Biochemical Network Matching and Composition

Martin Hugh Goodfellow
Advisors: John Wilson and Ela Hunt
University of Strathclyde
26 Richmond Street
Glasgow, Scotland
martin.goodfellow@cis.strath.ac.uk

ABSTRACT

Graph composition has applications in a variety of practical applications. In drug development, for instance, in order to understand possible drug interactions, one has to merge known networks and examine topological variants arising from such composition. Similarly, the design of sensor nets may use existing network infrastructures, and the superpositon of one network on another can help with network design and optimisation. The problem of network composition has not received much attention in algorithm and database research. Here, we work with biological networks encoded in Systems Biology Markup Language (SBML), based on XML syntax. We focus on XML merging and examine the algorithmic and performance challenges we encountered in our work and the possible solutions to the graph merge problem. We show that our XML graph merge solution performs well in practice and improves on the existing toolsets. This leads us into future work directions and the plan of research which will aim to implement graph merging primitives in a database engine.

1. INTRODUCTION

Graphs are of growing importance in our world, in politics, academia and in industry. They allow powerful abstractions of interactions occurring in the global economy, scientific models of nature and society, and of systems composed of engineered and natural components. As these graphs are generally dynamic they will continuously increase and decrease in size. To effectively model these networks, using a graph representation, methods for composition are required.

Power engineering, environmental sensor networks, drug target discovery, policy development, web security, traffic modelling and health applications, are just a few application areas. The resulting graphs can be very large and need efficient handling. To this aim we propose to work on algorithmic and indexing techniques in network modelling so that large networks become tractable.

We focus on biochemical networks. As network representations in XML are available (in SBML, see www.ebi.ac.uk/biomodels-main/), they provide an excellent training ground for the development of new algorithms and database methods for network reorganisation. A biochemical network consists of chemical reactions between species (reactants and products), and is enacted in a living organism. A simple network is shown in Figure 1(a). This network contains three chemical species, A, B and C, three reactions, $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow B$, and three rate constants, $k1$, $k2$ and $k3$, which define the speed of the reactions. The main motivation behind composition is to support the engineering of biochemical networks by allowing them to be easily designed, analysed, and modified.

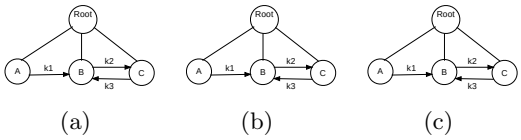


Figure 1: Merging two identical models ($A \rightarrow B \leftrightarrow C$): $a + b = c$

There are several different types of biochemical networks or pathways, including transcription, signal transduction and metabolic pathways. Biochemical network models can be rule-based, Petri nets, XML based or expressed as systems of equations. Models are used for simulation, analysis and drug target prediction. They are simulated in order to determine how a biochemical network will behave over a given time interval. Additionally, models can be analysed to discover interesting behaviour(s) they exhibit. They can also be used to predict the effect of drugs on the biochemical network they represent. This enables researchers to estimate the effectiveness of a drug developed for a specific purpose. It also allows drugs to be tested in silico without the need for human or animal test subjects.

Model composition is of high importance. Many biochemical network specifications are made up entirely or partly of existing network models. Some modellers manually merge existing models when building new models, which is time consuming, frustrating and error prone. Composition also allows models to be created from libraries or databases of standard parts. Additionally, composition can be used as the first step in automatically varying topologies to perform model identification. In formal terms, this involves finding a model which fits specific data. Finally, composition can also be used for merging models of pathways which interact,

EDBT 2010, March 22-26, 2010, Lausanne, Switzerland.

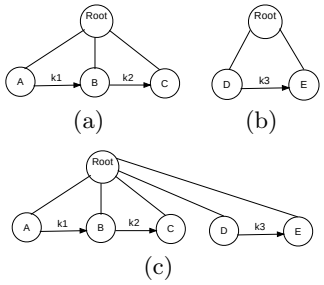


Figure 2: Merging two different models ($A \rightarrow B \rightarrow C$ and $D \rightarrow E$): $a + b = c$

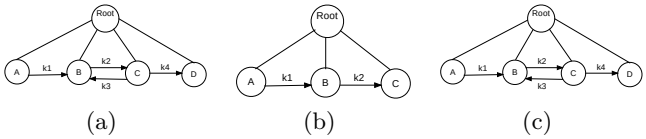


Figure 3: Merging models with shared reactions/species ($A \rightarrow B \leftrightarrow C \rightarrow D$ and $A \rightarrow B \rightarrow C$): $a + b = c$

where interact means they have shared entities or species.

