane; integral to membrane</td>
</tr>
<tr>
<td>AT1G77990</td>
<td>sulfate transport; transport; membrane; integral to membrane</td>
</tr>
<tr>
<td>AT3G04910</td>
<td>membrane</td>
</tr>
<tr>
<td>AT5G03730</td>
<td>None</td>
</tr>
<tr>
<td>AT5G46240</td>
<td>None</td>
</tr>
<tr>
<td>AT3G53810</td>
<td>protein amino acid phosphorylation; endomembrane system</td>
</tr>
<tr>
<td>AT5G10180</td>
<td>membrane; integral to membrane</td>
</tr>
<tr>
<td>AT1G20230</td>
<td>None</td>
</tr>
<tr>
<td>AT3G12770</td>
<td>None</td>
</tr>
<tr>
<td>AT5G50390</td>
<td>chloroplast</td>
</tr>
<tr>
<td>AT3G14730</td>
<td>None</td>
</tr>
<tr>
<td>AT3G46790</td>
<td>None</td>
</tr>
<tr>
<td>AT3G24000</td>
<td>None</td>
</tr>
<tr>
<td>AT4G03460</td>
<td>None</td>
</tr>
<tr>
<td>AT4G02420</td>
<td>protein amino acid phosphorylation; endomembrane system</td>
</tr>
<tr>
<td>AT3G55550</td>
<td>protein amino acid phosphorylation; plasma membrane</td>
</tr>
<tr>
<td>AT3G51895</td>
<td>sulfate transport; transporter activity; transport; membrane; integral to membrane; secondary active sulfate transmembrane transporter activity</td>
</tr>
<tr>
<td>AT3G15990</td>
<td>sulfate transport; transport; membrane; integral to membrane</td>
</tr>
<tr>
<td>AT1G23090</td>
<td>sulfate transport; transport; membrane; integral to membrane</td>
</tr>
<tr>
<td>AT5G13550</td>
<td>sulfate transport; transport; membrane; integral to membrane</td>
</tr>
<tr>
<td>AT1G78000</td>
<td>membrane; integral to membrane</td>
</tr>
<tr>
<td>AT3G09790</td>
<td>cell wall; vacuole</td>
</tr>
<tr>
<td>AT3G45440</td>
<td>protein amino acid phosphorylation; endomembrane system</td>
</tr>
<tr>
<td>AT5G67200</td>
<td>protein amino acid phosphorylation; ATP binding; plasma membrane; plasma membrane; protein kinase activity; protein binding</td>
</tr>
</tbody>
</table>
### Table 6.7: List of proteins in the local network of AT4G03460 and their associated GO terms

<table>
<thead>
<tr>
<th>AGI identifier (cont.)</th>
<th>Associated GO terms (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT4G29050</td>
<td>protein amino acid phosphorylation; endomembrane system</td>
</tr>
<tr>
<td>AT5G25150</td>
<td>regulation of transcription; transcription regulator activity; nucleus</td>
</tr>
<tr>
<td>AT3G55400</td>
<td>chloroplast</td>
</tr>
<tr>
<td>AT5G39960</td>
<td>GTP binding; intracellular</td>
</tr>
<tr>
<td>AT5G13520</td>
<td>zinc ion binding; binding; proteolysis; leukotriene biosynthetic process; metallopeptidase activity</td>
</tr>
<tr>
<td>AT5G60310</td>
<td>protein amino acid phosphorylation; endomembrane system</td>
</tr>
<tr>
<td>AT4G15720</td>
<td>None</td>
</tr>
<tr>
<td>AT3G22330</td>
<td>nucleic acid binding; helicase activity; ATP-dependent helicase activity; ATP binding; cell wall; nucleolus</td>
</tr>
</tbody>
</table>
Figure 6.7: Local network around the query *A. thaliana* protein AT4G03460. The dot file produced by the workflow is modified to highlight subgraphs of note in the network. The query protein is shown in red. The green proteins form a clique, for which there is only one member, AT5G50390, with any annotation. The yellow proteins also form a clique, whose members are annotated with several common GO terms, suggesting this is a functional clique, relating to sulfate transport across membranes. The blue proteins do not form a clique, but do share many similar GO terms.
6.4 Source and sink metabolites in a network model of metabolism

6.4.1 Motivation

Source and sink nodes in a directed biological network such as one representing metabolism are those which are at the start or end of enzymatic pathways, and are not themselves produced or consumed, respectively. If a model of metabolism is complete and accurate then the lists of sources and sinks should consist of only essential substrates and fermentation products, respectively. Production of both lists is therefore a key stage of model curation, to suggest reactions that are either erroneous, missing or incomplete. We would also hope to see elements in both lists which are expected to be present.

6.4.2 Workflow description

The workflow starts with a data retrieval step, to obtain the SBML model for analysis from the BioModels database. A data transformation step converts the SBML to the common graph format, which is then submitted to two data analysis steps, to get a list of the source and sink nodes. The nodes are then formatted so that SBML identifiers are replaced with more meaningful names. The workflow diagram is shown in Figure 6.8.

6.4.3 Table of workflow processors

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_model_by_id</td>
<td>Data retrieval</td>
<td>WSDL processor</td>
<td>Web service interface to the BioModels repository at the EBI</td>
</tr>
<tr>
<td>sbml_to_common_graph</td>
<td>Data transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>get_sink_nodes_directed</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>get_source_nodes_directed</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>format_sbml_id_list_sinks</td>
<td>Output transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>format_sbml_id_list_sources</td>
<td>Output transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
</tbody>
</table>

6.4.4 Input

An SBML model of human metabolism, developed by Duarte et al. (2007). The layout diagram for this network may be seen in Chapter 2 (Figure 2.1). The model has been de-
6.4 Source and sink metabolites in a network model of metabolism

Figure 6.8: Identification of source and sink nodes in a metabolic network posited in the BioModels repository at the EBI, with the identifier “MODEL6399676120”.

6.4.5 Output interpretation

Both lists of source and sink nodes are subject to errors caused by a number of issues. Curation of a network model such as the one analysed may lead to the following:

- Specific reaction terms, to which generic terms for metabolites do not connect
- Generic reaction terms, to which specific terms for metabolites do not connect
- Macromolecule modification, to which terms for a specific part of a molecule do not connect
- Non-enzymatic reactions, if not represented, result in terms which do not connect
- Reaction intermediates, if not represented, result in terms which do not connect

The results of the workflow give 112 source nodes and 161 sink nodes. The following nodes have been highlighted to try to deduce the biological reasons for their appearance in lists of sources and sinks. The full lists are available in Appendix D.
6.4.5.1 Source nodes

D-Proline, C5H9NO2
This is a metabolite that is synthesised in microbes, but not humans. It is an example of an external compound for which humans have a coping mechanism.

These are all monophosphates, which are specific mononucleotides produced by a generic reaction.

cocaine, C17H21NO4
This is another example of an external compound humans are able to break down, but would certainly not be considered an essential substrate. However as a reaction equation exists for it, and it is not synthesised by humans, it appears as a source.

hydroxy alkyl chain, C2H5OFULLR
This is an example of a generic metabolite term, produced by a specific reaction.

These metabolites have a role in steroid hormone biochemistry. They are further examples of enzymes included in the model with generic reaction terms. They have a role in detoxifying compounds introduced through diet.

(R)-Pantothenate, C9H16NO5
This is vitamin B5, and is an example of a genuine essential substrate, as it is needed to form coenzyme-A and is vital for carbohydrate, protein and fat metabolism. Humans can obtain it in small quantities from food, especially whole grains, eggs and legumes.

6.4.5.2 Sink nodes

dGTP, C10H12N5O13P3, dATP, C10H12N5O12P3, dCTP, C9H12N3O13P3, dTTP, C10H13N2O14P3
These are all DNA precursors. Interestingly, the node “DNA” appears in the list of sources, owing to there being no reaction present in the model to represent DNA synthesis. If this was introduced, then the precursors and DNA itself would be removed from both lists. The following two irreversible reactions in the SBML include DNA:

- S-Adenosyl-L-methionine, C15H23N6O5S + DNA, C10H17O8PR2 -->
  S-Adenosyl-L-homocysteine, C14H20N6O5S + DNA 5-methylcytosine, C11H19O8PR2 + H+ + H

- DNA, C10H17O8PR2 + S-Adenosylselenomethionine, C15H23N6O5Se -->
  DNA 5-methylcytosine, C11H19O8PR2 + H+ + S-Adenosylselenohomocysteine, C14H20N6O5Se
6.5 Annotating metabolic pathways with PPIs

6.5.1 Motivation

This workflow was developed to establish if, for a given metabolic pathway, any enzymes in the pathway are mediated by PPIs, as this may be used to generate hypotheses about the regulation of that pathway. The workflow accepts as input a KEGG pathway identifier, and then retrieves the corresponding pathway and its constituent enzyme identifiers. These are queried against a PPI resource (in this case IntAct) to discover if any proteins interact with more than one of these pathway enzymes.

6.5.2 Workflow description

The first three steps in the workflow are data retrieval: first, to obtain all the genes along a given pathway, second, to obtain the corresponding Uniprot protein identifiers for these genes, and third, to query the IntAct database to find out which proteins interact with enzymes along the pathway. A data transformation step is then applied to each result from the IntAct database, to convert each one into the common graph format. A data analysis step is then applied, to create the adjacency matrix of each IntAct result, and query it with the pathway enzyme. For each enzyme, this then produces an adjacency list representation, i.e. the enzyme followed by a list of the proteins it interacts with. All the adjacency lists are then merged to create a single file, which is then analysed using reverse_adjacency_list to discover which proteins interact with which enzymes. A final output formatting step is applied to provide annotations.
for the proteins in this reversed adjacency list.

The main workflow diagram is shown in Figure 6.9. The expanded nested workflows are shown in Figure 6.10 and Figure 6.11.

### 6.5.3 Table of workflow processors

Table of processors for main workflow:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_genes_by_pathway</td>
<td>Data retrieval</td>
<td>WSDL Processor</td>
<td>Web service interface to the KEGG database</td>
</tr>
<tr>
<td>get_uniprot_id _from_kegg_gene</td>
<td>Data retrieval</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>query_intact</td>
<td>Data retrieval</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>psitab_to_common_graph</td>
<td>Data transformation</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
<tr>
<td>query_adjacency_matrix _undirected</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from query_adjacency_matrix _undirected by one level</td>
</tr>
<tr>
<td>remove_blank_lines _from_file</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>reverse_adjacency_list</td>
<td>Data analysis</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>split</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the output from reverse_adjacency_list into a list of strings</td>
</tr>
<tr>
<td>extract</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Extracts the first 20 items from the results of split</td>
</tr>
<tr>
<td>merge1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from merge1 by one level</td>
</tr>
<tr>
<td>format_results</td>
<td>Output transformation</td>
<td>Nested workflow</td>
<td>See following tables</td>
</tr>
</tbody>
</table>
6.5 Annotating metabolic pathways with PPIs

Table of processors for data retrieval workflow get_uniprot_id_from_kegg_gene:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_linkdb_by_entry</td>
<td>Data retrieval</td>
<td>WSDL processor</td>
<td>Web service interface to the LinkDB database</td>
</tr>
<tr>
<td>returnXML</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Retrieves the internal data elements from the XML output of get_linkdb_by_entry</td>
</tr>
<tr>
<td>returnXML1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Retrieves the internal data elements from the XML output of returnXML</td>
</tr>
<tr>
<td>split</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the string output of returnXML1 into a list of strings</td>
</tr>
<tr>
<td>extract</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Selects the required items from the list output of split</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from extract by one level</td>
</tr>
<tr>
<td>merge1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from merge by one level</td>
</tr>
</tbody>
</table>

Table of processors for data retrieval workflow query_intact:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>infoRequestXML</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Presents the internal data elements for the XML input to parametersXML</td>
</tr>
<tr>
<td>parametersXML</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Presents the internal data elements for the XML input to getByQuery</td>
</tr>
<tr>
<td>getByQuery</td>
<td>Data retrieval</td>
<td>WSDL Processor</td>
<td>Web service interface to the IntAct database</td>
</tr>
<tr>
<td>parametersXML1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Retrieves the internal data elements from the XML output of getByQuery</td>
</tr>
<tr>
<td>queryResponseXML</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Retrieves the internal data elements from the XML output of parametersXML1</td>
</tr>
<tr>
<td>resultSetXML</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Retrieves the internal data elements from the XML output of queryResponseXML</td>
</tr>
</tbody>
</table>

122
6.5 Annotating metabolic pathways with PPIs

Table of processors for data transformation workflow `psitab_to_common_graph`:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>psitab_to_common_graph</code></td>
<td>Data transformation</td>
<td>Soaplab processor</td>
<td>See Appendix C</td>
</tr>
</tbody>
</table>

Table of processors for output transformation workflow `format_results`:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>split</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the input to the workflow into a list of strings</td>
</tr>
<tr>
<td>split1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the output of split into a list of strings</td>
</tr>
<tr>
<td>split2</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Splits the output of split1 into a list of strings</td>
</tr>
<tr>
<td>concat</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Prefixes each output of split2 with the string “uniprotkb:”</td>
</tr>
<tr>
<td>get_sp_info</td>
<td>Output formatting</td>
<td>Nested workflow</td>
<td>See following table</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from get_sp_info by one level</td>
</tr>
<tr>
<td>add_text</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Beanshell to insert text to make the output more readable</td>
</tr>
<tr>
<td>merge1</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from add_text by one level</td>
</tr>
</tbody>
</table>

Table of processors for output transformation workflow `get_sp_info`:

<table>
<thead>
<tr>
<th>Processor name</th>
<th>Framework category</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetchData</td>
<td>Data retrieval</td>
<td>WSDL processor</td>
<td>Web service interface to Dbfetch at the EBI</td>
</tr>
<tr>
<td>xpath</td>
<td>n/a - generic control</td>
<td>String constant</td>
<td>XPath query string</td>
</tr>
<tr>
<td>xpath_process</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Retrieves text from XML document given the string in xpath</td>
</tr>
<tr>
<td>merge</td>
<td>n/a - generic control</td>
<td>Local processor</td>
<td>Flattens the string list output from xpath_process by one level</td>
</tr>
<tr>
<td>join</td>
<td>n/a - generic control</td>
<td>Beanshell processor</td>
<td>Beanshell script which joins the string list output from merge using a tab character, to form a string</td>
</tr>
</tbody>
</table>
6.5 Annotating metabolic pathways with PPIs

6.5.4 Input

The input to this workflow is a KEGG pathway identifier, in this case representing glycolysis in *S. cerevisiae* (path: sce00010). Glycolysis lies at the heart of sugar and amino acid metabolism, and energy supply.

6.5.5 Output interpretation

The full results of the workflow are shown in Appendix D. The set of interactions retrieved was used to manually generate a network layout diagram using Cytoscape, under the supervision of an experienced biochemist who examined the workflow results to suggest potentially significant groups of interactions. The aim of the diagram was to depict the local network around the enzymes which are part of the yeast glycolytic pathway, as an enhancement of the text output generated by the workflow. The diagram is shown in Figure 6.12. The glycolytic enzymes are shown in the order they appear along the pathway, connected by dashed red lines. Proteins (nodes) are coloured according to their role and the interactions (edges) are coloured according to the detection method. Attribute files for the nodes and edges were used in conjunction with the VizMapper tool in Cytoscape to colour nodes and edges appropriately. The diagram presents some interesting hypotheses with regard to regulation of glycolysis.

The first part of the glycolytic pathway consumes energy (ATP): glucose is phosphorylated to produce glucose 6-phosphate, which is then rearranged to form fructose-6-phosphate and phosphorylated again. The latter part of glycolysis is an “energy production unit”, as the sequence of enzymatic reaction produce more ATP than is consumed in the first part. It is remarkable how many proteins are interacting with multiple enzymes. This probably has implications for energy supply but the details are unclear at this stage. However, the protein YD161_YEAST is particularly interesting. Of the 35 proteins to which, according to IntAct, it interacts, 7 are consecutive enzymes in the lower half of the glycolytic pathway (see Table 6.15). Since there are 6607 genes in yeast, the probability of YD161_YEAST interacting with one enzyme is 0.005. Hence the probability of YD161_YEAST interacting with these 7 enzymes is $2.99^{-16}$. Of the other YD161_YEAST interactors, twelve are ribosomal subunits, 4 are involved in nucleic acid biology, 2 are involved in ATP synthase, 5 are heat-shock/proteosome-related, and 4 are miscellaneous cytoplasmic enzymes. This leads to the hypothesis that YD161_YEAST placing an energy production unit where it is required. There are two possibilities for its mechanism of action:

- It binds all the glycolytic enzymes at the same time, thus facilitating the creation
of the energy production unit by bringing together those enzymes needed in order to quickly generate large amounts of ATP.

- It binds the glycolytic enzymes one at a time, thus enabling their translocation to different parts of the cell, depending on where they are needed.

The observations have been shared with yeast biologists at the University of Nottingham, who are planning biophysical experiments to test these hypotheses.

