A graph-based framework for comparing curricula

by
Linda Marshall
Department of Computer Science
University of Pretoria

Submitted in partial fulfillment of the requirements for the degree
Doctor of Philosophy in Information Technology
in the
Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

February 2014

A graph-based framework for comparing curricula

by

Linda Marshall
E-mail: lmarshall@cs.up.ac.za

Abstract

The problem addressed in this thesis was identified in a real life context in which an attempt was made to re-constitute a BSc Computer Science degree programme. The curriculum was modelled on the ACM/IEEE Computing Curriculum of 2001. It was further required to comply with accreditation requirements as defined by ABET's Computing Accreditation Commission. Relying on a spreadsheet, the curriculum was iteratively and manually evaluated against the ACM/IEEE curriculum specification. A need was identified to automate or at least semi-automate this process.

In this thesis a generalisation of the problem is presented. Curricula are modelled as directed graphs (digraphs) in which graph vertices represent curriculum elements such as topics, knowledge areas, knowledge units year-levels or modules. Edges in the graph represent dependencies between these vertices such as belonging to grouping or pre-requisites. The task of curriculum comparison then abstracts to a task of digraph comparison.

A framework, the Graph Comparison Framework, is proposed. The framework comprises of components which are used to guide the digraph comparison process. The so-called Graph Trans-morphism algorithm component is the only component in the framework which is mandatory. The algorithm converts the information from one of the digraphs being compared into the structure of the other. This conversion enables the graphs to be compared as *graph isomorphisms*. All digraphs are modelled as *sets of triples*, making it possible to subtract one digraph from another using the *set minus* operator. The resultant difference sets are used by components defined in the

framework to quantify and visualise the differences.

By modelling curricula as digraphs and applying the framework to the digraphs, it is possible to compare curricula. This application of the framework to a real-world problem forms the applications research part of the thesis. In this part, domain knowledge of curriculum design is necessary to apply to the curriculum being developed in order to improve it.

ACM Categories and Subject Descriptors: G.2.2 [Discrete Mathematics]: Graph Theory - Graph algorithms; K.3.2 [Computing Milieux]: Computers and Education - Computer and Information Science Education - Curriculum

Supervisor : Prof Derrick Kourie
Department : Department of Computer Science
Degree : PhD IT

Contents

List of Figures

List of Tables

List of Definitions

List of Transformation Rules

Chapter 1

Introduction

Many problems in the world exist where it is necessary to compare a given implementation with a specification and to provide some quantification as to whether the implementation complies with the specification. An example of such a problem is determining whether a given curriculum complies with a specification as stipulated by a professional body such as the ACM/IEEE curriculum for Computing. This thesis addresses such a problem.

The problem was identified when in a real life context an attempt was made to re-design a BSc Computer Science degree programme. The requirements for this degree programme were that it should comply with the ACM/IEEE Computing Curriculum for Computer Science and the requirements as stipulated by ABET's Computing Accreditation Commission. The development of the BSc Computer Science degree programme was initiated when the ACM/IEEE Computing curriculum 2001 was the most recent specification for Computer Science degree programme content[Joint Task Force for Computing Curricula, 2005]. The 2008 Review of the Computer Science curriculum was also imminent, but had not been released [ACM/IEEE-Curriculum CS2008 Joint Task Force, 2008].

Development of the curriculum relied on a spreadsheet. The rows of the spreadsheet represented the ACM/IEEE Computer Science curriculum 2001 and the columns, existing modules being presented in a degree that was to become the re-designed BSc Computer Science degree programme. The content of the existing modules was mapped onto the the ACM/IEEE curriculum specification by indicating for each module (column), the topics it addressed (rows). From this information a gap analysis between the existing modules content and the requirement of the ACM/IEEE curriculum 2001 was carried out. Development of the BSc Computer Science curriculum thereafter was iteratively carried out and each iteration was manually evaluated against the curriculum specification. A need was identified to semi-automate this process.

1.1 Research proposal

The research proposal under consideration is to investigate the extent to which the processes used to compare and develop curricula can be automated. To achieve this it is necessary to model the curricula and to use these models as input to the comparison process. The process for comparison is semi-automated and guided by the application of a framework.

1.2 Research approach

The research approach to be followed is to develop and verify a framework for digraph comparison. The framework will be illustrated in the domain of curriculum comparison and development. In such a comparison, curricula are modelled as digraphs in which vertices represent curricula elements (such as topics, knowledge untis, knowledge areas, year-levels, modules, etc.) and edges represent dependencies between these elements (such as belonging to a group, prerequisites, etc.).