SBML models (see www.systems-biology.org) can be created and simulated using interactive tools, including CellDesigner (celldesigner.org), and visualised using packages such as Cytoscape (www.cytoscape.org). An existing package for model merging, semanticSBML, sysbio.molgen.mpg.de/semanticsbml, allows for interactive model merging but requires a very significant amount of user interaction, which makes it unsuitable for automated integration of pathways. To remedy this shortcoming, and enable automated merge for large scale models we developed SBMLCompose which performs unsupervised model composition.

Our CONTRIBUTIONS are as follows. We designed and implemented SBMLCompose which performs automated model merging. This solves some problems of XML graph merging and of math expression comparison, synonymy, and unit conversion which are parts of the problem space. We benchmarked the performance of SBMLCompose against semanticSBML and show that SBMLCompose is an order of magnitude faster. We propose new research directions which will enable automated composition of large models in the future and support large-scale model merging.

The remainder of this paper is structured as follows. Section 2 explains our notation and summarises related work. Section 3 describes SBMLCompose and Section 4 presents results which show that our method is faster than semanticSBML. Finally, in Section 5 we resent future directions and conclusions.

2. PRELIMINARIES

Three examples of network merging are shown in Figures 1, 2, and 3. Intuitively, where models are identical, the result is the same as either of the models. When models share a node, two identical nodes are merged into one, and when models share an edge with its two adjacent nodes, those two identical nodes and the edge are merged. Matching here is different from matching as defined in graph theory. Matching here is defined as a matching of both edges and nodes. I define composition and decomposition as follows:

A graph G is denoted as (V, E, L, ϕ, ψ), where V is the set of nodes, $E \subseteq V$ x V is the set of (directed or undirected) edges and L is a set of labels, i.e. identifiers or math expressions. Labels are specified by mappings $\phi : V \rightarrow \Sigma_L$ and $\psi : E \rightarrow \Sigma_L$. Let G_1 and G_2 be two graphs. Graph composition is the union of the graphs, $G_1 \cup G_2$ with (potentially) shared nodes or shared nodes and unitable edges. Node and edge comparison is based on the comparison of labels. Two nodes n1 ϵ G_1 and n2 ϵ G_2 are equal iff their labels are identical or synonymous, i.e. $\phi(n1) \approx \phi(n2)$. Two edges are equivalent iff their labels can be united via an arithmetic operation, i.e. $\psi((n1, n2))\epsilon G_1$ is unitable with $\psi((n3, n4))\epsilon G_2$. The rules for edge comparison and unification follow the laws of chemistry, as explained later.

Graph composition is currently a relatively untouched research area. Previous work includes determining how many different compositions will produce the given graph [15, 11, 3] and finding the optimal topology for an automata network by artificial evolution [17], which requires the use of composition methods. However, these methods are specific to automata and not directly usable in our work. Work has also been performed on composition of perfect graphs [6].

Biochemical network composition is a relatively untouched area. However, composition exists in other, related contexts. We discuss here textual composition and XML composition. The simplest form of composition is TEXTUAL COMPOSITION, such as performed by the Unix utilities *diff* and *patch*. *Diff* finds the differences between two text files and *patch* uses those to compose the files. This composition is automated. Alternatively, *sdiff* can be used to interactively perform the composition of two text files. Files are equal if all lines are equal. *Patch* assigns the first file to be the composed file and makes the changes within it to make it match the other file. *Sdiff* behaves similarly but the user chooses what changes are accepted. *Diff* uses the Longest Common Subsequence (LCS) algorithm [18, 19]. LCS solutions include the use of indexes, such as persistent suffix trees, or online solutions via the Smith-Waterman algorithm [21] used in computational biology and plagiarism detection [13]. Global and local alignment in biology correspond to linear text merging. Textual comparison is also widely used in version control, collaborative editing [12] and the comparison of and updating of documents and software.