<table>
<thead>
<tr>
<th>Uniprot identifier</th>
<th>Protein name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glycolysis</td>
<td>Fructose-bisphosphate aldolase</td>
</tr>
<tr>
<td>ALF_YEAST</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>TPIS_YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>G3P3_YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>PMG1_YEAST</td>
<td>Mediator of RNA polymerase II transcription subunit 12</td>
</tr>
<tr>
<td>ENO2_YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>PYK1_YEAST</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>PDC1_YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>Ribosome</td>
<td>Elongation factor 1-alpha</td>
</tr>
<tr>
<td>EF1A_YEAST</td>
<td>40S ribosomal protein S0-A</td>
</tr>
<tr>
<td>EF2_YEAST</td>
<td>40S ribosomal protein S0-B</td>
</tr>
<tr>
<td>RS0A_YEAST</td>
<td>40S ribosomal protein S3</td>
</tr>
<tr>
<td>RS0B_YEAST</td>
<td>40S ribosomal protein S11</td>
</tr>
<tr>
<td>RS11A_YEAST</td>
<td>60S ribosomal protein L4-A</td>
</tr>
<tr>
<td>RL4A_YEAST</td>
<td>60S ribosomal protein L4-B</td>
</tr>
<tr>
<td>RL4B_YEAST</td>
<td>60S ribosomal protein L5</td>
</tr>
<tr>
<td>RL7A_YEAST</td>
<td>60S ribosomal protein L7-A</td>
</tr>
<tr>
<td>RL20_YEAST</td>
<td>60S ribosomal protein L20</td>
</tr>
<tr>
<td>RL0A_YEAST</td>
<td>60S acidic ribosomal protein P0</td>
</tr>
<tr>
<td>Nucleic acid biology</td>
<td></td>
</tr>
<tr>
<td>IF4A_YEAST</td>
<td>ATP-dependent RNA helicase eIF4A</td>
</tr>
<tr>
<td>BRR2_YEAST</td>
<td>Pre-mRNA-splicing helicase BRR2</td>
</tr>
<tr>
<td>MCM5_YEAST</td>
<td>Minichromosome maintenance protein 5 (ATPase)</td>
</tr>
<tr>
<td>IMB4_YEAST</td>
<td>Importin subunit beta-4 (espec. of ribosomal proteins)</td>
</tr>
<tr>
<td>ATP synthase</td>
<td>Vacuolar ATP synthase catalytic subunit A</td>
</tr>
<tr>
<td>VATA_YEAST</td>
<td>Vacuolar ATP synthase subunit A</td>
</tr>
<tr>
<td>VATB_YEAST</td>
<td>Vacuolar ATP synthase subunit B</td>
</tr>
<tr>
<td>Heat-shock/proteosome</td>
<td></td>
</tr>
<tr>
<td>HSP72_YEAST</td>
<td>Heat shock protein SSA2</td>
</tr>
<tr>
<td>HSP60_YEAST</td>
<td>Heat shock protein 60, mitochondrial</td>
</tr>
<tr>
<td>HSP75_YEAST</td>
<td>Heat shock protein SSB1</td>
</tr>
</tbody>
</table>
Table 6.15: Annotations for the 35 interacting proteins of YD161_YEAST, retrieved from IntAct. The proteins are grouped according to their role.

<table>
<thead>
<tr>
<th>Uniprot identifier</th>
<th>Protein name</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSC82_YEAST</td>
<td>ATP-dependent molecular chaperone HSC82</td>
</tr>
<tr>
<td>RPN1_YEAST</td>
<td>26S proteasome regulatory subunit RPN1</td>
</tr>
<tr>
<td>Misc. cytoplasmic</td>
<td></td>
</tr>
<tr>
<td>ADH1_YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
<tr>
<td>MPG1_YEAST</td>
<td>Mannose-1-phosphate guanylytransferase (also involved in cell-cycle progression)</td>
</tr>
<tr>
<td>PYR1_YEAST</td>
<td>Protein URA1 (hydrolyses ATP to form carbamoyl aspartate)</td>
</tr>
<tr>
<td>IMDH3_YEAST</td>
<td>Probable inosine-5-phosphate dehydrogenase IMD3</td>
</tr>
</tbody>
</table>

Another interesting result is the protein FAR11_YEAST (gene name Far11) which interacts with five of the nine glycolytic enzymes. The literature identifies Far11 as being a membrane protein involved in pheromone-induced G1 cell cycle arrest (Kemp and Sprague, 2003). From this analysis one can hypothesise that Far11 mediates cell cycle arrest by binding some or all of the glycolytic enzymes, effectively removing them from the cytoplasm. As glycolysis is a fundamental cellular process, its retardation could potentially lead to cell cycle arrest. A schematic showing how this may occur in the cell is shown in Figure 6.13.

6.6 Conclusion

This chapter has demonstrated how web services developed by the author can be combined with those from other service providers to ask biologically relevant questions of holistic interaction and reaction datasets, as well as of individual network entities. The most interesting result has arisen from the annotation of the yeast metabolic pathway with PPIs, in an unbiased and systematic fashion, as a potentially novel regulatory mechanism was hypothesised. The automation of all the stages involved in this workflow was made possible by wrapping data retrieval, data transformation, data analysis and output rendering operations as web services as per the recommendations of the Framework developed in Chapter 3, and querying a freely-available PPI dataset whose contents are also available through a web service interface.
Figure 6.9: Annotation of metabolic pathways with PPIs
6.6 Conclusion

(a) Retrieve Uniprot identifier given a KEGG gene identifier

(b) Query the IntAct database to retrieve interacting proteins for a given query protein

(c) Transform each PSI-MI TAB file into the common graph format

Figure 6.10: Nested workflows for (a) data retrieval and (b) data transformation, used in a workflow to annotate metabolic pathways with PPIs
Figure 6.11: Nested workflow for output formatting used in a workflow to annotate metabolic pathways with PPIs
6.6 Conclusion

Figure 6.12: Cytoscape layout diagram manually generated using the results from the workflow to annotate metabolic pathways with PPIs, using the yeast glycolytic pathway as input. The key for the diagram is as follows: [EDGES] blue represent interactions which are computationally inferred, pink represent interactions obtained using immunoprecipitation, green represent interactions obtained using tandem affinity purification. [NODES] yellow nodes are glycolytic enzymes, pink nodes are membrane proteins, green nodes are proteins related to nucleic acid biology, blue nodes are ribosomal proteins, purple nodes are proteins related to ATP synthase, white nodes are cytoskeletal proteins.
Figure 6.13: Schematic of cell showing possible role of Far11 in mediating pheromone-induced cell-cycle arrest. (a) Glycolysis continuing as normal in the cell; the black circles are enzymes and the arrows between them denote chemical reactions transforming one substrate into a product with enzymes catalysing these reactions. (b) In the presence of the pheromone, the membrane protein YNL127w binds the enzymes, removing them from the cytoplasm and slows down or stop glycolysis.
Chapter 7

Conclusions

Bioinformatics research commonly involves files on the researcher’s local computer, remote online databases and a set of analysis tools, both local and remote. As described in Chapter 3, the sort of ‘manual’ workflows which result from the pipelining of such entities are increasingly untenable. There are many stages where errors may be introduced and then propagated throughout the workflow, for example, manually transferring data from one application to another, converting files where appropriate, and managing and understanding diverse software environments, such as desktop tools and web browsers. Another issue is the growing quantity of data in public repositories; the results of an in silico experiment may change when executed at a later date, and so repeats are necessary to account for any novel data which may affect the outcome.

This thesis presents the development of a novel software system for the computational construction and analysis of biological network models, to overcome the above issues. The development of a supporting framework is a key contribution of this work, as it leads to the recommendation that tasks are implemented as web services, rather than as part of the more traditional software architectures, for example standalone and client-server (web-based). By proposing the development of a set of web services, the framework promotes reuse, composability and extensibility. Web services are tasks which may be executed remotely, via heterogeneous languages and platforms. They are designed for machine-to-machine interaction, and are described using a standard interface description, which is used to locate and invoke them. As such they are particularly well-suited to use in computational workflows.

Interaction and reaction data are currently available from a number of repositories, and in a variety of formats, and may be used to construct networks representing the whole ‘interactome’ or a particular region of interest. The network approach seeks to establish both the importance of single nodes or subsets of nodes within a holistic
network, as well as to characterise the global structure. The approach taken in this work is to examine relevant datasets at the level of network topology. This approach does not take into account non-linear reaction dynamics or cellular compartments, as these data are not complete for large-scale holistic models.

The workflows developed for this work are examples of how systematic queries can be made over tools and repositories available on heterogeneous systems. As there are ongoing issues of data quality and false-positives resulting from high-throughput experiments, such as yeast two-hybrid to determine PPIs, the results of workflows are used to suggest further refinements to an in silico experiment, or as a starting hypothesis which is testable in a laboratory, rather than a final result. The framework categories introduced have enabled the development of workflows in which services are replaceable with others from the same category, depending on the biological question being posed. For example, the workflow to investigate the global topological properties of a holistic network may be modified by inserting a data retrieval step, if a web service interface exists for some network data which the user wishes to analyse. Depending on the format of this data retrieved, a different transformation step could be applied. However the subsequent analysis and output formatting stages can remain the same.

Similarly, in the workflow to annotate a metabolic pathway with PPIs, the sequence of framework categories applied can remain the same, with modifications made depending on the metabolic pathway under study. For yeast, the IntAct repository was deemed a suitable source of PPIs, but for Arabidopsis a more specialised database such as AtPID may be substituted in the appropriate place, for a greater number of meaningful results. Identification of source and sink metabolites also calls for little change to the workflow presented in the previous chapter. If the model is available in a repository accessible through a web service, then this may be used instead of the BioModels operations “getModelSBMLById”.

Identification of cycles was an important task to implement for this work. It was created as a data analysis task which iteratively removes leaf nodes (i.e. those nodes in a network with only one incident edge) until none remain, leaving behind just the cycles in the network. The example given in Chapter 6 uses a known cycle in cholesterol metabolism to demonstrate how this method can leave behind a physiologically important and biologically relevant cycle. However the same method applied to a holistic network leaves behind a slightly smaller, but still very complex network. As an example, consider the Palsson human metabolic network used throughout this work. There are a total of 6931 nodes and 19195 edges. The cyclic core contains 5059 nodes and 15980 edges, 73% and 83% of the total nodes and edges respectively. These percent-
ages are astonishingly high, and imply that cycles heavily dominate the topological structure of such a network. There are however two important caveats regarding this analysis. The first is that cycles may be left behind which are only true cycles in an undirected network, but in a directed network such as one representing metabolism, the combination of nodes and directed edges result in a ‘false’ cycle, an example of which is given in Figure 7.1. The second is that reduction of a holistic metabolic network to its ‘cyclic core’ does not necessarily mean that every cycle contained within is of biological significance. Visualisation also becomes an issue, as the layout diagram of the cyclic core is as densely packed and difficult to interpret as the layout of the entire network. A direct comparison of the original Palsson network and the cyclic core is available in Appendix E.

Figure 7.1: A potential ‘cycle’ left behind in the cyclic core of a metabolic network. The undirected version of this part of the network is a true graph-theoretic cycle, however the directed version is not.

## 7.1 Limitations

The limitations on this work may be divided into two categories: those caused by the specific computational approaches taken, and those caused by the systems approach to the mechanisms of cellular interactions (see Table 7.1).

<table>
<thead>
<tr>
<th>Limitations due to computational approach</th>
<th>Limitations due to systems approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services and workflows may not be easily accessible to non-expert users</td>
<td>Data may be incomplete and/or biased</td>
</tr>
</tbody>
</table>
7.1 Limitations

<table>
<thead>
<tr>
<th>Limitations due to computational approach (cont.)</th>
<th>Limitations due to systems approach (cont.)</th>
</tr>
</thead>
</table>
| Execution of web services may incur large communi-
  cation overheads | Lack of significant amount of spatial and tem-
  poral data |
| Eventually the workflow may require manual inter-
  vention | |

Table 7.1: Limitations on this work

As web services and workflows are novel technologies they are still more likely to be used by expert bioinformaticians, who also develop them. Initiatives such as myExperiment enable pre-constructed workflows to be downloaded by interested researchers, but if the techniques are to become more widely adopted, they should be marketed effectively. In practice most research groups include individuals aware of specific technologies, and who are used as a resource by other group members. These ‘experts’ are therefore more likely to construct workflows, to be executed by others. As previously mentioned, a lot of bioinformatics research comprises repeated steps which scientists already carry out, so their automation and subsequent capturing as provenance are merits which should not be downplayed. With regard to the research carried out during the course of this thesis, the existence of tools such as Cytoscape and others implies that holistic network analysis is a relevant biological approach, and so the web services presented here may be a suitable alternative to those already developed as part of more traditional software architectures. A more detailed discussion on the relative merits of workflows and standalone tools may be found in Gollapudi et al. (2009).

A major issue with web services is that sending large files such as those containing interaction and reaction datasets over a network, may incur large computational overheads. While accessing remote computing power is an advantage from the perspective of the user, actually sending the data to and from services increases network traffic and may be detrimental to the performance of the web service, or workflow it is part of. One solution to this is the method implemented in Soaplab, a “pass-by-reference” approach rather than “pass-by-value”, discussed in Chapter 4. Another is to send data to a web service as an attachment to a SOAP message. The W3C recommendation for Message Transmission Optimisation Mechanism (MTOM, Gudgin et al. (2005)) is a method to send binary data to and from web services more efficiently. A web service which is MTOM-aware converts binary data to MIME data using an XML-binary Optimisation Package (XOP). The binary form is much smaller than the XML equivalent.
From a biological perspective, analysis of holistic intra-cellular networks carries two limitations: firstly that data are biased and incomplete, and secondly that these data commonly lack spatial and temporal annotations, i.e. where exactly a particular interaction takes place, and over what period of time. With regard to the first point, bias influences datasets, for example “popular” proteins are investigated more thoroughly, and so more interactions may be reported for these. The lack of spatial and temporal data results in an interaction dataset which is not biologically accurate: the clique detected in the first example in Chapter 5 (holistic network analysis of the human PPI network in MINT) did in fact consist of proteins which all bind each other at the same time, i.e. it is a functional module. However, other topological cliques may be reported, but contain proteins binding at different times. Spatial information is however more common, and is reported in a number of datasets.

As the examples given in Chapter 5 have demonstrated, in silico experiments carried out on data which may not always be of the highest quality and accuracy eventually require some manual curation once results are returned. The output formatting stage of a workflow aims to present results in a readable fashion. In some cases this is sufficient to draw biologically meaningful conclusions, for example in the workflow to identify source and sink nodes in a metabolic network. Here, the list of sources and sinks are formatted with meaningful names, which are then examined to detect interesting members of both lists. However in the final example, the workflow to annotate metabolic pathways with PPIs, the list of results is transformed into a layout diagram using Cytoscape, to take advantage of the many layout options provided by Cytoscape’s VizMapper tool, in order to present a biologically interesting final result. This particular stage was carried out under the supervision of an experienced biochemist, who was able to pick out the potentially significant PPIs and discard the rest, leading to a diagram which suggests an exciting new perspective on regulation of yeast glycolysis.

### 7.2 Future Work

The following future developments may be carried out to build on this work:

#### 7.2.1 Distributed workflows

To improve the efficiency of web service and workflow execution, a cluster of machines running a workload management system such as Condor (Litzkow, 1987) could be employed. When a web service is called, an intermediate step is carried out in which
the input data is placed in a database. A file is generated to submit a job to the
cluster, so the machine which carries out the actual task accesses this database, and
sends the result back. Enabling the use of multiple machines in a cluster rather than
executing all services on a single machine is particularly useful in scenarios where large
data (such as holistic networks) is submitted to computationally intensive tasks (for
example graph-theoretic operations such as calculation of betweenness centralities).

7.2.2 Mixed networks

As integration methods improve and networks move towards being a more accurate
representation of the mechanics of cellular interactions, they may incorporate edges
which are both directional and undirectional. For example, extending the last example
in the Biological Observations chapter, the entire metabolism (directed reactions) could
be annotated with PPIs (undirected interactions). Currently the common graph format
does not explicitly capture directionality, and the onus is on the user to apply the
appropriate analytical steps (either directed or undirected) depending on their network
type. A combined network as described would therefore always be interpreted as a
directed network, with undirected edges reported in both ‘directions’ (i.e. if protein A
binds protein B, the common graph format would contain one line showing A → B, and
another showing B → A). An additional task would need to be implemented, which
produces these ‘mirror image’ edges, to enable appropriate analyses to be carried out.

7.2.3 Extension of the common graph format

The limitations of the common graph format have been discussed in detail in Chapter
3. It is a simplistic representation, as it records only the interacting components in
a network. A possible extension to the format would be the inclusion of metadata to
describe edges. For metabolic networks, this could include rate information and/or
compartmental data. Such changes to the format would require changes to be made to
other parts of the framework, for example new analytical tools which make use of the
extra information. This is closely related to the above discussion on mixed networks,
as the metadata in the common graph format is one way to keep track of the type of
edge, and ultimately will also be used when rendering meaningful output.
7.3 Summary

7.2.4 Increasing exposure

To really test the usefulness of the web services and workflows developed by the author, it is necessary to increase their exposure, through publication and effective dissemination. Target users could first be identified, for example, based on their use of similar software such as Cytoscape. It is important to establish how useful the services are to the wider research community, and in what context they are effective. At the present time, the Soaplab2 installation which contains all the web services developed for this project are hosted at the Multidisciplinary Centre for Integrative Biology at the University of Nottingham, where access is limited to the University of Nottingham computer network. Steps have been taken, however, to publish the services such that they are globally accessible, and it is expected that the services will be registered on the BioCatalogue website, where they will be exposed to a large community of life-science researchers. The workflows described in Chapter 6 may also then be uploaded to the myExperiment workflow repository, to be downloaded, executed and modified by other users.

7.3 Summary

The questions posed by the author, along with the observations documented have shown how the combination of web service and workflow technologies aid discovery in the field of network biology. As interaction and reaction repositories grow in size and complexity, the tools to access and manipulate them will correspondingly have to become more powerful, as demonstrated by the middleware developed for the research described in this thesis. It is the opinion of the author that the traditional monolithic tools developed by academic consortia will increasingly give way to a suite of distributed web services. Sophisticated workflow management tools such as Taverna enable life science researchers to discover and compose these services into complex queries over diverse tools and datasets. The services presented here have challenged the established software paradigms that have guided network analysis software to date, and have been developed within a framework which enables the unbiased querying of publicly available data. Of particular interest are the results of a workflow to annotate a metabolic pathway with protein-protein interactions, in order to reveal potentially novel regulatory mechanisms. These have produced a testable hypothesis generated solely through computational methods, which if proved correct sheds new light on the biology of yeast metabolism. Such insight is ultimately the biggest reward and vindicates the approach taken.
References


References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


Service: Use SOAP, Take a REST or Go with HTML. In *Eighth IEEE International 

essentiality from topology features in metabolic networks: a case study in yeast. 