An algorithm for comparing digraphs was developed by Marshall and Kourie [2010]. This algorithm will be refined and included in the framework. The framework will first be verified using toy applications. These applications will focus on the possible outcomes of the algorithm. Comparison techniques and the visualisation of the results will be proposed.

Verifying the framework using a real-world application will form the second part of the work presented. The real-world applications that will be verified include: the comparison of the ACM/IEEE curriculum volumes; and the comparison of the real-world BSc Computer Science degree programme to the curriculum volumes.

1.3 Research description

In this thesis a generalisation of the proposal will be presented. Curricula are modelled as directed graphs (digraphs). The vertices of the digraph represent the curriculum elements. The edges between the vertices represent the relationships between the curriculum elements. For the ACM/IEEE curricula volumes these are elements such as knowledge areas, knowledge units and topics. For the degree programme, these elements are represented by elements such as year-levels, modules and topics. The digraphs are represented as a *set of triples* [Barla-Szabo et al., 2004; Koopman, 2009]. Each triple is of the form (*source*, *destination*, *label*). The *source* is the start vertex of the directed edge and the *destination* the end vertex of the directed edge. Each edge has a *label* associated with it.

Modelling curricula as digraphs abstracts the process of curriculum comparison to one of comparing digraphs. The results of the digraph compar-

ison are used to facilitate the process of curriculum development in which the digraph models of the curricula are updated accordingly. The processes referred to in the proposal are guided by a framework which will be referred to as the Graph Comparison Framework.

The Graph Comparison Framework comprises of components which are logically related. The so-called Graph Trans-morphism Algorithm is used by the framework and is the entry point to all processes guided by the framework. The algorithm transforms the information in one of the digraphs to be compared into the structure of the other. This conversion enables the graphs to be compared as *graph isomorphisms*. The fact that the digraphs are represented as sets makes it possible to subtract one digraph from another using the *set minus* operator. The resulting difference sets are used by other components in the framework to quantify the comparison of the digraphs and to visualise the digraphs and/or the comparison quantification.

By modelling curricula as digraphs and applying the framework to the digraphs, it is possible to compare curricula. This application of the framework to a real-world problem forms the applications research part of the thesis. In this part, domain knowledge of curriculum design is necessary to apply to the curriculum being developed in order to improve it.

1.4 Significance of the study

The framework presented in this thesis contributes to the computer science body of knowledge. It proposes an algorithm for comparison that generates a subgraph isomorphism of a digraph in terms of the structure of one graph being compared using the information of the other. Comparison of the subgraph isomorphism is carried out by quantifying and visualising the differences and similarities between the subgraph isomorphism, the digraph representing the structure and the digraph representing the information.

The contribution made in the context of the application of the framework to curriculum comparison and development allows for curricula to be compared in a semi-automated fashion. The task of comparison can be repeated more frequently as the time taken to setup and execute the comparison is shortened.

1.5 Scope of the study

The curriculum comparison and development discussed in the thesis will be limited to the core elements defined in the ACM/IEEE Curriculum volumes. Using the core elements will adequately illustrate the use of the framework. The addition of elective elements as defined in the ACM/IEEE Curriculum volumes is a decision to be made by the curriculum expert. The choice of comparing only the core, or only the electives, or the core and electives is

currently a manual process. It is envisaged that this kind of choice will be included in a tool that makes use of the framework and will no longer be a manual process. The digraph representation already makes provision for the inclusion of meta-information using the *label* element of the *set of triples*. This enables the *label* to be further expanded to include additional meta-information such as whether the element is core/elective, as well as time and credit constraints. The framework needs to be extended to make provision for these constraints.

In the introductory paragraph reference was made to the comparison of a curriculum with accreditation requirements. To implement accreditation comparison will require the framework to be extended. Accreditation requirements in general are not as specific as curriculum requirements. In many cases a general requirement such as "data structures and algorithms" is given. In general this maps onto the Knowledge Units specified in the ACM/IEEE Curriculum volumes. The details with regards to the topics relating to the Knowledge Units is not provided by the accreditation specification. A preprocessing step is therefore required to compare a real-world curriculum with an accreditation specification. This pre-processing step will need to extract the required topics for the accreditation structure from the curriculum volume from which he curriculum was designed. This process will not be discussed in this thesis.

The representation that will be used to model the curricula is digraphs. Digraphs are based on mathematical principles and modelling digraphs as a *set of triples* increases the expressiveness of the representation for comparisons. Digraphs are not the only representation. There are other more expressive representations that exist such as concept lattices and ontologies that could conceivably also have been used to represent the curricula. Techniques exist to convert between these representations and digraphs. It is therefore possible to extend the framework to include these representations.