XML COMPOSITION is more complex as it must take account of the XML structure, which corresponds to the tree to tree correction problem [22]. This is similar to the string to string correction problem [24] and involves calculating the minimum number of edits between two trees where the edits are replacing a node with another node, removing a node and inserting a node. This can be solved in two ways, using ordered or unordered tree model algorithms, see [7] and [25]. Available tools in this area include DeltaXML [27] and xmldiff [28].

Tree edit problem and composition arise also in software engineering and can be solved using a version of the LCS algorithm which is similar but not identical to the one used in diff. Tree based comparisons of programs have been proposed in [16] and are similar to XML tree comparison.

The most relevant applied work in model composition

is SBMLMerge, part of semanticSBML. The composition method first annotates the elements in the model with identifiers from biological model databases to allow the meaning of each element to be known. This involves database lookups which are slow and do not scale up. The next stage involves checking the semantic validity of the models to be composed, to ensure only valid models are merged. Finally, the models are merged to create the composed model. SBMLmerge first partitions the attributes of each SBML component into identifying attributes and describing attributes. It then combines all the components from each model into one model and parses this new model to remove all identical/conflicting components. Components are identified as identical if the identifying attributes are the same as well as all the describing attributes, otherwise they are different. Components are identified as conflicting if the inclusion of both of them goes against the semantic rules of SBML. However, semanticSBML has some disadvantages as it only works with SBML models from certain databases and not with any desired model. It doesn't annotate models automatically, as it requires user interaction. Once the annotations have been added, the composition is still not automated, as for example, the software cannot determine if the maths of initial assignments are equal. Users have to decide what initial assignment is included in the model, should two exist for the same entity. Additionally, should a group of modelers be creating a large new model for which not all of the elements have been defined, it is not possible for the model to be built incrementally. Finally, several passes over the source XML are required, which is inefficient.

3. SBMLCOMPOSE

Our new method, SBMLCompose, performs composition of biochemical networks.

We applied textual XML merging to SBML models which consist of various types of components, such as function definitions, unit definitions, species types, units, reactions, and others. Our algorithm checks each component type to see if two different components are identical. Depending on the component type, this check varies from relatively simple to quite complex. For example, for each pair of species components the names and identifiers were checked to see if they were identical or synonymous. The comparison algorithm is shown in Figures 4 and 5. The algorithm in Figure 4 details the execution of the complete composition process. Figure 5 shows the generic merge algorithm that is called at each stage of the algorithm in Figure 4. The merge algorithm varies for each element with respect to both the equality and composition step.

Currently the indexing structure mentioned in line 5 is a hash map. A hash map exists for each component contained in an SBML model. These indexes use a string as the key. The string used is determined by the SBML components the index is utilised for. In the majority of cases this is the ID or the name, if this is available. This index structure will be the subject of future research. We hope to determine which is the best index for this scenario and other XML-based graphs.

One of the challenges here is the existence of arbitrary names and synonymy. To overcome this, we use synonym tables and the users who create models are informed that biological entities must be given names expressing biological meaning. Identical components are identified via various methods. If the component is named, its name or id is checked for equality. All parameters in the original models have to be included in the composed model, as there is no way of confirming whether they are intended to be equal or not. However, if two parameters have the same name, then one is renamed to avoid conflicts. For all other components, we compare the underlying information, which is the maths in all other cases except for unit definitions. Unit definitions are compared by checking the list of known units. As identical components often have different names, mappings are stored to reduce comparison time, for example, if the kinetic law of a reaction was shared by two models, we stored its

```
   Data: Two model objects to be composed
   Result: The composed model object
 1 Compose Function Definitions;
 2 Compose Unit Definitions;
 3 Compose Compartment Types;
 4 Compose Species Types;
 5 Compose Compartments;
 6 Compose Species;
 7 Compose Parameters;
 8 Compose Rules;
 9 Compose Contraints;
10 Compose Reactions;
11 Compose Events;
12 return composed model
```