PERI, S., NAVARRO, J.D., KRISTIANSEN, T.Z., AMANCHY, R., SURENDRANATH, V., 
MUTHUSAMY, B., GANDHI, T.K.B., CHANDRIKA, K.N., DESHPANDE, N., SURESH, 
S., RASHMI, B.P., SHANKER, K., PADMA, N., NIRANJAN, V., HARSHA, H.C., TAL-
REJA, N., VRUSHABENDRA, B.M., RAMYA, M.A., YATISH, A.J., JOY, M., SHIV- 
ASHANKAR, H.N., KAVITHA, M.P., MENEZES, M., CHOUTHURY, D.R., GHOSH, 
N., SARAVANA, R., CHANDRAN, S., MOHAN, S., JONNALAGADDA, C.K., PRASAD, 
tein reference database as a discovery resource for proteomics. *Nucleic Acids Re-

PETTIFER, S., THORNE, D., MCDERMOTT, P., ATTWOOD, T., BARAN, J., BRYNE, 
J.C., HUPPONEN, T., MOWBRAY, D. and VRIEND, G. (2009) An Active Registry for 

PILLAI, S., SILVENTOINEN, V., KALLIO, K., SENDER, M., SOBHANY, S., TATE, J., 
VELANKAR, S., GOLOVIN, A., HENRICK, K., RICE, P., STOEHR, P. and LOPEZ, 

requirements for N-terminal acetylation of eukaryotic proteins. *Journal of Molecular 

putational Biology*, **2**(12), e174.


SALWINSKI, L., MILLER, C.S., SMITH, A.J., PETTIT, F.K., BOWIE, J.U. and EIEN-
REFERENCES


REFERENCES


Appendix A

Taverna tutorial

The following tutorial is intended as an introduction to using the Taverna workbench, and covers the following topics: download, configuration, execution, and building and running a simple workflow. The instructions are for Unix style operating systems, though the workbench also runs under Windows and Mac OS.

A.1 Prerequisites

The following prerequisites are required in order for the workbench to run:

- Java5 (http://java.sun.com/j2se/1.5.0/)
- Graphviz (http://www.graphviz.org/) - Taverna requires the ‘dot’ executable to render workflow diagrams (this is included with the Windows version)

A.2 Download

The workbench may be downloaded from the project website, http://www.mygrid.org.uk/tools/taverna/taverna-1/taverna-download/. It is available as a ZIP file, taverna-workbench-1.7.1.zip.

Once downloaded, the ZIP file should be extracted to create the directory taverna-1.7.1.

A.3 Configuration

Within the taverna-1.7.1 directory, there is a directory conf, which contains a file mygrid.properties. The following configuration steps refer to this file.

Proxy configuration:

- mygrid.properties contains the following lines:
A.4 Execution

The Taverna workbench is run by executing \texttt{runme.sh}, which is in the \texttt{taverna-1.7.1} directory, from within a terminal (the example given here uses bash):

\begin{verbatim}
bash-3.2$ ./taverna-1.7.1/runme.sh
\end{verbatim}

Figure A.1 shows the workbench loaded and ready for workflow construction. The Available Processors pane contains those processors defined in the configuration file \texttt{mygrid.properties}, though new processors may be added as required.
A.5 Building and running a simple workflow

The example workflow demonstrated here makes use of the BIND web service, which can be used to retrieve data on PPIs in the Biomolecular Object Network Databank. The web service is defined using a WSDL file, http://soap.bind.ca/wsd1/bind.wsdl. The aim of the workflow is to retrieve all *A. thaliana* PPIs in Cytoscape SIF form. The output of the workflow may be saved locally and viewed in Cytoscape, or may be used as input to further web services and workflows.

1. Add a new WSDL scavenger by right-clicking on Available Processors and selecting “Add new WSDL Scavenger...”. This will bring up a dialog box. Type “http://soap.bind.ca/wsd1/bind.wsdl” here and click OK (Figure A.2).

2. The WSDL is added to the Available Processors list, and can be expanded by clicking on the small circle to the left of the label. The WSDL processors available are now visible (Figure A.3).

3. The “idSearch” processor is added to the workflow, by right-clicking on the processor name, and selecting “Add to model”. The processor now appears in the
A.5 Building and running a simple workflow

Figure A.2: Add the BIND WSDL to the list of Available Processors

Figure A.3: WSDL processor list expanded

Advanced Model Explorer (AME) pane, and also in the Workflow Diagram pane (Figure A.4).

4. The “idSearch” processor has three input ports, “id”, “idType” and “returnType”. Each input port receives data, from workflow inputs in the AME. To add these, right click on “Workflow inputs” and select “Create New Input”. In the dialog box that appears, type a meaningful input name (in this case “id” to match the name of the input port of the processor) and click OK (Figure A.5).

5. Repeat for the inputs “idType” and “returnType”. There should now be three workflow inputs. As well as appearing in the AME, they appear in the Workflow Diagram pane (Figure A.6).
6. Now data links must be added between the workflow inputs, and the input ports of the “idSearch” processor. To do this, right-click on the “id” workflow input. This brings up a drop-down menu with a subheading “Processors”. The “idSearch” processor is listed here. Clicking this reveals a further drop-down menu “Choose an Input” which lists all three input ports to the “idSearch” processor. To create a data link, select the “id” input port (Figure A.7).

---

Figure A.4: Add the “idSearch” processor in the AME

Figure A.5: Add workflow inputs
A.5 Building and running a simple workflow

7. Repeat the previous step for the workflow inputs “idType” and “returnType”. The three data links are now visible in the AME, and in the Workflow Diagram pane, connecting the workflow inputs to the input ports of the “idSearch” processor (Figure A.8).

8. Now the workflow outputs must be added. The “idSearch” processor has an output port called “searchResultBean”. This produces a complex data structure which can be interrogated using an XML splitter. To add the XML splitter, right click on the “searchResultBean” and select “Add XML splitter” from the menu that appears (Figure A.9).

9. This adds the “searchResultBeanXML” processor to the workflow, with input ports and output ports as before (Figure A.10).

10. For this workflow, we wish to access the interaction records. First of all a workflow output should be created. This is done in a very similar way to creating workflow inputs. Right-click on “Workflow outputs” and select “Create New Output”. In the dialog box that appears, type a name for this output. In this example, we will use “interactions” (Figure A.11).

11. At this stage, the workflow consists of three inputs, one output and two processors.
A.5 Building and running a simple workflow

Figure A.8: All data links added in the AME

Figure A.9: Add an XML splitter

The processors are different colours depending on their type. Inputs are grouped together and denoted by a red triangle, and outputs are grouped and denoted by a green triangle (Figure A.12).

12. The final step in constructing the workflow is to create a data link between the output port of the XML splitter processor and the workflow output “interactions”. To do this, right click on the desired output port “records”. This brings up a menu “Connect output "records" to...”. From here it is possible to access a list of workflow outputs, which in this case contains just one, “interactions”. Click on “interactions” to create the data link (Figure A.13).

13. The workflow is now complete and ready to be run (Figure A.14).
A.5 Building and running a simple workflow

![Workflow Diagram](image)

**Figure A.10:** “searchResultBeanXML” processor added to the AME

![Workflow Outputs](image)

**Figure A.11:** Add a workflow outputs

14. To run the workflow, click on the File menu and select “Run workflow...” (Figure A.15).

15. An input window will appear, with the three inputs listed together with a small version of the workflow diagram (Figure A.16).

16. To populate each input with a value, click on the input name (e.g. “id”) and then click the “New Input” button. Replace the text “Some input data goes here” with the desired input. For this example, enter “3702” which is the Arabidopsis taxonomic identifier (Figure A.17).

17. Repeat for the remaining two inputs, “idType” and “returnType”. For this example, enter “taxid” and “cytoscape” respectively. A complete list of possible
A.5 Building and running a simple workflow

18. The workflow enactor will appear, and lists processors and their statuses. During workflow execution, the processors in the workflow graph diagram change colour according to whether they are scheduled (grey), running (purple) or completed (green) (Figure A.19).

19. When the workflow has successfully completely, the Results window is displayed,
A.5 Building and running a simple workflow

which is divided into two halves. The left hand side lists result items, while the right hand side displays the selected result items. The result of running this workflow is a file in Cytoscape SIF format, containing all the *A. thaliana* PPIs in BIND. The “Save to disk” button can be clicked to save to a local disk (Figure A.20).
A.5 Building and running a simple workflow

20. The “Workflow metadata” tab in the AME has fields for workflow name, author and description (Figure A.21).

21. If desired, the workflow itself can be saved to a local disk, as a Scufl file, by selecting “Save workflow...” from the “File” menu. This file can then be reloaded if the workflow needs to be run multiple times, or with a different set of inputs (Figure A.22).

22. The same workflow can be used, for example, to retrieve all human PPIs (by changing the taxonomy identifier to “9606”), or all interactions from a particular
A.5 Building and running a simple workflow

Figure A.18: All workflow inputs entered

Figure A.19: Workflow enactor window showing workflow partially executed

publication (using the PubMed identifier) (Figure A.23).
A.5 Building and running a simple workflow

Figure A.20: Workflow outputs displayed in Results window

Figure A.21: Editing workflow metadata
A.5 Building and running a simple workflow

Figure A.22: Saving a workflow to disk

Figure A.23: Running the unaltered workflow with a different set of inputs: this workflow will retrieve all PPIs from the publication with PubMed identifier “9581554” and return them in the PSI-MI 2.5 format
Appendix B

Technology Evaluation examples

B.1 Installing Perl modules under Linux without root access

Perl modules may be installed and accessed under the home directory. The following steps illustrate this process:

1. Decide where to install the modules. Two directories are required under the home directory: /lib and /bin. They will both be created automatically when the first module is installed.

2. Download the module required and unpack it.

3. cd into the newly-created directory

4. Issue the following set of commands:

   • % perl Makefile.PL PREFIX=/home/<username> \\
     INSTALLPRIVLIB=/home/<username>/lib/perl5 \\
     INSTALLSCRIPT=/home/<username>/bin \\
     INSTALLSITELIB=/home/<username>/lib/perl5/site_perl \\
     INSTALLBIN=/home/<username>/bin \\
     INSTALLMAN1DIR=/home/<username>/lib/perl5/man \\
     INSTALLMAN3DIR=/home/<username>/lib/perl5/man3

   • % make
   • % make test
   • % make install

5. To allow Perl scripts access to locally installed modules, the lib module may be used. Insert the following at the start of the Perl script:

   • use lib qw{/home/<username>/lib/perl5/5.00503/ 
     /home/<username>/lib/perl5/site_perl/5.005};
B.2 SOAP::Lite: Calculator service

Listing B.1: Calculator.pm (request handler)

```perl
#!/usr/bin/perl
package Calculator;
sub add {
    my ($class, $int1, $int2) = @_;
    my $sum = $int1 + $int2;
    return "\$sum\n";
}
sub subtract {
    my ($class, $int1, $int2) = @_;  
    my $diff = $int1 - $int2;
    return "$\$diff\n";
}
1;
```

Listing B.1 shows the web service operations add and subtract implemented as subroutines in a Perl module, Calculator.pm. This is the request handler. Every call to a web service method passed from a client first contains the package name, followed by any parameters passed in the method call. In the example this corresponds to ($class, $int1, $int2). The dispatcher code for the Calculator service is shown in Listing B.2.

Listing B.2: calculator.cgi (dispatcher)

```perl
#!/usr/bin/perl
use SOAP::Transport::HTTP;
use lib '/var/www/cgi-bin/sirisha';
SOAP::Transport::HTTP::CGI
    ->dispatch_to('Calculator')
    ->handle;
```

B.3 SOAP::Lite: Calculator service, modified to include POD

The modified Perl module for the Calculator service is shown in Listing B.3.
B.4 SOAP::Lite: WSDL generated by Pod::WSDL

The WSDL is generated using the script in Listing B.4.

```perl
#!/usr/bin/perl
use Pod::WSDL;
use lib '/services/www/cgi/sirisha';
open (WSDL, ">calc.wsdl") || die "could not open wsd file!

Listing B.3: Calculator.pm (modified to include POD)

B.4 SOAP::Lite: WSDL generated by Pod::WSDL
B.5 Apache Axis: Calculator service (.jws example)

Listing B.4 contains source code for a .jws version of the Calculator service.

```java
my $pod = new Pod::WSDL(source => 'Calculator',
    location => 'http://behemoth.mycib.ac.uk/cgi/sirisha/calculator.cgi',
    pretty => 1,
    withDocumentation => 1);

print WSDL $pod->WSDL;
```

Listing B.4: create-wsdl.pl

```java
public class Calculator {
    public int add(int i1, int i2) {
        return i1 + i2;
    }

    public int subtract(int i1, int i2) {
        return i1 - i2;
    }
}
```

Listing B.5: Calculator.jws

B.5 Apache Axis: Calculator service (.jws example)

Listing B.5 contains source code for a .jws version of the Calculator service.

```java
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;
import javax.xml.rpc.ParameterMode;
```

B.6 Apache Axis: Calculator client

Listing B.6 shows an Apache Axis client to consume the Calculator service. An example execution of this client:

```
java -cp .:$AXISCLASSPATH samples.userguide.example2.CalcClient -p8080 add 5 3
```

which prints out:

Got result : 8

```java
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;
import javax.xml.rpc.ParameterMode;
```
B.7 BioMoby: Service registration

The Perl implementation provides the API for communicating with MOBY Central to register the above. An example registration script for the simpleCalculatorAdd service is shown in Listing B.7.

```perl
#!/usr/bin/perl
use strict;
use MOBY::Client::Central;

# Instantiate a client to mobycentral
my $m = MOBY::Client::Central->new( Registries => { mobycentral => { URL => "http://moby.ucalgary.ca/moby/MOBY-Central.pl", URI => "http://moby.ucalgary.ca/MOBY/Central", }, },

# Service name
my $serviceName = "simpleCalculatorAdd";

# Service type
my $serviceType = "TutorialService";
```

Listing B.7: CalcClient.java
# URI of service provider
my $authURI = "mycib.ac.uk";

# E-mail address of service provider
my $email = 'sirisha@mycib.ac.uk';

# URL to service CGI file
my $URL = "http://localhost/cgi-bin/simpleCalculatorAdd.cgi";

# Small service description
my $description = "This service consumes two numbers and returns their sum";

# Service inputs
my @input_namespaces1 = ();
my @input_namespaces2 = ();

my @input_simples1 = ('String', \@input_namespaces1);
my @input_simples2 = ('String', \@input_namespaces2);

my @input_articles1 = ('int1', \@input_simples1);
my @input_articles2 = ('int2', \@input_simples2);

my @all_inputs = (\@input_articles1, \@input_articles2);

# Service outputs
my @output_namespaces = ();
my @output_simples = ('String', \@output_namespaces);
my @output_articles = ('sum', \@output_simples);
my @all_outputs = (\@output_articles);

# Service registration
my $REG = $m->registerService(
    serviceName => $serviceName,
    serviceType => $serviceType,  
    authURI => $authURI,  
    contactEmail => $email,  
    description => $description,  
    category => "cgi",  
    URL => $URL,  
    input => \@all_inputs,  
    output => \@all_outputs,  
    secondary => undef,  
);

# Display success or failure of registration
$REG->success?print "Success!\n":print "Failure": $REG->message,"\n";

Listing B.7: moby-service-registration.pl
B.8 BioMoby: Add service request handler

Listing B.8 contains the application logic of the BioMoby web service. It is automatically generated using MOSES-MOBY, but application logic itself must be entered manually by the service provider.

```perl
package Service::simpleCalculatorAdd;

use FindBin qw( $Bin );
use lib $Bin;

use MOSES::MOBY::Base;

# Dynamically load
BEGIN {
    use MOSES::MOBY::Generators::GenServices;
    new MOSES::MOBY::Generators::GenServices->load
        ( authority => 'mycib.ac.uk',
          service_names => ['simpleCalculatorAdd']);
}

use vars qw( @ISA );
@ISA = qw( uk::ac::mycib::simpleCalculatorAddBase );
use MOSES::MOBY::Package;
use MOSES::MOBY::ServiceException;
use strict;

use MOSES::MOBY::Data::String;

my %valid_namespaces = ( );

# The process_it method is called for every job in the client request
sub process_it {
    my ($self, $request, $response, $context) = @_;

    # Perform optional namespace checking for inputs to this service
    my $namespace1 = eval { $request->int1->namespace };  
    my $namespace2 = eval { $request->int2->namespace };  

    # Read input data (eval to protect against missing data)
    my $int1 = eval { $request->int1 };  
    my $int2 = eval { $request->int2 };  

    # Application logic
    my $number1 = $int1->value if defined $int1;
    my $number2 = $int2->value if defined $int2;
    my $total = $number1 + $number2;

    # Response
```
Listing B.9 contains the code which dispatches requests to the appropriate service module (i.e. BioMoby service), and is automatically generated using MOSES-MOBY.