Extensions to the scope presented in this thesis will be addressed in more detail in Chapter 12, Future Work. The Future Work chapter further expands on how the framework can be improved to facilitate curriculum comparison and development. It should be noted, that the framework will in all probability never fully automate the curriculum comparison process. Human intervention and domain expertise will continually be required. It will however provide some sort of automation of the process.

1.6 Structure of the document

The document comprises of an introductory chapter – which is this chapter, and four parts. Parts I, II and III form the main body of the thesis. Each part comprises of a number of chapters. Part IV groups the appendices.

Part I, referred to as Theory, presents a literature review of relevant

graph theory and algorithms. It introduces an algorithm to facilitate digraph comparison. A framework, which incorporates the algorithm is also presented. Part II of the thesis presents an overview of an application domain, curricula, for applying the comparison framework. A few non-trivial scenarios are presented to illustrate how the framework is applied to a real-world application. The third part, Part III, presents a chapter on future work before concluding. More detail regarding Part I, Part II and Part III will briefly be discussed in the paragraphs that follow.

Overview of Part I - Theory

Part I considers the theoretical background required for the framework. It presents an overview of relevant graph theory in Chapter 2 as this is the non-generative model[1] that is to be used to present the curricula being modelled in the Application part of the dissertation. The models specifically make use of directed graphs, or digraphs. Therefore Chapter 4 introduces implementation techniques for digraphs as well a brief overview of algorithms used to manipulate digraphs. Because algorithms are characterised by their complexity, and because algorithms presented here therefore require such characterisation, an overview of complexity theory is presented in Chapter 3.

Chapters 5 and 6 present an algorithm and framework for digraph comparison. The algorithm is based on the notion that two digraphs are to be compared. One of these digraphs represents what is being aspired to. This could be, for example, a specification. In Part II where the application is discussed the specification could refer to the ACM/IEEE Computer Science curriculum volume. The other digraph represents an implementation. In many cases, these two digraphs are not directly comparable due to structural differences between the digraphs. The algorithm presented in Chapter 5 builds a third digraph using the information presented in the implementation and found in the structure of the specification. This third digraph is used to determine how well the digraphs compare when applying the framework for comparison presented in Chapter 6.

Examples of the application of the algorithm and framework presented in this part makes use of toy applications.

Overview of Part II - Application

Part II introduces the application domain relating to Computer Science curricula and the application of the framework for comparing curricula as a means of refining (perhaps iteratively) a curriculum.

[1] A *generative model* is a model that provides both syntax and semantics. An ontology is an example of a generative model. It provides the structure as well as a concept language to query the structure. A *non-generative model* is a model that only provides the structure [Andreasen et al., 2003].

The first two chapters of the part provide background to curricula specifications and requirements for real-world curricula. In Chapter 8, an overview is presented of the curricula specifications defined by the ACM/IEEE joint task groups related to Computer Science. Chapter 9 considers the legislative and institutional requirements placed on a real-world curriculum. It also briefly presents the challenges faced when developing a Computer Science curriculum in the South African context.

Chapter 10 provides the link between Part I and the application domain. The modelling of curricula in terms of digraphs, both with respect to the ACM/IEEE Curriculum volumes and with respect to a real-world BSc curriculum, is presented. The challenges of capturing topic data when comparing and developing a real-world curriculum is briefly discussed. Once a curriculum has been modelled as a digraph, its integrity has to be checked and the possibility needs to be considered of equivalences between topics in the respective curricula in order to improve the integrity of the digraph comparison. These aspects are also highlighted in this chapter.

The final chapter of the part, Chapter 11, presents areas in which the framework can be applied in the application domain of curriculum comparison and development. Two of the identified areas are illustrated using non-trivial scenarios and the results thereof are presented.

Overview of Part III - Future Work and Conclusion

The third part comprises of two chapters. Chapter 12 discusses shortcomings of the work presented in the thesis. It also presents what needs to be completed in order to develop a tool that can be used to assist in curriculum comparison and development.

The final chapter of this part and of the thesis is the Conclusion.

Part I

Theory

Chapter 2

Graph Theory

2.1 Introduction

Graphs are a well known and well researched branch of mathematics with applications in many areas. Graphs are used to model systems and the relationships between system parts. Graphs are particularly well suited for systems that require the modeling of rules [Andries et al., 1999] and require a structure of the system to be maintained [Heckel, 2006], conceptually, behaviourally or both.

The graph theory presented in this chapter is by no means complete. It provides an overview of the fundamentals of graphs and basic definitions so that the topics can be discussed or expanded on in later chapters.