Figure 4: Model composition algorithm

```
   input  : Two SBML models
   output: The composed model
 1 if one SBML model empty then
 2 |  return the non-empty model
 3 else if both models are non empty then
 4 |  for each SBML component S2 in second model do
 5 |  |  S1 = Look for SBML component S2 in index of
   |  |  first model;
 6 |  |  if S2 != null then
 7 |  |  |  duplicate(S1,S2):= true;
 8 |  |  |  check for conflicts;
 9 |  |  end
10 |  |  if !duplicate then
11 |  |  |  if S2 = S1 then
12 |  |  |  |  S2:= S1 (rename);
13 |  |  |  |  add mapping;
14 |  |  |  end
15 |  |  |  check for conflicts;
16 |  |  |  add S2 and its data to first model;
17 |  |  else
18 |  |  |  if S1 id != S2 id then
19 |  |  |  |  add mapping;
20 |  |  |  end
21 |  |  |  duplicate(S1,S2):= false;
22 |  |  end
23 |  end
24 end
25 return First SBML model
```

Figure 5: Component Merge Algorithm

Zeroth order: $0 \rightarrow X$
First order: $X \rightarrow ?$
Second order: $X + Y \rightarrow ?$

Here, n_A denotes Avogadro's constant - the number of molecules in one mole of a substance - $n_A = 6.022\text{x}10^{23}$; Ms^{-1} is molecules per second; and k is the reaction rate constant.

Zeroth order: the number of moles is kMs^{-1}, so for a volume V, X is produced at a rate of n_AkV molecules/s. The number of molecules is c/s $c = n_AkV$.

First order: the number of moles is $k[X]Ms^{-1}$. As this involves [X], we need to know that for a volume V, a concentration of [X] corresponds to $x = n_A[X]V$ molecules. Since X decreases at rate $n_Ak[X]V = kx$ molecules/s, the number of molecules is cx/s $c = k$.

Second order: the number of moles is $k[X][Y]Ms^{-1}$. Here, for a volume V, the reaction proceeds at a rate of $n_Ak[X][Y]V = kxy/n_AV$ molecules/s. The number of molecules is cxy/s $c = \frac{k}{n_AV}$.

Figure 6: Converting moles into molecules [26]

mapping for future use.

Another challenge is how to handle inconsistencies/ conflicts between identical components. The default is to issue a warning when a conflict is discovered. The software then includes the first component in the model and writes a warning to a log file informing the user of this and of decisions taken. A significant problem encountered during conflict checking was that values in different models may be defined using different units. The equations used for the conversion of units in the species defining the model are shown in Figure 6. Similarly to unit conversions, conflicts in rate constants and stochiometry within reactions are resolved.

The most complex problem we offer a novel solution for was determining if the maths functions, equations and assignments within the models were equivalent. All maths within a model is described using MathML [2]. However, equations and assignments, even if they are not identical, may be equivalent, due to commutativity. Our solution involves pattern matching. The first algorithm is shown in Figure 7. The algorithm extracts a MathML pattern. This pattern takes commutative operators into account so that it will match commutative maths functions, equations or assignments, regardless of the order of the operands. The pattern is then used to determine if the component containing maths is unique in the context of all the models. The maths is extracted by an algorithm which mirrors the one shown in Figure 7 (not shown).

The kinetic law within a reaction can be expressed in two ways, either as mass action - see Figures 10 and 11 - or Michaelis-Menten kinetics - see Figure 12. To automate the processing of both description types, the formulae are compared using the algorithms just described. Additionally, the reactants, modifiers and products are checked for equality.