```perl
use strict;
use CGI;
use CGI::Carp qw(fatalsToBrowser);

# Maximum size of POST field
$CGI::POST_MAX=1024 * 1024 * 10;

use lib '/home/sirisha/Perl-MoSeS/';
use lib '/home/sirisha/Perl-MoSeS/generated/';
use lib '/home/sirisha/Perl-MoSeS/services/';

# Require service module and add it to ISA hierarchy
use base 'Service::simpleCalculatorAdd';

# Get the CGI variable
my $q = new CGI;

# Get the data from the 'data' parameter
my $data = $q->param('data') || $q->param('POSTDATA') || "";

# Call the service
my $x = __PACKAGE__->simpleCalculatorAdd($data);

# Print the results
print $q->header(-type=>'text/xml');
print $x;

# Override the method in Service::ServiceBase
```

B.9 BioMoby: Add service dispatcher

Listing B.9 contains the code which dispatches requests to the appropriate service module (i.e. BioMoby service), and is automatically generated using MOSES-MOBY.
B.10 Soaplab1: Web service and ACD definition

Figure B.1 shows the ACD description and corresponding script for the add service. A detailed summary of the ACD specification is given on the EMBOSS website\(^1\). The Calculator example can be used to highlight important aspects of ACD file creation. The ‘appl’ token specifies the name of the executable to be used. The ‘integer’ and ‘outfile’ tokens are among the many data types supported in ACD for input and output. ‘comment’ is a special attribute provided specifically for the Soaplab project, and is used to specify command-line options, together with the keyword ‘qualifier’ which is used to denote which flags are used to pass arguments to the script.

\(^1\)http://emboss.sourceforge.net/developers/acd/

```
#!/usr/bin/perl -w
use strict;
use warnings;
use Getopt::Std;
my %opts = ();
getopts('i:j:', \%opts);
my $number1 = $opts{\$i};
my $number2 = $opts{\$j};
my $sum = $number1 + $number2;
print $sum . "\n";
```

```
sub finish_output {
    my ($self, $out_package) = @_;
    return $out_package->toXMLDocument->toString (1);
}
```

Listing B.9: simpleCalculatorAdd.cgi

Figure B.1: The add Perl script on the left, together with the corresponding ACD file used to describe the command line of the script, add.acd
B.11 Soaplab2: Configuration steps prior to building

After downloading and extracting the Soaplab2 distribution, it is necessary to first locate a file `build.properties.template`. This should be copied and renamed to `build.properties`, and edited accordingly. There are a number of build-time properties in this file, however of particular importance are the following:

```plaintext
# Pointer to your own templates of Soaplab configuration files
my.soaplab.properties = testing.soaplab.properties
#my.soaplab.client.properties = testing.soaplab.client.properties
#my.log4j.properties = testing.log4j.properties

# Full path to your Tomcat installation.
tomcat.home = /usr/share/tomcat5

tomcat.port = 8080
tomcat.host = compute1.mycib.ac.uk
```

`tomcat.home`, `tomcat.port` and `tomcat.host` indicate the location of the Tomcat installation. `my.soaplab.properties` points to a file called, by default, `testing.soaplab.properties`. This file contains Soaplab2 properties, which are given values based on the local Soaplab2 installation. An example `testing.soaplab.properties` is given here:
B.11 Soaplab2: Configuration steps prior to building

base.dir = /home/sirisha/Desktop/soaplab2

metadata.dir = /home/sirisha/Desktop/soaplab2/metadata/generated
applist = ${metadata.dir}/OtherApplications.xml
applist = ${metadata.dir}/GowlabApplications.xml
applist = ${metadata.dir}/EBIApplications.xml

runtime.dir = /home/sirisha/Desktop/soaplab2/_R_
working.dir = ${runtime.dir}/SANDBOX
results.dir = ${runtime.dir}/RESULTS
scripts.dir = @SCRIPTS_DIR@
addtopath.dir = ${scripts.dir}

#jobs.cleaning.interval = 60000
#jobs.timeout = 864000000
#services.cleaning.interval = 300000

###classic.helloworld.metadata.file = a/b/c

#ignore.heartbeat.events = false

#accept.any.exitcode

#synonym.sowa = org.soaplab.sowa.SowaJobFactory

# Accessing results by URLs:
# -------------------------
tomcat.host = compute1.mycib.ac.uk
tomcat.port = 8080
results.url = http://${tomcat.host}:${tomcat.port}/soaplab2/results
results.url.target.dir = /usr/share/tomcat5/webapps/soaplab2/results
#results.url.ignore
Appendix C

Tutorial examples for web services developed

This Appendix contains further documentation for each Soaplab2 web service described in Chapter 5: inputs and outputs, and a tutorial example.

N.B. For Soaplab2 services there are two special outputs, report and detailed status which are generated automatically for every service, and are therefore not explicitly defined in the ACD file. This section of the documentation therefore only gives the name and description of outputs of the web service, which are specified by the author in the ACD file.

C.1 Group: analyse_directed

Three toy networks A, B and C (Figure C.1) are used to demonstrate the usage of web services in this category.

C.1.1 add_edges_directed

C.1.1.1 Expected inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_direct_data</td>
<td>Network represented in the common graph format</td>
</tr>
<tr>
<td>network_url</td>
<td>URL pointing to network_direct_data</td>
</tr>
<tr>
<td>edgelist</td>
<td>A list of edges represented in the common graph format</td>
</tr>
<tr>
<td>edgelist_url</td>
<td>URL pointing to edgelist</td>
</tr>
</tbody>
</table>

C.1.1.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_network</td>
<td>Network with specified edges added</td>
</tr>
<tr>
<td>new_network_url</td>
<td>URL pointing to new_network</td>
</tr>
</tbody>
</table>
Figure C.1: Three directed networks. Network A is strongly connected, that is, there exists a path between every pair of nodes in the network. Networks B and C are not strongly connected.

C.1.1.3 Example

C.1.1.3.1 Input name: network

Network A.

C.1.1.3.2 Input name: edgelist

c a

C.1.1.3.3 Output name: new_network

a b
b a
c a
c d
b c
e d
C.1 Group: analyse directed

C.1.2 get_network_diameter_directed

C.1.2.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.1.2.2 Outputs

- diameter: The diameter of the network
- diameter_url: URL pointing to diameter

C.1.2.3 Example

C.1.2.3.1 Input name: network

Network A.

C.1.2.3.2 Output name: diameter

4

C.1.3 get_network_radius_directed

C.1.3.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.1.3.2 Outputs

- radius: The radius of the network
- radius_url: URL pointing to radius

C.1.3.3 Example

C.1.3.3.1 Input name: network

Network A.
C.1.3.3.2 Output name: radius

C.1.4 get_sink_nodes_directed
C.1.4.1 Expected inputs
- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`

C.1.4.2 Outputs
- `sinks`: List of sink nodes
- `sinks_url`: URL pointing to `sinks`

C.1.4.3 Example
C.1.4.3.1 Input name: network
Network B.

C.1.4.3.2 Output name: sinks

```
a
e
f
```

C.1.5 get_source_nodes_directed
C.1.5.1 Expected inputs
- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`

C.1.5.2 Outputs
- `sources`: List of source nodes
- `sources_url`: URL pointing to `sources`

C.1.5.3 Example
C.1.5.3.1 Input name: network
Network B.
C.1.5.3.2 Output name: sources

```
b
g```

C.1.6 get_subgraph_directed

C.1.6.1 Expected inputs

- network\_direct\_data: Network represented in the common graph format
- network\_url: URL pointing to network\_direct\_data
- nodelist\_direct\_data: List of nodes to appear in the subgraph
- nodelist\_url: URL pointing to nodelist\_direct\_data

C.1.6.2 Outputs

- subgraph: Subgraph represented in the common graph format
- subgraph\_url: URL pointing to subgraph
- missing\_nodes: A list of any nodes which do not appear in the subgraph
- missing\_nodes\_url: URL pointing to missing\_nodes

C.1.6.3 Example

C.1.6.3.1 Input name: network

Network A.

C.1.6.3.2 Input name: nodelist

```
a
b
c
e```

C.1.6.3.3 Output name: subgraph

```
a   b
b   a
b   c```
C.1 Group: analyse\_directed

C.1.6.3.4 Output name: missing

C.1.7 largest\_strongly\_connected\_component\_directed

C.1.7.1 Expected inputs

- network\_direct\_data: Network represented in the common graph format
- network\_url: URL pointing to network\_direct\_data

C.1.7.2 Outputs

- largest\_strong\_component: List of nodes in the largest strongly connected component
- largest\_strong\_component\_url: URL pointing to largest\_strong\_component

C.1.7.3 Example

C.1.7.3.1 Input name: network

Network C.

C.1.7.3.2 Output name: largest\_strong\_component

C.1.8 list\_strongly\_connected\_components\_directed

C.1.8.1 Expected inputs

- network\_direct\_data: Network represented in the common graph format
- network\_url: URL pointing to network\_direct\_data

C.1.8.2 Outputs

- list\_strong\_components: List of nodes in each strongly connected component of the network, separated by a newline character
- list\_strong\_components\_url: URL pointing to list\_strong\_components

C.1.8.3 Example

C.1.8.3.1 Input name: network

Network C.
C.1.8.3.2 Output name: list_strong_components

C.1.9 node_degree_directed

C.1.9.1 Expected inputs

- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`
- `node`: Query node of interest

C.1.9.2 Outputs

- `degree`: Query node and its degree
- `degree_url`: URL pointing to degree

C.1.9.3 Example

C.1.9.3.1 Input name: network

Network A.

C.1.9.3.2 Input name: node

b

C.1.9.3.3 Output name: degree

b 4
C.1.10 node_in_degree_directed

C.1.10.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
- node: Query node of interest

C.1.10.2 Outputs

- in_degree: Query node and its in-degree
- in_degree_url: URL pointing to in_degree

C.1.10.3 Example

C.1.10.3.1 Input name: network

Network $B$.

C.1.10.3.2 Input name: node

```
\texttt{d}
```

C.1.10.3.3 Output name: in_degree

```
\texttt{d} \quad 2
```

C.1.11 node_out_degree_directed

C.1.11.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
- node: Query node of interest

C.1.11.2 Outputs

- out_degree: Query node and its out-degree
- out_degree_url: URL pointing to out_degree

C.1.11.3 Example

C.1.11.3.1 Input name: network

Network $B$.  

184
C.1 Group: analyse_directed

C.1.11.3.2 Input name: node

\[
g
\]

C.1.11.3.3 Output name: out_degree

\[
g \ 1
\]

C.1.12 query_adjacency_matrix_directed

C.1.12.1 Expected inputs

- network_direct_data Network represented in the common graph format
- network_url URL pointing to network_direct_data
- node Query node of interest

C.1.12.2 Outputs

- adjacency Query node, followed by tab-delimited list of nodes adjacent to it
- adjacency_url URL pointing to adjacency

C.1.12.3 Example

C.1.12.3.1 Input name: network

Network \( B \).

C.1.12.3.2 Input name: node

\[
d
\]

C.1.12.3.3 Output name: adjacency

\[
d \ f \ e
\]

C.1.13 rank_betweenness_directed

C.1.13.1 Expected inputs

- network_direct_data Network represented in the common graph format
- network_url URL pointing to network_direct_data
C.1.13.2 Outputs

betweenness_ranking List of nodes and associated betweenness centralities, ranked from highest to lowest
betweenness_ranking_url URL pointing to betweenness_ranking

C.1.13.3 Example

C.1.13.3.1 Input name: network

Network A.

C.1.13.3.2 Output name: betweenness_ranking

<table>
<thead>
<tr>
<th>Node</th>
<th>Betweenness Centrality</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0.6666666666667</td>
</tr>
<tr>
<td>d</td>
<td>0.6666666666667</td>
</tr>
<tr>
<td>c</td>
<td>0.333333333333</td>
</tr>
<tr>
<td>a</td>
<td>0.0</td>
</tr>
<tr>
<td>e</td>
<td>0.0</td>
</tr>
</tbody>
</table>

C.1.14 rank_closeness_directed

C.1.14.1 Expected inputs

network_direct_data Network represented in the common graph format
network_url URL pointing to network_direct_data

C.1.14.2 Outputs

closeness_ranking List of nodes and associated closeness centralities, ranked from highest to lowest
closeness_ranking_url URL pointing to closeness_ranking

C.1.14.3 Example

C.1.14.3.1 Input name: network

Network A.

C.1.14.3.2 Output name: closeness_ranking

<table>
<thead>
<tr>
<th>Node</th>
<th>Closeness Centrality</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>0.6666666666667</td>
</tr>
<tr>
<td>b</td>
<td>0.571428571429</td>
</tr>
<tr>
<td>c</td>
<td>0.5</td>
</tr>
<tr>
<td>e</td>
<td>0.444444444444</td>
</tr>
<tr>
<td>a</td>
<td>0.4</td>
</tr>
</tbody>
</table>
C.1.15  rank_degrees_directed
C.1.15.1  Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.1.15.2  Outputs

- degree_ranking: List of nodes and associated degrees, ranked from highest to lowest
- degree_ranking_url: URL pointing to degree_ranking

C.1.15.3  Example

C.1.15.3.1  Input name: network

Network A.

C.1.15.3.2  Output name: degree_ranking

<table>
<thead>
<tr>
<th>Node</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
</tr>
</tbody>
</table>

C.1.16  rank_secondary_degrees_directed
C.1.16.1  Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.1.16.2  Outputs

- secondary_degree_ranking: List of nodes and associated degrees, ranked from highest to lowest
- secondary_degree_ranking_url: URL pointing to secondary_degree_ranking

C.1.16.3  Example

C.1.16.3.1  Input name: network

Network A.
C.1.16.3.2 Output name: secondary_degree_ranking

\begin{verbatim}
c  2
d  2
a  1
b  1
e  1
\end{verbatim}

C.1.17 remove_edges_directed

C.1.17.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
- edgelist: A list of edges represented in the common graph format
- edgelist_url: URL pointing to edgelist

C.1.17.2 Outputs

- new_network: Network with specified edges removed
- new_network_url: URL pointing to new_network

C.1.17.3 Example

C.1.17.3.1 Input name: network

Network A.

C.1.17.3.2 Input name: edgelist

\begin{verbatim}
a  b
d  e
c  d
b  c
\end{verbatim}

C.1.17.3.3 Output name: new_network

\begin{verbatim}
b  a
e  d
d  b
c
\end{verbatim}
C.1.18 shortest_path_directed

C.1.18.1 Expected inputs

network_direct_data  Network represented in the common graph format
network_url        URL pointing to network_direct_data
source_node        First node in path
dest_node          Final node in path

C.1.18.2 Outputs

shortest_path       List of nodes along the shortest path between first node and final node
shortest_path_url   URL pointing to shortest_path

C.1.18.3 Example

C.1.18.3.1 Input name: network

Network B.

C.1.18.3.2 Input name: source_node

\[
g\]

C.1.18.3.3 Input name: dest_node

\[
e\]

C.1.18.3.4 Output name: shortest_path

\[
g \quad c \quad d \quad e\]

C.1.19 shortest_path_length_directed

C.1.19.1 Expected inputs

network_direct_data  Network represented in the common graph format
network_url        URL pointing to network_direct_data
source_node        First node in path
dest_node          Final node in path
C.1.19.2 Outputs

- shortest_path_length: Shortest path length between first node and final node
- shortest_path_length_url: URL pointing to shortest_path_length

C.1.19.3 Example

C.1.19.3.1 Input name: network

Network $B$.

C.1.19.3.2 Input name: source_node

$g$

C.1.19.3.3 Input name: dest_node

$e$

C.1.19.3.4 Output name: shortest_path

3

C.1.20 size_distribution_strongly_connected_components_directed

C.1.20.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.1.20.2 Outputs

- distribution: Size distribution of strongly connected components
- distribution_url: URL pointing to distribution

C.1.20.3 Example

C.1.20.3.1 Input name: network

Network $C$.  

190
C.2 Group: analyse_misc

C.2.1 compare_two_rankings

C.2.1.1 Expected inputs

- `number_to_compare`: String giving the number of nodes (x) to compare from each list; the top x nodes are compared
- `ranking1_direct_data`: The first list of ranked nodes
- `ranking1_url`: A URL pointing to `ranking1_direct_data`
- `ranking2_direct_data`: The second list of ranked nodes
- `ranking2_url`: A URL pointing to `ranking2_direct_data`
C.2 Group: analyse_misc

C.2.1.2 Outputs

- both_list
  - List of nodes which appear in both rankings
- both_url
  - A URL pointing to both
- ranking1_only
  - List of nodes which only appear in the first ranking
- ranking1_only_url
  - A URL pointing to ranking1_only
- ranking2_only
  - List of nodes which only appear in the second ranking
- ranking2_only_url
  - A URL pointing to ranking2_only

C.2.1.3 Example

C.2.1.3.1 Input name: ranking1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>0.010</td>
</tr>
<tr>
<td>b</td>
<td>0.008</td>
</tr>
<tr>
<td>e</td>
<td>0.005</td>
</tr>
<tr>
<td>f</td>
<td>0.004</td>
</tr>
<tr>
<td>a</td>
<td>0.000</td>
</tr>
<tr>
<td>c</td>
<td>0.000</td>
</tr>
</tbody>
</table>

C.2.1.3.2 Input name: ranking2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>15</td>
</tr>
<tr>
<td>b</td>
<td>12</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
</tr>
</tbody>
</table>

C.2.1.3.3 Input name: number_to_compare

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

C.2.1.3.4 Output name: ranking1_only

<table>
<thead>
<tr>
<th>d</th>
</tr>
</thead>
</table>

C.2.1.3.5 Output name: ranking2_only
C.3 Group: analyse_undirected

C.2.1.3.6 Output name: both

C.2.2 reverse_adjacency_list

C.2.2.1 Expected inputs

- adjacency_direct_data: Adjacency list representation of interactions
- adjacency_url: URL pointing to adjacency_direct_data

C.2.2.2 Outputs

- reversed: Reversed adjacency list representation of interactions
- reversed_url: URL pointing to reversed

C.2.2.3 Example

C.2.2.3.1 Input name: adjacency

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>b</td>
<td></td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td></td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

C.2.2.3.2 Output name: reversed

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td></td>
<td>a</td>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C.3 Group: analyse_undirected

Two toy networks $D$ and $E$ (Figure C.2) are used to demonstrate the usage of web services in this category.
C.3 Group: analyse_undirected

C.3.1 add_edges_undirected

C.3.1.1 Expected inputs

<table>
<thead>
<tr>
<th>network_direct_data</th>
<th>Network represented in the common graph format</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_url</td>
<td>URL pointing to network_direct_data</td>
</tr>
<tr>
<td>edgelist</td>
<td>A list of edges represented in the common graph format</td>
</tr>
<tr>
<td>edgelist_url</td>
<td>URL pointing to edgelist</td>
</tr>
</tbody>
</table>

C.3.1.2 Outputs

<table>
<thead>
<tr>
<th>new_network</th>
<th>Network with specified edges removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_network_url</td>
<td>URL pointing to new_network</td>
</tr>
</tbody>
</table>

C.3.1.3 Example

C.3.1.3.1 Input name: network

Network $D$. 