The chapter is divided into three sections devoted to discussing types, matching and the transformation of graphs. Graph types, Section 2.2, introduces the notation to be used in the dissertation when specifying graphs. Section 2.3 discusses graph matching techniques. These techniques fall into two categories, they are either exact or inexact. The exact matching technique that solves the *subgraph isomorphism problem* is of particular interest later in the thesis and therefore the definition of a subgraph isomorphism is discussed. A definition and example for graph transformation is given in Section 2.4.

2.2 Graph types

All graphs can be seen as having vertices and edges connecting the vertices. How these vertices and edges are specified dictate the properties the defined graphs have. In this section, three basic graph types are presented, beginning with the most general form of a graph, the undirected graph where edges are bidirectional. More specialised graph types, with directional edges are also defined, namely the directed graph and the directed acyclic graph.

2.2.1 Undirected graph

Many mathematics-based definitions for (undirected) graphs have been published. These graph definitions vary in detail, but each definition essentially relies on the fact that a graph comprises of a set of vertices, a set of edges and a way of specifying the edges between the vertices. As illustration, two definitions are presented with a publication span of just under 30 years between them.

The first definition, reproduced in Definition 2.1, was published in 1976 by Bondy and Murty. The definition views a graph as a 3-tuple (or triple). The first element of the triple represents a set of vertices. According to the definition, a graph is defined by at least 1 vertex. The second triple element represents the set of edges. Each edge is defined using the third element of the triple, which associates an edge with a pair of vertices. According to the definition, this is an unordered pair. However, since a pair is normally regarded as ordered, it would probably be better to regard the incidence function as mapping to a set of cardinality 2.

Definition 2.1 (*Graph definition 1 - Bondy and Murty, 1976*)

A graph G is defined as an ordered triple $(V(G), E(G), \psi_G)$, where:

i $V(G)$ *is a non-empty set of vertices*

ii $E(G)$ *is a set disjoint from* $V(G)$ *of edges, and*

iii ψ_G *is an incidence function that associates with each edge of* G *an unordered pair of vertices of* G.

An example of a specification of a graph using the definition presented in Definition 2.1 is given by:

$$\begin{aligned}
V(G) &= \{a, b, c, d, e, f\} \\
E(G) &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\} \\
\psi(e_1) &= (a, b), \\
\psi(e_2) &= (a, c), \\
\psi(e_3) &= (a, f), \\
\psi(e_4) &= (b, c), \\
\psi(e_5) &= (c, d), \\
\psi(e_6) &= (d, e), \\
\psi(e_7) &= (b, f), \\
\psi(e_8) &= (b, e)
\end{aligned}$$

From the specification, it should be noted that the definition implicitly labels the edges using the elements of the set $E(G)$ to represent the labels.

The label for the edge between vertices c and d, is e_5.

The second definition of a graph, defined by Diestel in his book on *Graph Theory*, is reproduced in Definition 2.2. In this case a graph is defined as a pair comprising of a set of vertices and a set of edges. Each element in the set of edges is a subset of V comprising of two elements. In this definition it is possible to have an empty set of vertices, which implies an empty set of edges.

Definition 2.2 (*Graph definition 2 - Diestel, 2005*)

A graph G is defined as a pair of the form (V, E), where V is a set of vertices of graph G and E a set of edges. The set E is a 2-element subset of V ($E \subseteq [V]^2$). It is further assumed that $V \cap E = \emptyset$.

The assumption that $V \cap E = \emptyset$ ensures that the graph does not contain loops. A loop is defined by Definition 2.3. To illustrate the need for this assumption, consider the following specification for a graph that adheres to this definition:

$$\begin{aligned} V &= \{a, b, c\} \\ E &= \{\{a, b\}, \{b, b\}, \{b, c\}\} \end{aligned}$$

The edge $\{b, b\} \in E$, according to set theory, will reduce to $\{b\} \in E$, which is in V resulting in $V \cap E = \{b\}$.

Definition 2.3 (*Loop*)

A loop is an edge that connects a vertex to itself.

An example of a specification of a graph using the definition presented in Definition 2.2 that results in the same graph as the specification given for Definition 2.1 is given by:

$$\begin{aligned} V &= \{a, b, c, d, e, f\} \\ E &= \{\{a, b\}, \{a, c\}, \{a, f\}, \{b, c\}, \{c, d\}, \\ &\quad \{d, e\}, \{b, f\}, \{b, e\}\} \end{aligned}$$

From the example above, it can be seen that the definition does not make provision for labeling of the edges.

Both definitions presented by Definition 2.1 and 2.2 have a similar structure. Both define a set of vertices and a set of edges and in both the set of edges is defined in terms of a function that is applied to the set of vertices. Definition 2.1 allows for loops, but not an empty graph. Definition 2.2 does

allow for a graph to be defined as empty, it does not allow for loops or the explicit labeling of edges.