The final problem encountered was checking that the initial values of component attributes were the same. These values can be set in different places in the model, within

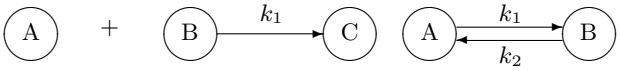
```
getMaths(N, M, P)
input  : current math node N; the maths string M;
         prefix P of current math node N
output: The maths
Data: Node (string value of a node)
Data: K: Number of children (in N)
1  if K > 0 then
2  |  Node = string value of current math tree node
   |  (after applying mappings) ;
3  |  M:= M + P + Node ;
4  |  if N is a commutative operator then
5  |  |  for each child C of N do
6  |  |  |  M = getMaths(C, M, "")
7  |  |  end
8  |  end
9  |  else
10 |  |  for each child C of N do
11 |  |  |  M = getMaths(C, M, (C + child number))
12 |  |  end
13 |  end
14 else
15 |  Node = string value of current math tree node
   |  (after applying mappings) ;
16 |  M:= M + P + Node ;
17 |  Convert M to maths pattern ;
18 |  add pattern to the list of maths patterns ;
19 end
20 return The maths string M
```

Figure 7: Get Maths Patterns Algorithm

Mass Action Kinetics of reaction = -$k_1[A]$

Mass Action Kinetics of reaction = -$k_2[B]$

Figure 10: Mass Action Example 1



Mass Action Kinetics of reaction = -$k_1[A][B]$

Mass Action Kinetics of reaction = $k_1[A] - k_2[B]$

Figure 11: Mass Action Example 2

an initial assignment component and in the definition of the component itself. To solve this, the initial values of all component attributes are collected before composition begins. If a component has an initial assignment, it is extracted and evaluated and the value is saved. The maths is extracted as previously described. The initial values are then used in the check for conflicts during model composition.

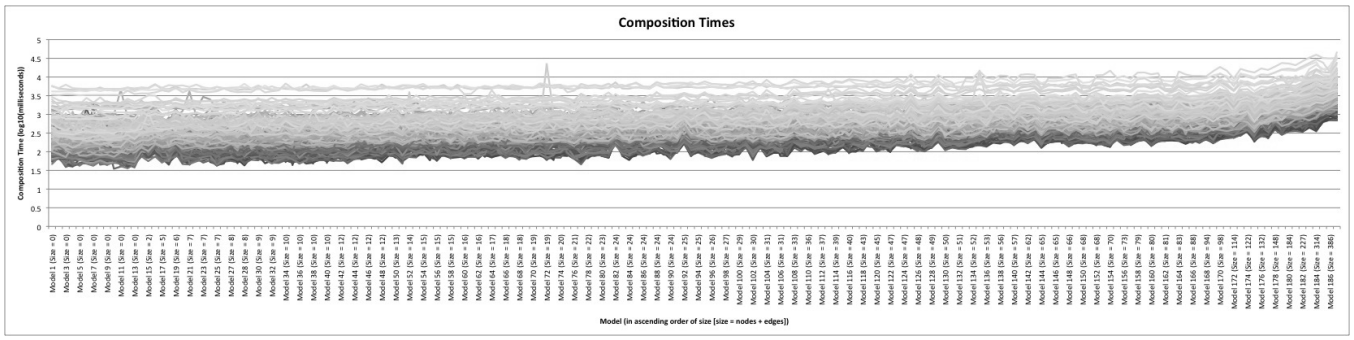


Figure 8: The log_{10}(time in ms) required to compose each model with every other model in order of size (size = nodes + edges) using our method, SBMLCompose