---

Figure C.2: Two undirected networks. Network $D$ comprises one connected component. Network $E$ comprises four connected components, including a singleton node, $j$. The singleton is connected to itself by a self loop.
C.3 Group: analyse_undirected

C.3.1.3.2 Input name: edgelist

\[
\begin{array}{ll}
  a & g \\
\end{array}
\]

C.3.1.3.3 Output name: new_network

\[
\begin{array}{llllll}
  a & b \\
  b & c \\
  c & g \\
  b & d \\
  c & d \\
  d & f \\
  a & g \\
  d & e \\
  b & f \\
\end{array}
\]

C.3.2 cliques_containing_node_undirected

C.3.2.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
- node: A query node of interest

C.3.2.2 Outputs

- cliques_containing_node: List of cliques of nodes, which contain the node of interest
- cliques_containing_node_url: URL pointing to cliques_containing_node

C.3.2.3 Example

C.3.2.3.1 Input name: network

Network $D$.

C.3.2.3.2 Input name: node

\[
d
\]
C.3 Group: analyse_undirected

C.3.2.3.3 Output name: cliques_containing_node

\[
\begin{align*}
&b \\
&d \\
&c \\
&b \\
&d \\
&f \\
&e \\
&d
\end{align*}
\]

C.3.3 find_cliques_undirected

C.3.3.1 Expected inputs

- **network_direct_data** Network represented in the common graph format
- **network_url** URL pointing to network_direct_data

C.3.3.2 Outputs

- **cliques** List of cliques, each of which is a list of the nodes contained within that clique
- **cliques_url** URL pointing to cliques

C.3.3.3 Example

C.3.3.3.1 Input name: network

Network \( D \).

C.3.3.3.2 Output name: cliques

\[
\begin{align*}
&b \\
&d \\
&c \\
&b \\
&d \\
&f \\
&b \\
&a \\
&e
\end{align*}
\]
C.3 Group: analyse_undirected

C.3.4 get_average_clustering_coefficient_undirected

C.3.4.1 Expected inputs

network_direct_data Network represented in the common graph format
network_url URL pointing to network_direct_data

C.3.4.2 Outputs

avg_clustering Average clustering coefficient of the whole network
avg_clustering_url URL pointing to avg_clustering

C.3.4.3 Example

C.3.4.3.1 Input name: network
Network D.

C.3.4.3.2 Output name: avg_clustering

0.285714285714

C.3.5 get_bridges_undirected

C.3.5.1 Expected inputs

network_direct_data Network represented in the common graph format
network_url URL pointing to network_direct_data

C.3.5.2 Outputs

bridges List of bridge edges
bridges_url URL pointing to bridges

C.3.5.3 Example

C.3.5.3.1 Input name: network
Network D.
C.3 Group: analyse_undirected

C.3.5.3.2 Output name: bridges

<table>
<thead>
<tr>
<th>g</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

C.3.6 get_clique_by_size_undirected

C.3.6.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
- size: A number denoting the clique size of interest

C.3.6.2 Outputs

- cliques: List of cliques of nodes, which contain the specified number of nodes
- cliques_url: URL pointing to cliques

C.3.6.3 Example

C.3.6.3.1 Input name: network

Network $D$.

C.3.6.3.2 Input name: size

| 2 |

C.3.6.3.3 Output name: cliques

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>g</td>
<td>c</td>
</tr>
</tbody>
</table>
C.3 Group: analyse_undirected

C.3.7 get_cut_nodes_undirected

C.3.7.1 Expected inputs

- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`

C.3.7.2 Outputs

- `cut_nodes`: List of cut nodes
- `cut_nodes_url`: URL pointing to `cut_nodes`

C.3.7.3 Example

C.3.7.3.1 Input name: network

Input name: network

Network $D$.

C.3.7.3.2 Output name: cut_nodes

| c | b | d |

C.3.8 get_cyclic_core

C.3.8.1 Expected inputs

- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`

C.3.8.2 Outputs

- `cyclic_nodes`: List of nodes in the cyclic core
- `cyclic_nodes_url`: URL pointing to `cyclic_nodes`

C.3.8.3 Example

C.3.8.3.1 Input name: network

Input name: network

Network $D$.

C.3.8.3.2 Output name: cyclic_nodes
C.3.9  get_network_diameter_undirected

C.3.9.1  Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.3.9.2  Outputs

- diameter: The diameter of the network
- diameter_url: URL pointing to diameter

C.3.9.3  Example

C.3.9.3.1  Input name: network

Network D.

C.3.9.3.2  Output name: diameter

3

C.3.10  get_network_radius_undirected

C.3.10.1  Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.3.10.2  Outputs

- radius: The radius of the network
- radius_url: URL pointing to radius

C.3.10.3  Example

C.3.10.3.1  Input name: network

Network D.
C.310.3.2 Output name: radius

2

C.311 get\_subgraph\_undirected

C.3.11.1 Expected inputs

- network\_direct\_data: Network represented in the common graph format
- network\_url: URL pointing to network\_direct\_data
- nodelist\_direct\_data: List of nodes
- nodelist\_url: URL pointing to nodelist\_direct\_data

C.3.11.2 Outputs

- subgraph: Subgraph represented in the common graph format
- subgraph\_url: URL pointing to subgraph
- missing\_nodes: A list of any nodes which do not appear in the subgraph
- missing\_nodes\_url: URL pointing to missing\_nodes

C.3.11.3 Example

C.3.11.3.1 Input name: network

Network $D$.

C.3.11.3.2 Input name: nodelist

$\begin{array}{c} b \\ c \\ d \\ e \end{array}$

C.3.11.3.3 Output name: subgraph

$\begin{array}{cc} b & c \\ b & d \\ c & d \\ c & g \end{array}$

C.3.11.3.4 Output name: missing

In this case, an empty list, as all the specified nodes are connected in the subgraph.
C.3.12 largest_connected_component_undirected

C.3.12.1 Expected inputs

<table>
<thead>
<tr>
<th>network_direct_data</th>
<th>Network represented in the common graph format</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_url</td>
<td>URL pointing to network_direct_data</td>
</tr>
</tbody>
</table>

C.3.12.2 Outputs

<table>
<thead>
<tr>
<th>largest_component</th>
<th>Largest component represented in the common graph format</th>
</tr>
</thead>
<tbody>
<tr>
<td>largest_component_url</td>
<td>URL pointing to largest_component</td>
</tr>
</tbody>
</table>

C.3.12.3 Example

C.3.12.3.1 Input name: network

Network \( E \).

C.3.12.3.2 Output name: largest_component

```
i
h
f
```

C.3.13 list_connected_components_undirected

C.3.13.1 Expected inputs

<table>
<thead>
<tr>
<th>network_direct_data</th>
<th>Network represented in the common graph format</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_url</td>
<td>URL pointing to network_direct_data</td>
</tr>
</tbody>
</table>

C.3.13.2 Outputs

<table>
<thead>
<tr>
<th>list_components</th>
<th>List of the nodes in each connected component of the network</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_components_url</td>
<td>URL pointing to list_components</td>
</tr>
</tbody>
</table>

C.3.13.3 Example

C.3.13.3.1 Input name: network

Network \( E \).
C.3.13.3.2 Output name: list_components

```
i
h
g
f
a
c
b
e
d
j
```

C.3.14 node_clustering_coefficient_undirected

C.3.14.1 Expected inputs

- `network_direct_data` Network represented in the common graph format
- `network_url` URL pointing to `network_direct_data`
- `node` Query node of interest

C.3.14.2 Outputs

- `clustering` Clustering coefficient of query node
- `clustering_url` URL pointing to `clustering`

C.3.14.3 Example

C.3.14.3.1 Input name: network

Network $D$.

C.3.14.3.2 Input name: node

```
c
```

C.3.14.3.3 Output name: clustering

```
c 0.333333333333
```
C.3.15 node_degree_undirected

C.3.15.1 Expected inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_direct_data</td>
<td>Network represented in the common graph format</td>
</tr>
<tr>
<td>network_url</td>
<td>URL pointing to network_direct_data</td>
</tr>
<tr>
<td>node</td>
<td>Query node of interest</td>
</tr>
</tbody>
</table>

C.3.15.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>Query node and its degree</td>
</tr>
<tr>
<td>degree_url</td>
<td>URL pointing to degree</td>
</tr>
</tbody>
</table>

C.3.15.3 Example

C.3.15.3.1 Input name: network

Network $E$.

C.3.15.3.2 Input name: node

$$h$$

C.3.15.3.3 Output name: degree

$$h \quad 2$$

C.3.16 query_adjacency_matrix_undirected

C.3.16.1 Expected inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_direct_data</td>
<td>Network represented in the common graph format</td>
</tr>
<tr>
<td>network_url</td>
<td>URL pointing to network_direct_data</td>
</tr>
<tr>
<td>node</td>
<td>Query node of interest</td>
</tr>
</tbody>
</table>

C.3.16.2 Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacency</td>
<td>Query node, followed by tab-delimited list of nodes adjacent to it</td>
</tr>
<tr>
<td>adjacency_url</td>
<td>URL pointing to adjacency</td>
</tr>
</tbody>
</table>

C.3.16.3 Example

C.3.16.3.1 Input name: network

Network $E$. 

204
C.3 Group: analyse_undirected

C.3.16.3.2 Input name: node

\[
g
\]

C.3.16.3.3 Output name: adjacency

\[
g \quad h \quad f
\]

C.3.17 rank_betweenness_undirected

C.3.17.1 Expected inputs

- `network_direct_data` Network represented in the common graph format
- `network_url` URL pointing to `network_direct_data`

C.3.17.2 Outputs

- `betweenness_ranking` List of nodes and associated betweenness centralities, ranked from highest to lowest
- `betweenness_ranking_url` URL pointing to `betweenness_ranking`

C.3.17.3 Example

C.3.17.3.1 Input name: network

Network D.

C.3.17.3.2 Output name: betweenness_ranking

\[
\begin{array}{l}
b & 0.4 \\
d & 0.4 \\
c & 0.333333333333 \\
a & 0.0 \\
e & 0.0 \\
g & 0.0 \\
f & 0.0 \\
\end{array}
\]

C.3.18 rank_closeness_undirected

C.3.18.1 Expected inputs

- `network_direct_data` Network represented in the common graph format
- `network_url` URL pointing to `network_direct_data`
C.3 Group: analyse_undirected

C.3.18.2 Outputs

closeness_ranking  List of nodes and associated closeness centralities, ranked from highest to lowest

closeness_ranking_url  URL pointing to closeness_ranking

C.3.18.3 Example

C.3.18.3.1 Input name: network

Network D.

C.3.18.3.2 Output name: closeness_ranking

<table>
<thead>
<tr>
<th>Node</th>
<th>Closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0.75</td>
</tr>
<tr>
<td>d</td>
<td>0.75</td>
</tr>
<tr>
<td>c</td>
<td>0.666666666667</td>
</tr>
<tr>
<td>f</td>
<td>0.545454545455</td>
</tr>
<tr>
<td>a</td>
<td>0.461538461538</td>
</tr>
<tr>
<td>e</td>
<td>0.461538461538</td>
</tr>
<tr>
<td>g</td>
<td>0.428571428571</td>
</tr>
</tbody>
</table>

C.3.19 rank_clustering_coefficients_undirected

C.3.19.1 Expected inputs

network_direct_data  Network represented in the common graph format

network_url  URL pointing to network_direct_data

C.3.19.2 Outputs

clustering_ranking  List of nodes and associated clustering coefficients, ranked from highest to lowest

clustering_ranking_url  URL pointing to clustering_ranking

C.3.19.3 Example

C.3.19.3.1 Input name: network

Network D.

C.3.19.3.2 Output name: clustering_ranking
C.3 Group: analyse_undirected

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>1.0</td>
</tr>
<tr>
<td>c</td>
<td>0.333333333333</td>
</tr>
<tr>
<td>b</td>
<td>0.333333333333</td>
</tr>
<tr>
<td>d</td>
<td>0.333333333333</td>
</tr>
<tr>
<td>a</td>
<td>0.0</td>
</tr>
<tr>
<td>e</td>
<td>0.0</td>
</tr>
<tr>
<td>g</td>
<td>0.0</td>
</tr>
</tbody>
</table>

C.3.20 rank_degrees_undirected

C.3.20.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.3.20.2 Outputs

- degree_ranking: List of nodes and associated degrees, ranked from highest to lowest
- degree_ranking_url: URL pointing to degree_ranking

C.3.20.3 Example

C.3.20.3.1 Input name: network

Network $D$.

C.3.20.3.2 Output name: degree_ranking

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
</tr>
</tbody>
</table>

C.3.21 rank_secondary_degrees_undirected

C.3.21.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
C.3 Group: analyse_undirected

C.3.21.2 Outputs

- secondary_degree_ranking: List of nodes and associated degrees, ranked from highest to lowest.
- secondary_degree_ranking_url: URL pointing to secondary_degree_ranking.

C.3.21.3 Example

C.3.21.3.1 Input name: network

Network D.

C.3.21.3.2 Output name: secondary_degree_ranking

<table>
<thead>
<tr>
<th>Node</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>d</td>
<td>7</td>
</tr>
<tr>
<td>f</td>
<td>6</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
<tr>
<td>g</td>
<td>2</td>
</tr>
</tbody>
</table>

C.3.22 remove_edges_undirected

C.3.22.1 Expected inputs

- network_direct_data: Network represented in the common graph format.
- network_url: URL pointing to network_direct_data.
- edgelist: A list of edges represented in the common graph format.
- edgelist_url: URL pointing to edgelist.

C.3.22.2 Outputs

- new_network: Network with specified edges removed.
- new_network_url: URL pointing to new_network.

C.3.22.3 Example

C.3.22.3.1 Input name: network

Network D.

C.3.22.3.2 Input name: edgelist

A list of edges to be removed:
C.3 Group: analyse_undirected

\[
\begin{array}{cc}
    a & b \\
    d & e \\
    c & d \\
\end{array}
\]

C.3.22.3.3 Output name: new_network

\[
\begin{array}{cc}
    c & b \\
    c & g \\
    b & d \\
    b & f \\
    d & f \\
    a \\
    e \\
\end{array}
\]

C.3.23 remove_nodes

C.3.23.1 Expected inputs

- network\_direct\_data: Network represented in the common graph format
- network\_url: URL pointing to network\_direct\_data
- nodelist: A list of nodes
- nodelist\_url: URL pointing to nodelist

C.3.23.2 Outputs

- new\_network: Network with specified nodes removed
- new\_network\_url: URL pointing to new\_network

C.3.23.3 Example

C.3.23.3.1 Input name: network

Network \( D \).

C.3.23.3.2 Input name: nodelist

\[
\begin{array}{c}
    c \\
    e \\
\end{array}
\]
C.3 Group: analyse_undirected

C.3.23.3 Output name: new_network

\begin{verbatim}
a b
b d
b f
d f
g
\end{verbatim}

C.3.24 remove_self_loops_undirected

C.3.24.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.3.24.2 Outputs

- network_self_loops_removed: Network in common graph format, with no self loops
- network_self_loops_removed_url: URL pointing to network_self_loops_removed
- nodes_with_self_loops: List of network nodes with self loops
- nodes_with_self_loops_url: URL pointing to nodes_with_self_loops

C.3.24.3 Example

C.3.24.3.1 Input name: network

Network $E$.

C.3.24.3.2 Output name: network_self_loops_removed

\begin{verbatim}
a c
c b
d e
g h
g f
i h
j
\end{verbatim}

C.3.24.3.3 Output name: nodes_with_self_loops

\begin{verbatim}
j
\end{verbatim}
C.3 Group: analyse_undirected

C.3.25 remove_singleton_nodes

C.3.25.1 Expected inputs

- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`

C.3.25.2 Outputs

- `network_singletons_removed`: Network in common graph format
- `network_singletons_removed_url`: URL pointing to `network_singletons_removed`
- `singletons_list`: List of singleton nodes which have been removed from the network
- `singletons_list_url`: URL pointing to `singletons_list`

C.3.25.3 Example

C.3.25.3.1 Input name: network

Network $E$.

C.3.25.3.2 Output name: `network_singletons_removed`

\[
\begin{array}{cc}
a & b \\
b & c \\
a & c \\
d & e \\
f & g \\
g & h \\
h & i \\
\end{array}
\]

C.3.25.3.3 Output name: `singletons_list`

\[
\begin{array}{c}
j \\
\end{array}
\]

C.3.26 size_distribution_connected_components_undirected

C.3.26.1 Expected inputs

- `network_direct_data`: Network represented in the common graph format
- `network_url`: URL pointing to `network_direct_data`

C.3.26.2 Outputs

- `distribution`: Size distribution of connected components
- `distribution_url`: URL pointing to `distribution`
C.3.26.3 Example

C.3.26.3.1 Input name: network

Network $E$.

C.3.26.3.2 Output name: distribution

<table>
<thead>
<tr>
<th>Component size</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

C.3.27 size_of_largest_clique_undirected

C.3.27.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data

C.3.27.2 Outputs

- largest_clique_size: Number of nodes in largest clique
- largest_clique_size_url: URL pointing to largest_clique_size

C.3.27.3 Example

C.3.27.3.1 Input name: network

Network $D$.

C.3.27.3.2 Output name: largest_clique_size

3

C.3.28 total_edges_undirected

C.3.28.1 Expected inputs

- network_direct_data: Network represented in the common graph format
- network_url: URL pointing to network_direct_data
C.3.28.2 Outputs

\texttt{total\_edges}  Number of edges in the network
\texttt{total\_edges\_url}  URL pointing to \texttt{total\_edges}

C.3.28.3 Example

C.3.28.3.1 Input name: network

Network $E$.