For the thesis, a graph is required to have the following properties:

- A graph can be empty implying that there are no vertices for the graph, yet the graph is defined to exist. This property is necessary for the discussion of the algorithm proposed in Chapter 5.
- A graph edge must be able to carry a label along with the possibility of additional meta-data associated with the edge.
- The graph does not contain loops.

A definition, which takes the above properties into account for an undirected graph, is given by Definition 2.4 using a notation that represents operations which take parameters. These operations are analogous to functions in computer programming languages. Assigning the result of an operation to a variable indicates that the execution of the operation will give a resultant value that will be assigned to the variable. The general form of an operation is given by:

$$var \quad = \quad operation(parameterlist)$$

Definition 2.4 (*Undirected graph*)

A graph, defined by $G = G(V, E)$, comprises of:

i $V = V(G)$*, a set of elements, called vertices of* G*.*

ii $E = E(G)$*, a set of edge pairs of* G*. Each edge pair,* $(\mathcal{E}, \mathcal{L})$*, comprises an edge* $(\mathcal{E})$ *and a label* $(\mathcal{L})$*.* $\mathcal{E}$ *is a two element subset of* V *(i.e.* $\mathcal{E} = \{v_1, v_2\}$*, where* $v_1 \neq v_2$ *and* $v_1, v_2 \in V$*) and* $\mathcal{L}$ *is an n-tuple of which the first element enumerates the label* $(label, ...)$ *of the corresponding edge.*

Definition 2.4 satisfies all the desired properties,

- $V = \emptyset$ is valid. It follows that if $V = \emptyset$ then $E = \emptyset$.
- Edge pairs in E incorporate an n-tuple representation for a label.
- The definition of the edge as a two element subset of V ensures that the elements in the subset cannot be the same with $v_1 \neq v_2$, otherwise it would reduce the subset to one element making it invalid.

A graph defined using Definition 2.4 is represented by two sets, a set of vertices (V), and a set of edges (E). Each pair in E comprises of a set of

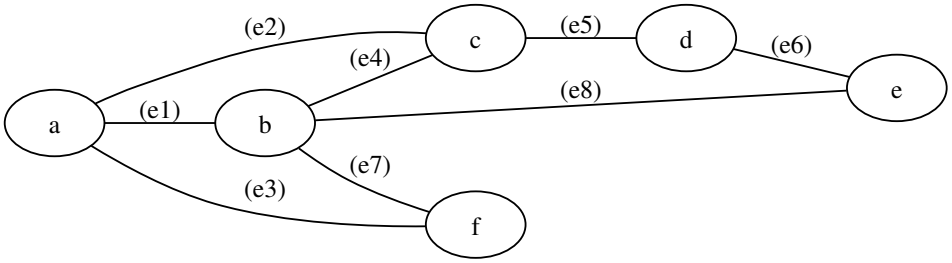


Figure 2.1: Example of graph G

two elements of V and an n-tuple representing the label. The following is a *textual representation* of a graph G comprising of 5 vertices and 8 edges that represents the same graph as for the specifications used to illustrate Definitions 2.1 and 2.2.

$$\begin{aligned} V &= \{a, b, c, d, e, f\} \\ E &= \{(\{a, b\}, (e_1)), \\ &\quad (\{a, c\}, (e_2)), \\ &\quad (\{a, f\}, (e_3)), \\ &\quad (\{b, c\}, (e_4)), \\ &\quad (\{c, d\}, (e_5)), \\ &\quad (\{d, e\}, (e_6)), \\ &\quad (\{b, f\}, (e_7)), \\ &\quad (\{b, e\}, (e_8))\} \end{aligned}$$

For a *visual representation* of G, a vertex $v \in V$, is represented by an oval. An edge of an edge pair is represented by an arc connecting v_1 and v_2 and the label is placed on the corresponding arc. The visual representation of the textual representation given above is presented in Figure 2.1.

2.2.2 Directed graph

A directed graph (also referred to as a digraph) G, is a graph in which the edges have direction. Each edge begins at a source vertex and ends at a destination vertex [Diestel, 2005]. For there to be an edge in the opposite direction it needs to be specifically defined as an edge of the graph. It is permissible to have an edge in the opposite direction as well so that the source of one edge is the destination of the other, and *vice versa*. A formal definition, in the notation of the Definition 2.4 of a graph, is given by Definition 2.5.