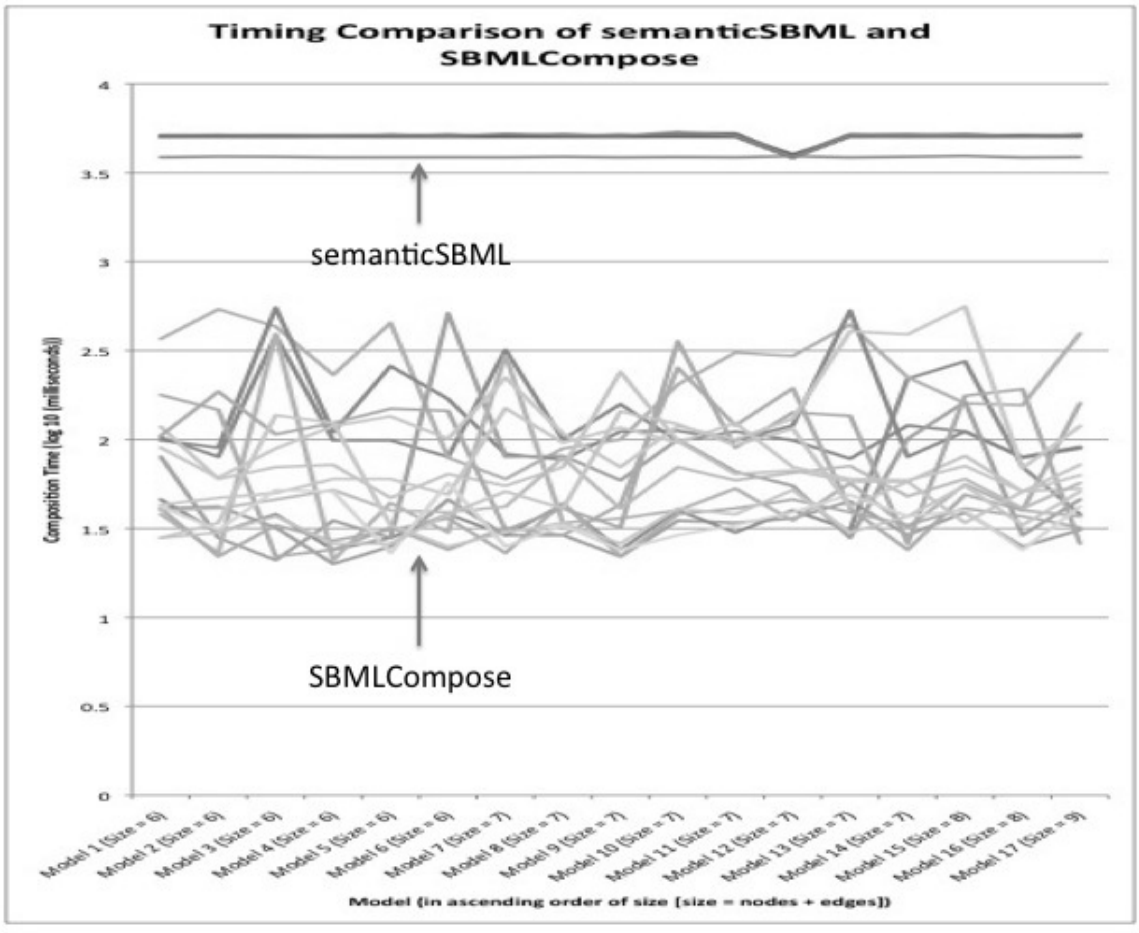


Figure 9: log_{10}(composition time in ms) for semanticSBML and SBMLCompose for each model with every other model in ascending order of size

$$V = V_{max}\frac{[A]}{(K_M + [A])};\quad k_{cat} = \frac{V_{max}}{[E_T]}$$

$$\frac{d[A]}{dt} = -\frac{d[B]}{dt} = -k_{cat}[E_T]\frac{[A]}{(K_M + [A])}$$

V: Reaction velocity
K_M: Michaelis constant - concentration of substrate at which reaction rate is half max value
$[E_T]$: Total enzyme concentration

Figure 12: Michaelis-Menten equations

4. RESULTS

SBMLCompose was developed in Java 1.6. Beanshell [4] was utilised to allow Java maths strings to be executed as code. The software performance was tested on Linux kubunto version 2.6 with 1GB RAM on a 2.60GHz Intel Pentium 4 CPU.

The models were sourced from the BioModels database - 187 models. Model size ranged from 0 to 194 nodes and 0 to 313 edges. Each of the models was composed with every other model using our method, SBMLCompose, and the composition time recorded. Each composition involved only two models and the composition testing proceeded as follows: the smallest model was composed with the smallest model, the smallest model was composed with the second smallest model, ... , the largest model was composed with the second largest model, the largest model was composed with the largest model. The results are summarised in Figure 8. Composition has $O(nm)$ time complexity for two models of sizes n and m.