C.3.28.3.2 Output name: \texttt{total\_edges}

\begin{center}
8
\end{center}

C.3.29 \texttt{total\_nodes}

C.3.29.1 Expected inputs

\texttt{network\_direct\_data}  Network represented in the common graph format
\texttt{network\_url}  URL pointing to \texttt{network\_direct\_data}

C.3.29.2 Outputs

\texttt{total\_nodes}  Number of nodes in the network
\texttt{total\_nodes\_url}  URL pointing to \texttt{total\_nodes}

C.3.29.3 Example

C.3.29.3.1 Input name: network

Network $E$.

C.3.29.3.2 Output name: \texttt{total\_nodes}

\begin{center}
10
\end{center}

C.4 Group: format\_output

C.4.1 \texttt{common\_graph\_to\_dot\_directed}

C.4.1.1 Expected inputs

\texttt{network\_direct\_data}  Network represented in the common graph format
\texttt{network\_url}  URL pointing to \texttt{network\_direct\_data}
C.4 Group: format_output

C.4.1.2 Outputs
dotfile Network represented using the DOT language
dotfile_url URL pointing to dotfile

C.4.1.3 Example
C.4.1.3.1 Input name: network

```
 a   b
a   c
b   c
d   b
c   d
e   c
e   f
g   e
```

C.4.1.3.2 Output name: dotfile

```
digraph G {  
node [style=filled];  
overlap=scale;
a -> b;
a -> c;
b -> c;
d -> b;
c -> d;
e -> c;
e -> f;
g -> e;
}
```

C.4.2 common_graph_to_dot_undirected

C.4.2.1 Expected inputs

network_direct_data Network represented in the common graph format
network_url URL pointing to network_direct_data

C.4.2.2 Outputs

dotfile Network represented using the DOT language
dotfile_url URL pointing to dotfile
C.4 Group: format_output

C.4.2.3 Example

C.4.2.3.1 Input name: network

\[
\begin{array}{cc}
  a & b \\
  a & c \\
  b & c \\
  d & b \\
  c & d \\
  e & c \\
  e & f \\
  g & e \\
\end{array}
\]

C.4.2.3.2 Output name: dotfile

\[
digraph G 
\{
  edge [dir=none];
  node [style=filled];
  overlap=scale;
  a -> b;
  a -> c;
  b -> c;
  d -> b;
  c -> d;
  e -> c;
  e -> f;
  g -> e;
\}
\]

C.4.3 dot

C.4.3.1 Expected inputs

- **dotfile_direct_data**: Network represented in the DOT language
- **dotfile_url**: URL pointing to dotfile_direct_data
- **library_direct_data**: A custom PostScript library file
- **library_url**: A URL pointing to library_direct_data
- **format**: Image format (from the following list: canon, dot, fig, gd, gif, hpgl, imap, jpg, mif, mp, pcl, pic, plain, png, ps, ps2, svg, vrml, vtx, wbmp)

C.4.3.2 Outputs

- **resultgraph**: Network diagram rendered using hierarchical layout
- **resultgraph_url**: URL pointing to resultgraph

215
C.4.3.3 Example

C.4.3.3.1 Input name: dotfile

In this example, a directed network:

```
digraph G {
    node [style=filled];
    overlap=scale;
    a -> b;
    a -> c;
    b -> c;
    d -> b;
    c -> d;
    e -> c;
    e -> f;
    g -> e;
}
```

C.4.3.3.2 Input name: format

```
png
```

C.4.3.3.3 Output name: resultgraph
C.4.4 format_psi25_id_list

C.4.4.1 Expected inputs

- protein_ids_direct_data: List of protein ids
- protein_ids_url: URL pointing to protein_ids_direct_data
- psi25_proteins_file_direct_data: PSI-MI 2.5 proteins file, as generated by psi25_to_common_graph
- psi25_proteins_file_url: URL pointing to psi25_proteins_file_direct_data

C.4.4.2 Outputs

- protein_names: List of PSI-MI descriptive names
- protein_names_url: URL pointing to protein_names

C.4.4.3 Example

C.4.4.3.1 Input name: protein_ids

| Q9Y631  |
| Q99461  |
| Q93009  |
| Q8IX98  |

C.4.4.3.2 Input name: psi25_proteins_file

In this example, the first five lines of the Viruses.psi25 from MINT:

| Q9Y631 | Transformation/transcription domain–associated protein Homo sapiens |
| Q99461 | Mitogen–activated protein kinase kinase kinase 5 Homo sapiens |
| Q4JQW5 | ORF10 Human herpesvirus |
| Q93009 | Ubiquitin carboxyl–terminal hydrolase 7 Homo sapiens |
| P88918 | ORF 28 Human herpesvirus |

C.4.4.3.3 Output name: protein_names

| Transformation/transcription domain–associated protein |
| Mitogen–activated protein kinase kinase kinase 5 |
| Ubiquitin carboxyl–terminal hydrolase 7 |
| SH3 domain–containing kinase–binding protein 1 |
C.4.5 format_sbm_id_list

C.4.5.1 Expected inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbml_ids_direct_data</td>
<td>List of SBML ids</td>
</tr>
<tr>
<td>sbml_ids_url</td>
<td>URL pointing to sbml_ids_direct_data</td>
</tr>
<tr>
<td>sbml_species_file_direct_data</td>
<td>SBML species file, as generated by sbml_to_common_graph</td>
</tr>
<tr>
<td>sbml_species_file_url</td>
<td>URL pointing to sbml_species_file_direct_data</td>
</tr>
</tbody>
</table>

C.4.5.2 Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbml_names</td>
<td>List of SBML names</td>
</tr>
<tr>
<td>sbml_names_url</td>
<td>URL pointing to sbml_names</td>
</tr>
</tbody>
</table>

C.4.5.3 Example

C.4.5.3.1 Input name: sbml_ids

```
ACP_c
Lfmkynr_c
Nacasp_c
2kmb_c
```

C.4.5.3.2 Input name: sbml_species_file

In this example the first five species in the Palsson human metabolic network:

```
ACP_c acylcarrierprotein NONE ...
ACP_m acylcarrierprotein NONE ...
Asn_X_Ser_Thr_b protein_linkedasparagineresidue_N_glycosylationsite ...
Asn_X_Ser_Thr_l protein_linkedasparagineresidue_N_glycosylationsite ...
Asn_X_Ser_Thr_r protein_linkedasparagineresidue_N_glycosylationsite ...
<snipped>
```

C.4.5.3.3 Output name: sbml_names

```
acylcarrierprotein
L_Formylkynurenine
N_Acetyl_L_aspartate
2_keto_4_methylthiobutyrate
```
C.4.6 neato

C.4.6.1 Expected inputs

- **dotfile**
  - Network represented in the DOT language
- **dotfile_url**
  - URL pointing to `dotfile`
- **library**
  - A custom PostScript library file
- **library_url**
  - A URL pointing to `library`
- **format**
  - Image format (from the following list: canon, dot, fig, gd, gif, hpgl, imap, jpg, mif, mp, pcl, pic, plain, png, ps, ps2, svg, vrml, vtx, wbmp)

C.4.6.2 Outputs

- **resultgraph**
  - Network diagram rendered using spring-embedded layout
- **resultgraph_url**
  - URL pointing to `resultgraph`

C.4.6.3 Example

C.4.6.3.1 Input name: dotfile

In this example, an undirected network:

```dot
digraph G {
edge [dir=none];
node [style=filled];
overlap=scale;
a -> b;
a -> c;
b -> c;
d -> b;
c -> d;
e -> c;
e -> f;
g -> e;
}
```

C.4.6.3.2 Input name: format

```
png
```
C.4.6.3.3 Output name: resultgraph

C.5 Group: retrieve

C.5.1 query_atpid

C.5.1.1 Expected inputs

query_protein String representing an A. thaliana AGI identifier.

C.5.1.2 Outputs

atpid_interactors List of proteins that interact with the query protein.
atpid_interactors_url URL pointing to atpid_interactors

C.5.1.3 Example

C.5.1.3.1 Input name: query_protein

| AT2G01250 |

C.5.1.3.2 Output name: atpid_interactors

| AT2G44120 | AT5G14520 | AT1G80750 | AT3G13580 | AT4G01560 |
C.5 Group: retrieve

C.5.2 query_inferred

C.5.2.1 Expected inputs

query_protein String representing an A. thaliana AGI identifier.

C.5.2.2 Outputs

inferred_interactors List of proteins that interact with the query protein.

inferred_interactors_url URL pointing to inferred_interactors

C.5.2.3 Example

C.5.2.3.1 Input name: query_protein

| AT2G01250 |

C.5.2.3.2 Output name: inferred_interactors

| AT5G14520  |
| AT1G36730  |
| AT5G15550  |
| AT2G36930  |
| AT1G21160  |
| AT3G11964  |
| AT1G03530  |
| AT1G10170  |
| AT1G03000  |
| AT4G01560  |
| AT2G18220  |
| AT3G01610  |
| AT2G30770  |
| AT1G13160  |
| AT3G16840  |
| AT1G05520  |
| AT4G38630  |
| AT4G11820  |
| AT4G26840  |
| AT3G27530  |
| AT4G17620  |
| AT3G55620  |
| AT1G06380  |
| AT1G8830   |
| AT3G55410  |
| AT1G72440  |
| AT3G13640  |
| AT1G19910  |
| AT4G26910  |
C.6 Group: transform

Extracts from real biological networks are used to demonstrate applicability of web services in this group.

C.6.1 common_graph_to_sif

C.6.1.1 Expected inputs

- **common_direct_data**: Network represented in the common graph format
- **common_url**: URL pointing to common_direct_data
- **relationship**: A string representing the relationship type between the entities in the network. Common types include pp to represent PPI interactions, and pd to represent protein-DNA interactions (e.g. binding of a TF upstream of a gene). A full list of types is available from the Cytoscape website\(^1\).

C.6.1.2 Outputs

- **sif**: Network represented in SIF
- **sif_url**: URL pointing to sif

C.6.1.3 Example

C.6.1.3.1 Input name: common

In this example, the first five interactions between proteins in the AtPID dataset:

| AT2G01250 | AT2G44120 |
| AT5G07090 | AT5G58420 |
| AT4G16720 | AT4G17390 |
| AT5G10400 | AT5G65360 |
| AT5G10390 | AT5G65360 |

C.6.1.3.2 Input name: relationship

A full list of possible interaction types is available from the Cytoscape website\(^1\):

\(^1\)http://www.cytoscape.org/cgi-bin/moin.cgi/Cytoscape_User_Manual/Network_Formats

\(^2\)http://www.cytoscape.org/cgi-bin/moin.cgi/Cytoscape_User_Manual/Network_Formats
C.6 Group: transform

C.6.1.3.3 Output name: sif

AT2G01250 pp AT2G4120
AT5G07090 pp AT5G58420
AT4G16720 pp AT4G17390
AT5G10400 pp AT5G65360
AT5G10390 pp AT5G65360

C.6.2 psi25_to_common_graph

C.6.2.1 Expected inputs

- psi25_direct_data: Network represented in the PSI-MI Level 2 format
- psi25_url: URL pointing to psi25_direct_data

C.6.2.2 Outputs

- common: Network represented in the common graph format
- common_url: URL pointing to common
- proteins: Proteins file (full details of this file are given in the example below)
- proteins_url: URL pointing to proteins

C.6.2.3 Example

C.6.2.3.1 Input name: psi25

In this example, the first interaction between human viral proteins in the Viruses-3.psi25.xml dataset, available to download from the MINT FTP site¹ (the example has been modified to show the relevant parts of the document):

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<entrySet version="5" minorVersion="3" level="2"
  xmlns="net:sf:psidev:mi"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<entry>
```

¹ftp://mint.bio.uniroma2.it/pub/release/psi/current/psi25/dataset/
<source releaseDate="2009-04-16">
  <names>
    <shortLabel>MINT</shortLabel>
    <fullName>MINT, Dpt of Biology, University of Rome Tor Vergata</fullName>
  </names>
  <xref>
    <primaryRef secondary="mint" refTypeAc="MI:0356"
      refType="identity" id="MI:0471" dbAc="MI:0488" db="psi−mi"/>
  </xref>
</source>

<experimentList>
  <!-- SNIPPED -->
</experimentList>

<interatorList>
  <interactor id="6385">
    <!-- SNIPPED -->
  </interactor>
  <interactor id="6388">
    <!-- SNIPPED -->
  </interactor>
</interactorList>

<interactionList>
  <interaction id="6382">
    <names>
      <shortLabel>ve6−srtd1</shortLabel>
    </names>
    <xref>
      <primaryRef refTypeAc="MI:0356" refType="identity"
        id="MINT−73312" dbAc="MI:0471" db="mint"/>
    </xref>
    <experimentList>
      <experimentRef>6383</experimentRef>
    </experimentList>
    <participantList>
      <participant id="6384">
        <names>
          ...
        </names>
      </participant>
    </participantList>
  </interaction>
</interactionList>
C.6 Group: transform

<shortLabel>srtd1_human</shortLabel>
<names>
<interactorRef>6385</interactorRef>
<participant id="6387">
<names>
<shortLabel>ve6_hp16</shortLabel>
<interactorRef>6388</interactorRef>
</participant>
</participantList>
</interaction>
</interactionList>
</entrySet>

C.6.2.3.2 Output name: common

Q9BUE7  Q71BI7
O00530  Q14637
Q01860  P06438
Q44M00  Q9H4Z5
P06463  Q7Z5D1

C.6.2.3.3 Output name: proteins

A file containing information about proteins in the network, where the first column is the protein accession, the second is the descriptive name, and the third is the organism:

P06790  Regulatory protein E2 Human papillomavirus type 18
P04015  Regulatory protein E2 Human papillomavirus type 11
Q0Y6K0  NF–kappa–B essential modulator Homo sapiens
Q15328  Transcription factor E2F4 Homo sapiens
Q9BUE7  SERTA domain–containing protein 1 Homo sapiens
C.6 Group: transform

C.6.3 psitab_to_common_graph

C.6.3.1 Expected inputs

<table>
<thead>
<tr>
<th>psitab_direct_data</th>
<th>Network represented in the PSI-MI Level 2 format</th>
</tr>
</thead>
<tbody>
<tr>
<td>psi25_url</td>
<td>URL pointing to psitab_direct_data</td>
</tr>
</tbody>
</table>

C.6.3.2 Outputs

<table>
<thead>
<tr>
<th>common</th>
<th>Network represented in the common graph format</th>
</tr>
</thead>
<tbody>
<tr>
<td>common_url</td>
<td>URL pointing to common</td>
</tr>
</tbody>
</table>

C.6.3.3 Example

C.6.3.3.1 Input name: psitab

In this example, the first five interactions from the DIP *Mus musculus* dataset, available to download from the DIP website ¹ (only the first two columns of each interaction record are given here):

<table>
<thead>
<tr>
<th>DIP−278N</th>
<th>uniprotkb : Q60520</th>
<th>DIP−951N</th>
<th>uniprotkb : Q9Y618</th>
<th>&lt;snipped&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP−445N</td>
<td>uniprotkb : P46414</td>
<td>DIP−559N</td>
<td>&lt;snipped&gt;</td>
<td></td>
</tr>
<tr>
<td>DIP−497N</td>
<td>uniprotkb : P43063</td>
<td>DIP−445N</td>
<td>uniprotkb : P46414</td>
<td>&lt;snipped&gt;</td>
</tr>
<tr>
<td>DIP−165N</td>
<td>uniprotkb : P15692</td>
<td>DIP−213N</td>
<td>uniprotkb : P35918</td>
<td>&lt;snipped&gt;</td>
</tr>
<tr>
<td>DIP−1084N</td>
<td>uniprotkb : P03995</td>
<td>DIP−1081N</td>
<td>uniprotkb : P04273</td>
<td>&lt;snipped&gt;</td>
</tr>
</tbody>
</table>

C.6.3.3.2 Output name: common

<table>
<thead>
<tr>
<th>Q60520</th>
<th>Q9Y618</th>
</tr>
</thead>
<tbody>
<tr>
<td>P46414</td>
<td></td>
</tr>
<tr>
<td>P43063</td>
<td>P46414</td>
</tr>
<tr>
<td>P15692</td>
<td>P35918</td>
</tr>
<tr>
<td>P03995</td>
<td>P04273</td>
</tr>
</tbody>
</table>

C.6.4 sbml_to_common_graph

C.6.4.1 Expected inputs

<table>
<thead>
<tr>
<th>sbml_direct_data</th>
<th>Network represented in SBML format</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbml_url</td>
<td>URL pointing to sbml_direct_data</td>
</tr>
</tbody>
</table>

¹http://dip.doe-mbi.ucla.edu/dip/Download.cgi?SM=7&TX=10090
C.6.4.2 Outputs

<table>
<thead>
<tr>
<th>common</th>
<th>Network represented in the common graph format</th>
</tr>
</thead>
<tbody>
<tr>
<td>common_url</td>
<td>URL pointing to common</td>
</tr>
<tr>
<td>species</td>
<td>Species file (full details of this file are given in the example below)</td>
</tr>
<tr>
<td>species_url</td>
<td>URL pointing to species</td>
</tr>
<tr>
<td>reactions</td>
<td>Reactions file (full details of this file are given in the example below)</td>
</tr>
<tr>
<td>reactions_url</td>
<td>URL pointing to reactions</td>
</tr>
</tbody>
</table>

C.6.4.3 Example

C.6.4.3.1 Input name: sbml

In this example caprolactam degradation in *E. coli*:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version1" level="2"
      id="eco00930" name="eco00930">
  <model id="eco00930" name="eco00930">
    <listOfCompartments>
      <compartment id="default" name="default" />
      <compartment id="uVol" name="uVol" outside="default" />
    </listOfCompartments>
    <listOfSpecies>
      <species id="E3_space_7_space_1_space_2_minus_" name="3.7.1.–" compartment="uVol" initialAmount="0.0"></species>
      <species id="E2_space_6_space_1_space_2_minus_" name="2.6.1.–" compartment="uVol" initialAmount="0.0"></species>
      <species id="Adipate_space_semaldehyde" name="Adipate semialdehyde" compartment="uVol" initialAmount="0.0"></species>
      <species id="6_minus_Aminohexanoate" name="6–Aminohexanoate" compartment="uVol" initialAmount="0.0"></species>
      <species id="Cyclohexan_minus_1_space_2_minus_dione" name="Cyclohexan–1,2–dione" compartment="uVol" initialAmount="0.0"></species>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="R05507" name="R05507" reversible="true">
        <listOfReactants>
          <speciesReference species="6_minus_Aminohexanoate" />
        </listOfReactants>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```
C.6.4.3.2 Output name: common