Definition 2.5 (*Directed graph - Digraph*)

A digraph, defined by $G = G(V, E)$, comprises of:

i) $V = V(G)$, *a set of elements, called vertices of* G.

ii) $E = E(G)$, *a set of edge pairs of* G. *Each edge pair,* $(\mathcal{E}, \mathcal{L})$, *comprises an edge* $(\mathcal{E})$ *and a label* $(\mathcal{L})$. $\mathcal{E}$ *is an ordered pair comprising elements of* V *(i.e.* $\mathcal{E} = (v_1, v_2)$, *where* $v_1 \neq v_2$ *and* $v_1, v_2 \in V$*) and* $\mathcal{L}$ *is an n-tuple of which the first element enumerates the label* $(label, ...)$ *of the corresponding edge.*

iii) Each edge $(\mathcal{E})$ *is defined by two mappings, namely*

$$source : E \longrightarrow V$$

and

$$destination : E \longrightarrow V$$

representing the edge, $\mathcal{E} = (source, destination)$, *showing the direction of the edge from source to destination.*

If the digraph has several edges between the same two vertices then these edges are called *multiple edges*. If these edges have the same direction they are referred to as *parallel*. The definition of a *digraph* allows for cycles. A cycle is defined in terms of a walk. The definitions of a walk and a cycle are given by Definitions 2.6 and 2.7 respectively [Bondy and Murty, 1976]. The term *distinct* used in the definition of a cycle means that the vertex is not repeated in the sequence.

Definition 2.6 (*Walk*)

A walk in G *is a finite non-empty sequence of alternating vertices and edges,* $W_G = [v_0\ e_1\ v_1\ ...\ e_k\ v_k]$. *For an edge,* $e_i \in E$, $1 \leq i \leq k$, *the vertices (* $v_i \in V, 1 \leq i \leq k$*) on either side are* v_{i-1} *and* v_i. v_0 *and* v_k *are referred to the origin and the terminus of the walk respectively.* e_i *is either represented in* $\mathcal{E}$ *or* $\mathcal{L}$.

Definition 2.7 (*Cycle*)

A cycle in G *is a walk in which* $v_0 = v_k, k \geq 1$ *and* v_1 *to* v_{k-1} *are distinct,* $C_G = [v_0\ e_1\ v_1\ ...\ e_k\ v_0]$.

The graph for V and E as defined below, is given in Figure 2.2. The graph contains a cycle given by the sequence $C_G = [b\ (e4)\ c\ (e5)\ d\ (e6)\ e\ (e8)\ b]$.

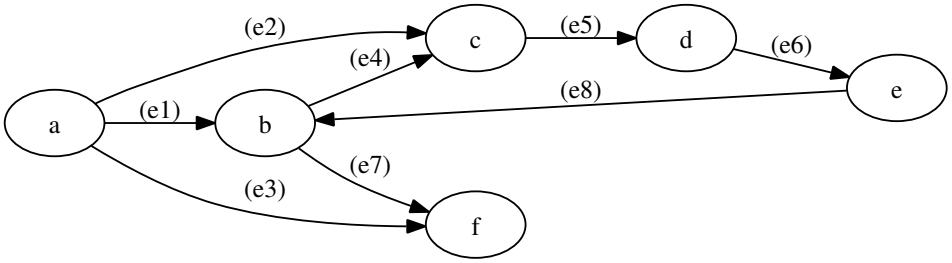


Figure 2.2: Example of digraph G

$$
\begin{aligned}
V &= \{a, b, c, d, e, f\} \\
E &= \{((a, b), (e1)), \\
&\quad ((a, c), (e2)), \\
&\quad ((a, f), (e3)), \\
&\quad ((b, c), (e4)), \\
&\quad ((c, d), (e5)), \\
&\quad ((d, e), (e6)), \\
&\quad ((b, f), (e7)), \\
&\quad ((e, b), (e8))\}
\end{aligned}
$$

Not only are cycles defined in terms of walks, but so are paths. A path can be seen a specialisation of a cycle. A path beginning at a specific vertex in a graph and ending at another vertex of the same graph is given by Definition 2.8.

Definition 2.8 (*Path*)

A path in G is a walk in which both the vertices, v_0 to v_k , and edges, e_1 to e_k , of the walk are distinct. A path in G is written as $P_G = [v_0\ e_1\ v_1\ ...\ e_k\ v_k]$.

The length of the path is the number of edges that the path comprises of. An example of a path in digraph G presented in Figure 2.2 is given by $P_G = [b\ (e4)\ c\ (e5)\ d\ (e6)\ e]$. The length of P_G is 3.

2.2.3 Directed acyclic graph

A *Directed Acyclic Graph* (DAG) is a *digraph* without cycles [Bang-Jensen and Gutin, 2007]. To convert the digraph in Figure 2.2 into a DAG, one of the edges in the cycle would have to be excluded from the graph. For example, if the edge labelled $(e8)$ was excluded, a DAG would result with a representation as given in Figure 2.3.