Only 17 test models which can be fully parsed are provided with semanticSBML, with all models already annotated biologically and requiring a local database lookup. The size of these models ranges from 4 to 7 nodes and 0 to 3 edges. Each of these models was composed with every other model in the collection and the composition time recorded for both semanticSBML and SBMLCompose. The composition was performed the same way as the previous experiment. The results of this experiment are shown in Figure 9. SBMLCompose is at least an order of magnitude faster than semanticSBML, and this is visible even for small models. The complexity of semanticSBML is $O(mn)$, with several passes over the data. One possible reason for SBMLCompose's better performance is that for each run of semanticSBML, a local database is loaded consisting of 54,929 entries from Gene Ontology [1], KEGG Compound [14], ChEBI [8], PubChem (pubchem.ncbi.nlm.nih.gov), 3DMET (www.3dmet.dna.affrc.go.jp) and CAS (www.cas.org). During the composition process this database is consulted to resolve similarities/dissimilarities by identifying the components within it and assigning to them the unique id, for that component, contained within the database. In contrast, SBMLCompose does not require any database accesses to perform composition, as it uses local synonym tables instead. All equality methods are based on the structure of the SBML. Our synonym tables are smaller and contain only the entries required for the composition. This also ensures that new biological entities can be added to support composition, as needed.

4.1 Evaluation of Test Results

Evaluation involved pairwise comparison, comparing the merged model with the expected model, and checking quality by eye. The methods used for comparison are discussed in the following sections.

4.1.1 Textual Comparison of SBML

A simple but tedious method involved the textual comparison of SBML by examining the resulting SBML model and the expected SBML model. This was performed manually as available XML differencing utilities treated the order of XML components as either important or unimportant. However for SBML the order of components is relevant in some cases but irrelevant in others. This comparison was feasible for small models. However, this would be impractical for larger models as some had in excess of 400 lines of SBML. Thus, textual comparison was used when practical. Therefore other forms of evaluation were required.

4.1.2 Visual Comparison of Simulations

The simplest method of evaluating test results was the visual comparison of simulations. This involved simulating the expected output and the actual output. The graphs of these simulations were then compared to confirm correctness. This method was however crude and inaccurate. Similar simulation graphs are not always identical. Subtle differences between concentrations of species or behaviour could go unnoticed. Therefore, more precise forms of evaluation were required, as described further.

4.1.3 Pairwise Comparison of Traces using Residual Sum of Squares

Residual sum of squares over simulation times was also used, as follows. A file of time series data of concentrations for various species was generated. This was then used to calculate the sum of squares between identical species from the two models. The results were used to determine if the models were equivalent - the sum of squares is close to 0 for all identical species.

4.1.4 Model Checking of Properties

The output of the composed model that the software produced was also checked for correctness using a model checker which examined specific model properties, expressed using temporal logic. We then used the Monte Carlo Model Checker (MC2) [9].

5. FUTURE WORK AND CONCLUSIONS

This preliminary investigation leads us to several future directions. A generic graph composition method would be preferable but is it possible to perform efficient and correct composition without semantics? We plan on reducing the semantic reliability of the current SBML method to only require light semantics as opposed to heavy semantics as is currently used. This would allow us to compare the performance and determine how reliant composition is on semantics. This comparison can be further extended by creating a generic method that requires no semantics and comparing it with both the SBML composition method for light and heavy semantics.

Our work plan includes the following types of investigation:

1. Optimising access to locally cached copies of synonym databases

2. Defining a method for XML graph decomposition or splitting

3. Development of new indexing techniques to support network merging and splitting

4. Development of indexes to support zooming in and out of networks and their subparts (indexing and algorithms for semantic graph zooming)

5. New formalisms and notations for graph merging and splitting

6. Algorithmic optimisation of graph operations

7. Complexity reduction by using a suffix tree [10] or a hash table, down to $O(m+n)$, as graph nodes can be indexed while being parsed, and looked up via hash table or suffix tree lookup.

8. Design of graph encodings for other application areas

This exploration opens up a new research area on the boundary of algorithms, databases and bioinformatics. It parallels ongoing work in XML Schema integration [20] and in subgraph identification [23]. A recent database approach in this area involves the work on path matching [5]. We have demonstrated a new approach to graph merging, which uses XML formatted graphs and performs a textual merge. This method, despite being tested with SBML, demonstrates a generic framework for the composition of annotated graphs. The performance of our method appears to be better than the performance of a similar existing package and these preliminary results require further study. What is also novel is that we solve the problem of math composition, which appears to be new in this context.