```
<reaction id="R05100" name="R05100" reversible="true">
  <listOfReactants>
    <speciesReference species="Cyclohexan_minus_1_space_2_minus_dione"/>
  </listOfReactants>
  <listOfProducts>
    <speciesReference species="Adipate_space_semaldehyde"/>
  </listOfProducts>
  <listOfModifiers>
    <modifierSpeciesReference species="E2_space_6_space_1_space__minus__"/>
  </listOfModifiers>
</reaction>
```

C.6.4.3.3 Output name: reactions

A file containing information about reactions, where the first column is the reaction identifier, the second is its name, the third indicates if it is reversible, and the fourth
indicates if it is fast:

<table>
<thead>
<tr>
<th>R05507</th>
<th>R05507</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>R05100</td>
<td>R05100</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

### C.6.4.3.4 Output name: species

A file containing information about the species (in SBML this refers to molecules) in the network, where the first column is the species identifier, the second is the name, the third is the type, the fourth is the compartment, the fifth is the initial amount, the sixth is the initial concentration, the seventh is the substance units, the eighth indicates if it has only substance units, the ninth indicates if it has a boundary condition, the tenth indicates the charge, and the eleventh indicates if it is constant:

<table>
<thead>
<tr>
<th>E3</th>
<th>3.7.1</th>
<th>NONE</th>
<th>uVol</th>
<th>0.0</th>
<th>NONE</th>
<th>false</th>
<th>false</th>
<th>NONE</th>
<th>NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2</td>
<td>2.6.1</td>
<td>NONE</td>
<td>uVol</td>
<td>0.0</td>
<td>NONE</td>
<td>false</td>
<td>false</td>
<td>NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>Adipate semialdehyde</td>
<td>NONE</td>
<td>uVol</td>
<td>0.0</td>
<td>NONE</td>
<td>false</td>
<td>false</td>
<td>NONE</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>Adipate</td>
<td>Adipate</td>
<td>semialdehyde</td>
<td>NONE</td>
<td>uVol</td>
<td>0.0</td>
<td>NONE</td>
<td>false</td>
<td>false</td>
<td>NONE</td>
</tr>
<tr>
<td>Aminohexanoate</td>
<td>6-Aminohexanoate</td>
<td>NONE</td>
<td>uVol</td>
<td>0.0</td>
<td>NONE</td>
<td>false</td>
<td>false</td>
<td>NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>Cyclohexan-1,2-dione</td>
<td>Cyclohexan-1,2-dione</td>
<td>NONE</td>
<td>uVol</td>
<td>0.0</td>
<td>NONE</td>
<td>false</td>
<td>false</td>
<td>NONE</td>
<td>NONE</td>
</tr>
</tbody>
</table>

#### C.6.5 sif_to_common_graph

##### C.6.5.1 Expected inputs

- **sif_direct_data**: Network represented in SIF
- **sif_url**: URL pointing to sif_direct_data

##### C.6.5.2 Outputs

- **common**: Network represented in the common graph format
- **common_url**: URL pointing to common

##### C.6.5.3 Example

##### C.6.5.3.1 Input name: sif

In this example, the first five interactions in the AtPID dataset:
### C.6 Group: transform

| AT2G01250 | pp | AT2G44120 |
| AT5G07090 | pp | AT5G58420 |
| AT4G16720 | pp | AT4G17390 |
| AT5G10400 | pp | AT5G65360 |
| AT5G10390 | pp | AT5G65360 |

#### C.6.5.3.2 Output name: common

| AT2G01250 | AT2G44120 |
| AT5G07090 | AT5G58420 |
| AT4G16720 | AT4G17390 |
| AT5G10400 | AT5G65360 |
| AT5G10390 | AT5G65360 |
Appendix D

Detailed workflow outputs

D.1 A workflow to identify source and sink metabolites in a network model of metabolism

D.1.1 Source metabolites

<table>
<thead>
<tr>
<th>Metabolite</th>
<th>Chemical Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Androsterone glucuronide</td>
<td>C25H38O8</td>
</tr>
<tr>
<td>ADPribose</td>
<td>C15H21N5O14P2</td>
</tr>
<tr>
<td>Imidazole –4-acetaldehyde</td>
<td>C5H6N2O</td>
</tr>
<tr>
<td>dTMP</td>
<td>C10H13N2O8P</td>
</tr>
<tr>
<td>Isocitrate</td>
<td>C6H5O7</td>
</tr>
<tr>
<td>_1-Phosphatidyl-1D-myo-inositol 5-phosphate (Homo sapiens)</td>
<td>C9H15O12P2FULLRCO2FULLR2CO2</td>
</tr>
<tr>
<td>L-Proline</td>
<td>C5H9NO2</td>
</tr>
<tr>
<td>XTP</td>
<td>C10H11N4O15P3</td>
</tr>
<tr>
<td>L5-Hydroxykynurenine</td>
<td>C10H12N2O4</td>
</tr>
<tr>
<td>Dihydroxyacetone phosphate</td>
<td>C3H5O6P</td>
</tr>
<tr>
<td>GI3 (homo sapiens)</td>
<td>C52H88N3O28FULLRCO</td>
</tr>
<tr>
<td>NMN</td>
<td>C11H14N2O8P</td>
</tr>
<tr>
<td>1D-myo-Inositol 3,4,5,6-tetrakisphosphate</td>
<td>C6H8O18P4</td>
</tr>
<tr>
<td>Procollagen L-lysine</td>
<td>C7H14N3O2R2</td>
</tr>
<tr>
<td>3-Oxohexadecanoyl-CoA</td>
<td>C37H60N7O18P3S</td>
</tr>
<tr>
<td>Nicotinamide adenine dinucleotide phosphate – reduced</td>
<td>C21H26N7O17P3</td>
</tr>
<tr>
<td>protein-linked serine residue (glycosaminoglycan attachment site)_XH</td>
<td></td>
</tr>
<tr>
<td>D-Lactaldehyde</td>
<td>C3H6O2</td>
</tr>
<tr>
<td>P1,P4-Bis(5’-adenosyl) tetraphosphate</td>
<td>C20H24N10O19P4</td>
</tr>
<tr>
<td>Ubiquinone –10_C59H90O4</td>
<td></td>
</tr>
<tr>
<td>2-Oxoglutarate</td>
<td>C5H4O5</td>
</tr>
<tr>
<td>N2-Acetyl-L-ornithine</td>
<td>C7H14N2O3</td>
</tr>
<tr>
<td>N-Trimethyl-2-aminoethylphosphonate</td>
<td>C5H13NO3P</td>
</tr>
<tr>
<td>(13E)-11alpha-Hydroxy-9,15-dioxoprost –13-enoate</td>
<td>C20H31O5</td>
</tr>
<tr>
<td>Betaine aldehyde</td>
<td>C5H12NO</td>
</tr>
<tr>
<td>R group 2 Coenzyme A homosapiens_XCO2C21H31N7O15P3S</td>
<td></td>
</tr>
<tr>
<td>protein-linked serine or threonine residue (O-glycosylation site)_XH</td>
<td></td>
</tr>
<tr>
<td>Malonyl–CoA_C24H33N7O19P3</td>
<td></td>
</tr>
</tbody>
</table>
D.1 A workflow to identify source and sink metabolites in a network model of metabolism

DNA\(_{\text{C10H17O8PR2}}\)

trans-4-Hydroxy-L-proline\(_{\text{C5H9NO3}}\)

phosphatidylinositol (homo sapiens)\(_{\text{C9H16O9PFULLRCO2FULLR2CO2}}\)

dGMP\(_{\text{C10H12N5O7P}}\)

2-Phosphoglycerate\(_{\text{C2H2O6P}}\)

glycophosphatidylinositol (GPI)–anchored protein precursor\(_{XY}\)

dCMP\(_{\text{C9H12N3O7P}}\)

R group 1 Coenzyme A homo sapiens\(_{XCO2C21H31N7O15P3S}\)

pregnenolone sulfate\(_{C21H31O5S}\)

L–Carnosine\(_{\text{C9H14N4O3}}\)

Trehalose\(_{\text{C12H22O11}}\)

Hydroxyproline\(_{\text{C3H3O4}}\)

(alpha−D−mannosyl)2−beta−D−mannosyl−N−acetylglucosamine\(_{C26H45NO21}\)

3',5'−Cyclic GMP\(_{\text{C10H11N5O7P}}\)

Isoptenyl dipiphosphate\(_{\text{C5H9O7P2}}\)

Hydroxymethylglutaryl−CoA\(_{\text{C27H39N7O20P3S}}\)

Selenomethionine\(_{\text{C5H11NO2Se}}\)

Perillyl aldehyde\(_{\text{C10H14O}}\)

ADPglucose\(_{\text{C16H23N5O15P2}}\)

UMP\(_{\text{C9H11N2O8P}}\)

Nicotinate D−ribonucleotide\(_{\text{C11H12NO9P}}\)

dAMP\(_{\text{C10H12N5O6P}}\)

D−Ornithine\(_{\text{C5H13N2O2}}\)

3−Hydroxy−N6,N6,N6−trimethyl−L−lysine\(_{\text{C9H20N2O3}}\)

ADPmannose\(_{\text{C16H23N5O15P2}}\)

N−Acetyl−L−glutamyl 5−phosphate\(_{\text{C7H9O8P}}\)

cis−beta−D−Glucosyl−2−hydroxycinnamate\(_{\text{C15H18O8}}\)

(R)−Pantothenate\(_{\text{C9H16NO5}}\)

N−Methylputrescine \(_{\text{C5H16N2}}\)

chitin\(_{\text{C24H41N3O16}}\)

CMP\(_{\text{C9H12N3O8P}}\)

trans−4−Hydroxycinnamate\(_{\text{C9H7O3}}\)

3α−Hydroxy−5beta−androstan−17−one\(_ {\text{C19H30O2}}\)

Lanosterol\(_ {\text{C30H50O}}\)

R total Coenzyme A \(_{\text{C10H12N5O6P}}\)

D−Ribulose\(_ {\text{C5H10O5}}\)

(2−Aminoethyl) phosphonate\(_ {\text{C2H7NO3P}}\)

Indole−3−acetaldehyde\(_ {\text{C10H9NO}}\)

D−Proline\(_ {\text{C5H9NO2}}\)

17alpha−Hydroxypregnenolone\(_ {\text{C21H32O3}}\)

cholesterol sulfate\(_ {\text{C27H45O4S}}\)

Ribitol\(_ {\text{C5H12O5}}\)

(S)−Methylmalonyl−CoA\(_ {\text{C25H35N7O19P3S}}\)

(S)−Methylmalonyl−CoA\(_ {\text{C25H35N7O19P3S}}\)

cAMP\(_ {\text{C10H11N5O6P}}\)

3−Hydroxypropionyl−CoA\(_ {\text{C24H36N7O18P3S}}\)

coenzyme C17H21NO4

Estrone 3−sulfate\(_ {\text{C18H21O15S}}\)

5−Hydroxyindoleacetaldehyde\(_ {\text{C10H7NO2}}\)

n2m2nnmsn (w/o peptide linkage)\(_ {\text{C58H97N5O41}}\)

Cholesterol\(_ {\text{C27H46O}}\)
D.1 A workflow to identify source and sink metabolites in a network model of metabolism

D.1.1 Sink metabolites

\[ \text{beta-1,4-mannose-\text{N-acetylglucosamine}} \}
\[ \text{C14H25NO11} \]
\[ \text{androsterone glucuronide} \}
\[ \text{C25H38O8} \]
\[ \text{Anthranilate} \]
\[ \text{C7H6NO2} \]
\[ \text{13-cis-retinolate} \]
\[ \text{C20H27O2} \]
\[ \text{L-Xylonate} \]
\[ \text{C5H9O6} \]
\[ \text{2-Hydroxyphenylacetate} \]
\[ \text{C8H7O3} \]
\[ \text{Trihexosyl ceramide (homo sapiens)} \]
\[ \text{C36H66N17O17FULLRCO} \]
\[ \text{Tetradecanoyl-CoA (n-C14:0CoA)} \]
\[ \text{C35H58N7O17P3S} \]
\[ \text{Dihydroxyacetone} \]
\[ \text{C3H6O3} \]
\[ \text{1-(2-Amino-3-hydroxyphenyl)-2,4-dioxobutanoate} \]
\[ \text{C10H8NO5} \]
\[ \text{alpha-D-Ribose 5-phosphate} \]
\[ \text{C5H9O8} \]
\[ \text{3alpha,7alpha,12alpha-Trihydroxy-5beta-cholestanoyl-CoA(8)} \]
\[ \text{C48H76N7O20P3S} \]
D.1 A workflow to identify source and sink metabolites in a network model of metabolism
### D.1 A workflow to identify source and sink metabolites in a network model of metabolism

<table>
<thead>
<tr>
<th>Compound Name</th>
<th>Chemical Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procollagen 5-hydroxy-L-lysine</td>
<td>C6H14N2O2</td>
</tr>
<tr>
<td>dCTP</td>
<td>C9H12N3O13P3</td>
</tr>
<tr>
<td>Hexadecanoate (n=C16:0)</td>
<td>C16H31O2</td>
</tr>
<tr>
<td>Cholesterol ester</td>
<td>C27H45XCO2</td>
</tr>
<tr>
<td>1-Methylpyrrolinium</td>
<td>C5H10N</td>
</tr>
<tr>
<td>Deamino-NAD</td>
<td>C21H24N6O15P2</td>
</tr>
<tr>
<td>Cob (II) alamin</td>
<td>C62H92CoN13O14P</td>
</tr>
<tr>
<td>4-hydroxy retinoic acid</td>
<td>C20H27O3</td>
</tr>
<tr>
<td>Imidazole-4-acetate</td>
<td>C5H10N</td>
</tr>
<tr>
<td>Nicotinamide adenine dinucleotide phosphate</td>
<td>C21H25N7O17P3</td>
</tr>
<tr>
<td>Coenzyme A</td>
<td>C21H32N7O16P3S</td>
</tr>
<tr>
<td>Phosphatidylinositol-3,5-bisphosphate (Homo sapiens)</td>
<td>C9H14O15P3</td>
</tr>
<tr>
<td>Deacetylated-(phosphoethanolaminyl-dimannosyl)-(phosphoethanolaminyl)-mannosyl-glucosaminyl-aclylphosphatidylinositol</td>
<td>C37H70N3O34P3</td>
</tr>
<tr>
<td>Sialyl-Tn antigen</td>
<td>C19H30N2O13X</td>
</tr>
<tr>
<td>Lysophosphatidic acid (homo sapiens)</td>
<td>C3H6O5PFULLRCO2</td>
</tr>
<tr>
<td>4,4-dimethylcholesta-8,14,24-trienol</td>
<td>C29H46O</td>
</tr>
<tr>
<td>Phosphate</td>
<td>HO4P</td>
</tr>
<tr>
<td>N-Formylanthranilate</td>
<td>C8H6NO3</td>
</tr>
<tr>
<td>Dodecanoyl-CoA (n=C12:0CoA)</td>
<td>C33H54N7O17P3S</td>
</tr>
<tr>
<td>1-Phosphatidylinositol 1D-myo-inositol 3-phosphate (Homo sapiens)</td>
<td>C9H15O12P2</td>
</tr>
<tr>
<td>5-Guanidino-2-oxopentanoate</td>
<td>C6H11N3O3</td>
</tr>
<tr>
<td>N-Methylserotonin</td>
<td>C11H15N2O</td>
</tr>
<tr>
<td>Melanin</td>
<td>C9H6NO4</td>
</tr>
<tr>
<td>5-Hydroxymelatonin</td>
<td>C13H16N2O3</td>
</tr>
<tr>
<td>dTDP-L-rhamnose</td>
<td>C16H24N2O15P2</td>
</tr>
<tr>
<td>Ethanolamine phosphate</td>
<td>C2H7NO4P</td>
</tr>
<tr>
<td>Ubiquinol</td>
<td>C59H92O4</td>
</tr>
<tr>
<td>Perillic acid</td>
<td>C10H13O2</td>
</tr>
<tr>
<td>Perillic acid</td>
<td>C10H13O2</td>
</tr>
<tr>
<td>18 hydroxy arachidonic acid</td>
<td>C20H31O3</td>
</tr>
<tr>
<td>CMP-N-trimethyl-2-aminoethylphosphonate</td>
<td>C14H25N4O10P2</td>
</tr>
<tr>
<td>Formyl-N-acetyl-5-methoxykynurenamine</td>
<td>C13H16N2O4</td>
</tr>
<tr>
<td>Phosphatidylinositol-3,4-bisphosphate (Homo sapiens)</td>
<td>C9H14O15P3</td>
</tr>
<tr>
<td>Selenophosphate</td>
<td>H2O3PSe</td>
</tr>
<tr>
<td>glucose-1,2-(2)</td>
<td>glucose-1,3-</td>
</tr>
<tr>
<td>Lipoate</td>
<td>C8H14O2S2</td>
</tr>
<tr>
<td>Glycophosphatidylinositol (GPI) signal sequence (C-terminal peptide)</td>
<td>Y</td>
</tr>
<tr>
<td>Hexacosanoyl-CoA (n=C26:0CoA)</td>
<td>C47H82N7O17P3</td>
</tr>
<tr>
<td>Succinate</td>
<td>C4H4O4</td>
</tr>
<tr>
<td>D-Ribulose 5-phosphate</td>
<td>C5H9O8P</td>
</tr>
<tr>
<td>Lignoceryl coenzyme A</td>
<td>C45H78N7O17P3S</td>
</tr>
<tr>
<td>12 hydroxy arachidonic acid</td>
<td>C20H31O3</td>
</tr>
<tr>
<td>glucose-1,3-mannose oligosaccharide</td>
<td>C12H22O11</td>
</tr>
</tbody>
</table>
D.1 A workflow to identify source and sink metabolites in a network model of metabolism
D.2 A workflow to annotate metabolic pathways with PPIs

HSP72\textsubscript{YEAST} Heat shock protein SSA2
interacts with

ODP\textsubscript{2}YEAST Pyruvate dehydrogenase E1 component subunit beta, mitochondrial
PGK\textsubscript{YEAST} Phosphoglycerate kinase
ALDH\textsubscript{5}YEAST Aldehyde dehydrogenase 5, mitochondrial
DLDH\textsubscript{YEAST} Dihydrolipoyl dehydrogenase, mitochondrial
ENO\textsubscript{1}YEAST Enolase 1
G3P\textsubscript{1}YEAST Glyceraldehyde–3–phosphate dehydrogenase 1
G3P\textsubscript{2}YEAST Glyceraldehyde–3–phosphate dehydrogenase 2
PGM\textsubscript{1}YEAST Phosphoglucomutase –1
ACS\textsubscript{2}YEAST Acetyl–coenzyme A synthetase 2
ALDH\textsubscript{5}YEAST Aldehyde dehydrogenase 5, mitochondrial
ODP\textsubscript{2}YEAST Dihydrolipoyllysine–residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial

MPG\textsubscript{1}YEAST Mannose–1–phosphate guanyltransferase
interacts with

KPYK\textsubscript{1}YEAST Pyruvate kinase 1
THI\textsubscript{3}YEAST Thiamine metabolism regulatory protein THI3
ALDH\textsubscript{5}YEAST Aldehyde dehydrogenase 5, mitochondrial
ODP\textsubscript{2}YEAST Pyruvate dehydrogenase E1 component subunit alpha, mitochondrial
DLDH\textsubscript{YEAST} Dihydrolipoyl dehydrogenase, mitochondrial
PGM\textsubscript{1}YEAST Phosphoglucomutase –1
ACS\textsubscript{2}YEAST Acetyl–coenzyme A synthetase 2
ALDH\textsubscript{5}YEAST Aldehyde dehydrogenase 5, mitochondrial
ODP\textsubscript{2}YEAST Dihydrolipoyllysine–residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial
KPYK\textsubscript{2}YEAST Pyruvate kinase 2
ALDH\textsubscript{4}YEAST Potassium–activated aldehyde dehydrogenase, mitochondrial
D.2 A workflow to annotate metabolic pathways with PPIs

<table>
<thead>
<tr>
<th>Gene Symbol</th>
<th>Gene Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN5_YEAST</td>
<td>Histone acetyltransferase GCN5</td>
</tr>
<tr>
<td>PGK_YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>G3P1_YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>K6PF1_YEAST</td>