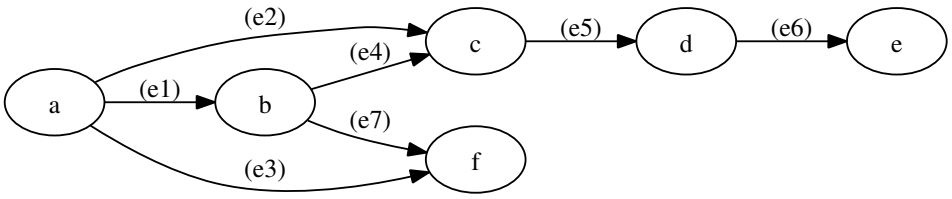


Figure 2.3: Example of DAG G

2.3 Graph matching

The notion of graph matching entails the use of techniques to determine the similarity between two graphs. Bengoetxea [2002] and Zaslavskiy [2010] differentiate between exact and inexact graph matching. With exact graph matching the graphs are said to be *isomorphic*, while with inexact graph matching the graphs are *homomorphic*. Definition 2.9 defines exact graph matching.

Definition 2.9 (*Exact graph matching - Graph isomorphism*)

Consider two graphs, $G_A(V_A, E_A)$ and $G_B(V_B, E_B)$ where the number of vertices in V_A is the same as the number of vertices in V_B, $\mid V_A \mid = \mid V_B \mid$. Suppose there exists a one-to-one mapping

$$\mathcal{F} : V_A \longrightarrow V_B$$

such that

$$\{v_1, v_2\} \in E'_A \iff \{\mathcal{F}(v_1), \mathcal{F}(v_2)\} \in E'_B$$

where, for $n = A, B$

$$E'_n = \{e | (e, l) \in E_n\}$$

Then $\mathcal{F}$ is referred to as an isomorphism, and G_A is said to be isomorphic to G_B.

The most extreme case of similarity is when the graphs are identical, defined by Definition 2.10. This is also referred to as a graph *automorphism* [Diestel, 2005].

Definition 2.10 (*Identical graphs - Graph automorphism*)

Two graphs, G_A and G_B, are identical if:

i) the set of vertices of G_A is equal to the set of vertices of G_B, that is $V_A = V_B$; and

ii) the set of edges extracted from the set of edge pairs of G_A (E'_A) is equal to the set of edges extracted from the edge pairs of G_B (E'_B),

that is $E'_A = E'_B$.

Identical graphs will have the same diagrammatic representation [Bondy and Murty, 1976]. Non-identical graphs may have the same diagrammatic representation and may be found to be *isomorphic* if the graph matching property of the number of vertices of the two graphs are the same and there exists a bijective mapping function $\mathcal{F}$.

When the graph matching property of $\mid V_A \mid = \mid V_B \mid$ does not hold, no graph isomorphism can be determined and the problem changes from finding exact matches between vertices to finding the best match between vertices. These problems belong to the class of problems known as *inexact* graph matching. In such cases, a non-bijective relationship between G_A and G_B is sought [Bengoetxea, 2002], also referred to as a graph *homomorphism*.

In the sections that follow, the graph matching techniques are discussed in more detail. The graph comparing algorithm presented in Chapter 5 makes use of the notions presented by these techniques for building a sub-graph isomorphism.

2.3.1 Graph isomorphism

A graph isomorphism (iso - equal, morphism - shape), which holds for both undirected and directed graphs, is a 1-to-1 mapping of the vertices in the graph G_A onto the vertices in the graph G_B such that the edges of the vertices are preserved. The definition for exact graph matching (Definition 2.9) is equivalent to the definition of a graph *isomorphism* [Bondy and Murty, 1976; Diestel, 2005; Bang-Jensen and Gutin, 2007].

The notation used to denote that graph G_A is isomorphic to graph G_B is given by $G_A \cong G_B$.

Consider the following two textual representations and their respective graphical representations (given in Figures 2.4 and 2.5) of graphs G_A and G_B.

$$V_A = \{a, b, c, d, e\}$$

$$E_A = \{(\{a, b\}, ()), (\{b, c\}, ()), (\{c, d\}, ()), (\{d, a\}, ()), (\{a, e\}, (), (\{e, d\}, ())\}$$

$$V_B = \{g, h, i, j, k\}$$

$$E_B = \{(\{k, g\}, ()), (\{g, h\}.()), (\{h, i\}, ()), (\{i, j\}, ()), (\{j, k\}, ()), (\{k, i\}, ())\}$$

From their respective figures, G_A and G_B look different. By applying Definition 2.9 it can be shown that G_A and G_B are isomorphic. The first part of the definition states that the number of vertices in each of the graphs must be equal, G_A has 5 vertices and so does G_B. The second requirement of the definition requires that some mapping $\mathcal{F}$ between the vertices can