Part of the development of SBMLCompose took place in the Glasgow Bioinformatics Research Centre led by D. Gilbert, and was part of MG's Honours project. MG is funded by the EPSRC. EH is now at the ETH Zurich.

6. REFERENCES

[1] M. Ashburner et al. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*, 25(1):25–29, May 2000.

[2] R. Ausbrooks et al. Mathematical Markup Language (MathML) v. 2.0. October 2003. http://www.w3.org/TR/MathML2/.

[3] W. Bajguz. Graph and Union of Graphs Compositions, Jan. 31 2006. http://arxiv.org/abs/math/0601755.

[4] BeanShell - Lightweight Scripting for Java. http://www.beanshell.org/.

[5] P. Bouros et al. Evaluating reachability queries over path collections. In *SSDBM*, pages 398–416, 2009.

[6] W. H. C. C. Cornuejols. Compositions for Perfect Graphs. *Discrete Mathematics*, 55:245–254, 1985.

[7] S. S. Chawathe. Comparing Hierarchical Data in External Memory. *VLDB*, pages 90–101, 1999.

[8] K. Degtyarenko et al. ChEBI: A Database and Ontology for Chemical Entities of Biological Interest. *Nucleic Acids Research*, 36(Database-Issue):344–350, 2008.

[9] R. Donaldson and D. Gilbert. A Model Checking Approach to the Parameter Estimation of Biochemical Pathways, CMSB. *LNCS*, 5307:269–287, 2008.

[10] E. Hunt et al. A Database Index to Large Biological Sequences. In *VLDB*, pages 139–48, 2001.

[11] A. Huq. Compositions of Graphs Revisited. *Electr. J. Comb.*, 14(1), 2007.

[12] C.-L. Ignat and M. C. Norrie. Flexible Collaboration over XML Documents. In *CDVE*, pages 267–274, 2006.

[13] R. Irving. Plagiarism and Collusion Detection using the Smith-Waterman Algorithm. Technical report, University of Glasgow, Department of Computing Science, 2004. http://www.dcs.gla.ac.uk/publications/PAPERS/7444-/TR-2004-164.pdf.

[14] M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.*, 28:29–34, 2000.

[15] A. Knopfmacher and M. E. Mays. Graph Compositions I: Basic Enumeration, Feb. 11 2003. http://citeseer.ist.psu.edu/564967.html.

[16] L. Lian et al. Development of Program Difference Tool based on Tree Mapping. *IEICE Trans. Inf. and Systems*, 78(10):1261–1268, 1995.

[17] M. H. Luerssen and D. M. W. Powers. Graph Composition in a Graph Grammar-Based Method for Automata Network Evolution. In *IEEE Congress on Evolutionary Computation*, pages 1653–1660, 2005.

[18] W. Miller and E. W. Myers. A File Comparison Program. *Software - Practice and Experience*, 15(11):1025–1040, 1985.

[19] E. Myers. An $0(nd)$ Difference Algorithm and its Variations. *Algorithmica*, 1(2):251–266, 1986.

[20] K. Saleem et al. PORSCHE: Performance ORiented SCHEma mediation. *Inf. Syst.*, 33(7-8):637–657, 2008.

[21] T. F. Smith and M. S. Waterman. Identification of Common Molecular Sub-Sequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[22] K.-C. Tai. The Tree-to-Tree Correction Problem. *J. ACM*, 26(3):422–433, 1979.

[23] Y. Tian and J. M. Patel. TALE: A Tool for Approximate Large Graph Matching. In *ICDE*, pages 963–72. IEEE, 2008.

[24] R. A. Wagner and M. J. Fischer. The String-to-String Correction Problem. *J. ACM*, 21(1):168–173, 1974.

[25] Y. Wang et al. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *ICDE*, pages 519–530, 2003.

[26] D. J. Wilkinson. *Stochastic Modelling for Systems Biology*. Chapman & Hall, 2006.

[27] DeltaXML. http://www.deltaxml.com/.

[28] xmldiff. www.logilab.org/859.