<td>6-phosphofructokinase subunit alpha</td>
</tr>
<tr>
<td>ENO2_YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>G3P1_YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 1</td>
</tr>
<tr>
<td>G3P2_YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 2</td>
</tr>
<tr>
<td>ALF_YEAST</td>
<td>Fructose-bisphosphate aldolase</td>
</tr>
<tr>
<td>ACS2_YEAST</td>
<td>Acetyl-coenzyme A synthetase 2</td>
</tr>
<tr>
<td>ADH3_YEAST</td>
<td>Alcohol dehydrogenase 3, mitochondrial</td>
</tr>
<tr>
<td>K6PF2_YEAST</td>
<td>6-phosphofructokinase subunit beta</td>
</tr>
<tr>
<td>ADH2_YEAST</td>
<td>Alcohol dehydrogenase 2</td>
</tr>
<tr>
<td>PYR1_YEAST</td>
<td>Protein URA1</td>
</tr>
<tr>
<td>ODP2_YEAST</td>
<td>Pyruvate dehydrogenase E1 component subunit beta, mitochondrial</td>
</tr>
<tr>
<td>THI3_YEAST</td>
<td>Thiamine metabolism regulatory protein THI3</td>
</tr>
<tr>
<td>ALDH5_YEAST</td>
<td>Aldehyde dehydrogenase 5, mitochondrial</td>
</tr>
<tr>
<td>ODP2_YEAST</td>
<td>Pyruvate dehydrogenase E1 component subunit alpha, mitochondrial</td>
</tr>
<tr>
<td>PGML_YEAST</td>
<td>Phosphoglucomutase-1</td>
</tr>
<tr>
<td>ACS2_YEAST</td>
<td>Acetyl-coenzyme A synthetase 2</td>
</tr>
<tr>
<td>ADH3_YEAST</td>
<td>Alcohol dehydrogenase 3, mitochondrial</td>
</tr>
<tr>
<td>ODP2_YEAST</td>
<td>Dihydrolipoyllysine-residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial</td>
</tr>
<tr>
<td>KPYK2_YEAST</td>
<td>Pyruvate kinase 2</td>
</tr>
<tr>
<td>ALDH4_YEAST</td>
<td>Potassium-activated aldehyde dehydrogenase, mitochondrial</td>
</tr>
<tr>
<td>EF1A_YEAST</td>
<td>Elongation factor 1-alpha</td>
</tr>
</tbody>
</table>

EF1A_YEAST interacts with:

<table>
<thead>
<tr>
<th>Gene Symbol</th>
<th>Gene Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODP2_YEAST</td>
<td>Pyruvate dehydrogenase E1 component subunit beta, mitochondrial</td>
</tr>
<tr>
<td>THI3_YEAST</td>
<td>Thiamine metabolism regulatory protein THI3</td>
</tr>
<tr>
<td>ALDH5_YEAST</td>
<td>Aldehyde dehydrogenase 5, mitochondrial</td>
</tr>
<tr>
<td>DLDH_YEAST</td>
<td>Dihydrolipoyl dehydrogenase, mitochondrial</td>
</tr>
<tr>
<td>G3P2_YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 2</td>
</tr>
<tr>
<td>ACS2_YEAST</td>
<td>Acetyl-coenzyme A synthetase 2</td>
</tr>
<tr>
<td>ADH3_YEAST</td>
<td>Alcohol dehydrogenase 3, mitochondrial</td>
</tr>
<tr>
<td>K6PF2_YEAST</td>
<td>6-phosphofructokinase subunit beta</td>
</tr>
</tbody>
</table>
### D.2 A workflow to annotate metabolic pathways with PPIs

<table>
<thead>
<tr>
<th>Gene ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODP2_YEAST</td>
<td>Dihydrolipoyllysine–residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial</td>
</tr>
<tr>
<td>KPYK2_YEAST</td>
<td>Pyruvate kinase 2</td>
</tr>
<tr>
<td>YD161_YEAST</td>
<td>UPF0661 TPR repeat–containing protein YDR161W</td>
</tr>
</tbody>
</table>

**interacts with**

<table>
<thead>
<tr>
<th>Gene ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPYK1_YEAST</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>PGK_YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>TPI5_YEAST</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>G3P4_YEAST</td>
<td>Glyceraldehyde–3–phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ENO2_YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>ALF_YEAST</td>
<td>Fructose–bisphosphate aldolase</td>
</tr>
<tr>
<td>PMG1_YEAST</td>
<td>Phosphoglycerate mutase</td>
</tr>
<tr>
<td>PDC1_YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ADH1_YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
<tr>
<td>PSF2_YEAST</td>
<td>DNA replication complex GINS protein PSF2</td>
</tr>
</tbody>
</table>

**interacts with**

<table>
<thead>
<tr>
<th>Gene ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G6P1_YEAST</td>
<td>Glucose–6–phosphate isomerase</td>
</tr>
<tr>
<td>TPI5_YEAST</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>PDC1_YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>K6PF1_YEAST</td>
<td>6–phosphofructokinase subunit alpha</td>
</tr>
<tr>
<td>ENO1_YEAST</td>
<td>Enolase 1</td>
</tr>
<tr>
<td>ALF_YEAST</td>
<td>Fructose–bisphosphate aldolase</td>
</tr>
<tr>
<td>PMG1_YEAST</td>
<td>Phosphoglycerate mutase</td>
</tr>
<tr>
<td>KPYK2_YEAST</td>
<td>Pyruvate kinase 2</td>
</tr>
</tbody>
</table>

**interacts with**

<table>
<thead>
<tr>
<th>Gene ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THI3_YEAST</td>
<td>Thiamine metabolism regulatory protein THI3</td>
</tr>
<tr>
<td>ALDH5_YEAST</td>
<td>Aldehyde dehydrogenase 5, mitochondrial</td>
</tr>
<tr>
<td>ACS2_YEAST</td>
<td>Acetyl–coenzyme A synthetase 2</td>
</tr>
<tr>
<td>ADH3_YEAST</td>
<td>Alcohol dehydrogenase 3, mitochondrial</td>
</tr>
<tr>
<td>ALDH2_YEAST</td>
<td>Aldehyde dehydrogenase [NAD(P)+] 1</td>
</tr>
<tr>
<td>ODP2_YEAST</td>
<td>Dihydrolipoyllysine–residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial</td>
</tr>
<tr>
<td>KPYK2_YEAST</td>
<td>Pyruvate kinase 2</td>
</tr>
<tr>
<td>ALDH4_YEAST</td>
<td>Potassium–activated aldehyde dehydrogenase, mitochondrial</td>
</tr>
<tr>
<td>6PGD1_YEAST</td>
<td>6–phosphogluconate dehydrogenase, decarboxylating 1</td>
</tr>
</tbody>
</table>
### D.2 A workflow to annotate metabolic pathways with PPIs

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPYK1YEAST</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>PGK1YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>ENO1YEAST</td>
<td>Enolase 1</td>
</tr>
<tr>
<td>ENO2YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>G3P1YEAST</td>
<td>Glyceraldehyde–3–phosphate dehydrogenase 1</td>
</tr>
<tr>
<td>ALF1YEAST</td>
<td>Fructose–bisphosphate aldolase</td>
</tr>
<tr>
<td>PDC1YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ADH1YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP47YEAST</td>
<td>Protein EMP47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGK1YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>G3P3YEAST</td>
<td>Glyceraldehyde–3–phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ENO2YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>ALF2YEAST</td>
<td>Fructose–bisphosphate aldolase</td>
</tr>
<tr>
<td>PMG1YEAST</td>
<td>Phosphoglycerate mutase 1</td>
</tr>
<tr>
<td>PDC1YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ADH1YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVP23YEAST</td>
<td>Golgi apparatus membrane protein TVP23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGK1YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>G3P3YEAST</td>
<td>Glyceraldehyde–3–phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ENO2YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>ALF2YEAST</td>
<td>Fructose–bisphosphate aldolase</td>
</tr>
<tr>
<td>PMG1YEAST</td>
<td>Phosphoglycerate mutase 1</td>
</tr>
<tr>
<td>PDC1YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ADH1YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAR11YEAST</td>
<td>Factor arrest protein 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPYK1YEAST</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>TPISYEAST</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>ENO2YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>ALF1YEAST</td>
<td>Fructose–bisphosphate aldolase</td>
</tr>
<tr>
<td>PMG1YEAST</td>
<td>Phosphoglycerate mutase 1</td>
</tr>
<tr>
<td>PDC1YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ADH1YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Interacts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTE1YEAST</td>
<td>Guanine nucleotide exchange factor LTE1</td>
</tr>
</tbody>
</table>
## D.2 A workflow to annotate metabolic pathways with PPIs

<table>
<thead>
<tr>
<th>Interacts With</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HXKG&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Glucokinase-1</td>
</tr>
<tr>
<td>PGK&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>TPIS&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>DLDH&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Dihydrolipoamide dehydrogenase, mitochondrial</td>
</tr>
<tr>
<td>ENO1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Enolase 1</td>
</tr>
<tr>
<td>ENO2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>PDC1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>TBA1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Tubulin alpha-1 chain</td>
</tr>
</tbody>
</table>

Interacts with

<table>
<thead>
<tr>
<th>Interacts With</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALDH5&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Aldehyde dehydrogenase 5, mitochondrial</td>
</tr>
<tr>
<td>G3P3&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ACS2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Acetyl-coenzyme A synthetase 2</td>
</tr>
<tr>
<td>ADH3&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Alcohol dehydrogenase 3, mitochondrial</td>
</tr>
<tr>
<td>ALDH2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Aldehyde dehydrogenase [NAD(P)+] 1</td>
</tr>
<tr>
<td>ODP2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Dihydrolipoamide-residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial</td>
</tr>
<tr>
<td>KPYK2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Pyruvate kinase 2</td>
</tr>
<tr>
<td>DSN1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Kinetochore-associated protein DSN1</td>
</tr>
</tbody>
</table>

Interacts with

<table>
<thead>
<tr>
<th>Interacts With</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPYK1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>G6P&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Glucose-6-phosphate isomerase</td>
</tr>
<tr>
<td>TPIS&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>K6PF1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>6-phosphofructokinase subunit alpha</td>
</tr>
<tr>
<td>ENO1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Enolase 1</td>
</tr>
<tr>
<td>ALF&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Fructose-bisphosphate aldolase</td>
</tr>
<tr>
<td>PMG1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Phosphoglycerate mutase 1</td>
</tr>
<tr>
<td>NCE2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Non-classical export protein 2</td>
</tr>
</tbody>
</table>

Interacts with

<table>
<thead>
<tr>
<th>Interacts With</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPYK1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>PGK&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>G3P3&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ENO2&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>ALF&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Fructose-bisphosphate aldolase</td>
</tr>
<tr>
<td>PDC1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ADH1&lt;sub&gt;YEAST&lt;/sub&gt;</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
</tbody>
</table>
### D.2 A workflow to annotate metabolic pathways with PPIs

<table>
<thead>
<tr>
<th>Yeast Protein</th>
<th>Enzyme/Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSP75, YEAST</td>
<td>Heat shock protein SSB1</td>
</tr>
<tr>
<td>ODP2, YEAST</td>
<td>Pyruvate dehydrogenase E1 component subunit beta, mitochondrial</td>
</tr>
<tr>
<td>PGK, YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>DLH2, YEAST</td>
<td>Dihydrolipoyl dehydrogenase, mitochondrial</td>
</tr>
<tr>
<td>ENO1, YEAST</td>
<td>Enolase 1</td>
</tr>
<tr>
<td>ACS2, YEAST</td>
<td>Acetyl-coenzyme A synthetase 2</td>
</tr>
<tr>
<td>ADH3, YEAST</td>
<td>Alcohol dehydrogenase 3, mitochondrial</td>
</tr>
<tr>
<td>ALDH4, YEAST</td>
<td>Potassium-activated aldehyde dehydrogenase, mitochondrial</td>
</tr>
<tr>
<td>YMB8, YEAST</td>
<td>Uncharacterized vacuolar membrane protein YML018C</td>
</tr>
<tr>
<td>G6P, YEAST</td>
<td>Glucose-6-phosphate isomerase</td>
</tr>
<tr>
<td>G3P3, YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ENO2, YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>PDC1, YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ODP2, YEAST</td>
<td>Dihydrolipoyllysine-residue acetyltransferase component of pyruvate dehydrogenase complex, mitochondrial</td>
</tr>
<tr>
<td>ADH1, YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
<tr>
<td>BUB2, YEAST</td>
<td>Mitotic check point protein BUB2</td>
</tr>
<tr>
<td>KPYK1, YEAST</td>
<td>Pyruvate kinase 1</td>
</tr>
<tr>
<td>PGK, YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>G3P3, YEAST</td>
<td>Glyceraldehyde-3-phosphate dehydrogenase 3</td>
</tr>
<tr>
<td>ENO2, YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>ALF, YEAST</td>
<td>Fructose-bisphosphate aldolase</td>
</tr>
<tr>
<td>ADH1, YEAST</td>
<td>Alcohol dehydrogenase 1</td>
</tr>
<tr>
<td>ALF, YEAST</td>
<td>Fructose-bisphosphate aldolase</td>
</tr>
<tr>
<td>PGK, YEAST</td>
<td>Phosphoglycerate kinase</td>
</tr>
<tr>
<td>TPIS, YEAST</td>
<td>Triosephosphate isomerase</td>
</tr>
<tr>
<td>ENO2, YEAST</td>
<td>Enolase 2</td>
</tr>
<tr>
<td>PMG1, YEAST</td>
<td>Phosphoglycerate mutase 1</td>
</tr>
<tr>
<td>PDC1, YEAST</td>
<td>Pyruvate decarboxylase isozyme 1</td>
</tr>
<tr>
<td>ACS2, YEAST</td>
<td>Acetyl-coenzyme A synthetase 2</td>
</tr>
</tbody>
</table>
Appendix E

Cyclic cores

The following page shows two Palsson human metabolic network layouts, the original layout (Figure E.1), and then the ‘cyclic’ core (Figure E.2). As discussed in the Conclusion, a very large proportion of enzymes and metabolites are involved in at least one cyclic process.
Figure E.1: The Palsson human network before cyclic core analysis, visualised using the spring-embedded layout in Cytoscape. This diagram shows all nodes and edges.

Figure E.2: The Palsson human network after cyclic core analysis, visualised using the spring-embedded layout in Cytoscape. This diagram contains only those nodes and edges involved in at least one cycle.