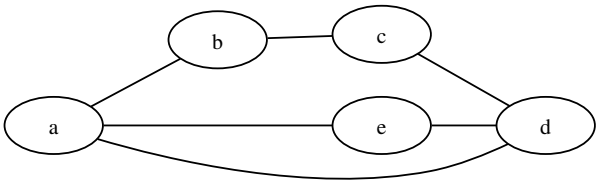


Figure 2.4: Visual representation of example graph G_A

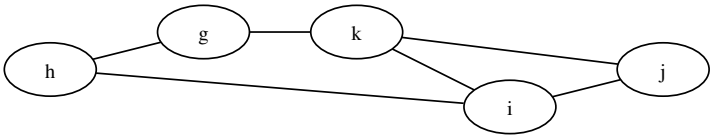


Figure 2.5: Visual representation of example graph G_B

be found such that the edges of G_A are preserved in G_B and *vice versa*. There exists such a mapping, $\mathcal{F}(a) = i, \mathcal{F}(b) = h, \mathcal{F}(c) = g, \mathcal{F}(d) = k$ and $\mathcal{F}(e) = j$. The set of edges extracted from G_A is given by:

$$E'_A = \{\{a,b\},\{b,c\},\{c,d\},\{d,a\},\{a,e\},\{e,d\}\}$$

By applying the mapping $\mathcal{F}$ to E'_A, the following set of edges result:

$$\begin{aligned} E'_B &= \{\{\mathcal{F}(a),\mathcal{F}(b)\},\{\mathcal{F}(b),\mathcal{F}(c)\},\{\mathcal{F}(c),\mathcal{F}(d)\},\{\mathcal{F}(d),\mathcal{F}(a)\}, \\ &\quad \{\mathcal{F}(a),\mathcal{F}(e)\},\{\mathcal{F}(e),\mathcal{F}(d)\}\} \\ &= \{\{i,h\},\{h,g\},\{g,k\},\{k,i\},\{i,j\},\{j,k\}\} \end{aligned}$$

The edges of E'_A map directly onto E'_B; therefore $G_A \cong G_B$.

A graph isomorphism therefore compares the structures of graphs. The vertex "names" and labels of the edges are merely used for referral purposes.

Subgraphs and graph isomorphisms

The definition of a subgraph is given by the Definition 2.11 [Diestel, 2005; Bondy and Murty, 1976].

Definition 2.11 (*Subgraph*)

A graph G_A is a subgraph of G_B if $V_A \subseteq V_B$ and $E'_A \subseteq E'_B$.

The notation used to show that G_A is a subgraph of G_B is given by, $G_A \subseteq G_B$.

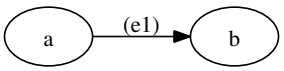


Figure 2.6: An edge

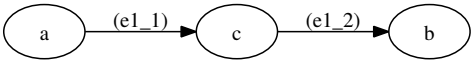


Figure 2.7: An edge subdivision

Definition 2.12 (*Subgraph isomorphism*)

Graph G_A is a subgraph isomorphism of G_B if $V_A \subset V_B$ and there exists a mapping $\mathcal{F} : V_A \longrightarrow V_B$ such that $\{v_1, v_2\} \in E'_A \iff \{\mathcal{F}(v_1), \mathcal{F}(v_2)\} \in E'_B$.

The subgraph isomorphism mapping needs only to preserve the edges of the vertices defined in the subgraph (graph G_A). Any edges that may exist between vertices in G_A and those only, need to be preserved in G_B.

Graph homeomorphism

A graph homeomorphism (homeo - similar, morphism - shape) is a topological graph isomorphism. This means that if vertices can be added to one graph to get another graph, then the graph is homeomorphic. The definition, given by Definition 2.14, adds vertices by using a technique called edge subdivision (Definition 2.13) [Alekseev, 2013].

Definition 2.13 (*Edge subdivision*)

A subdivision of an edge $\{v_1, v_2\} \in E'$ of graph G results in the addition of a vertex, say u, to graph G resulting in two edges $\{v_1, u\}, \{u, v_2\} \in E'$.

Consider the edge given in Figure 2.6. The subdivision of the edge will result in:

- the edge, $(e1)$, being removed from the graph,
- a vertex, c, being added to the graph, and
- the addition of two edges linking the additional vertex to the original vertices that were linked by the original edge. Edge $(e1\_1)$ links vertices a and c and $(e1\_2)$ links vertex c to vertex d.

Refer to Figure 2.7 for the result of the edge subdivision described above. It is important that edge subdivision preserves edge direction in digraphs and DAGs.

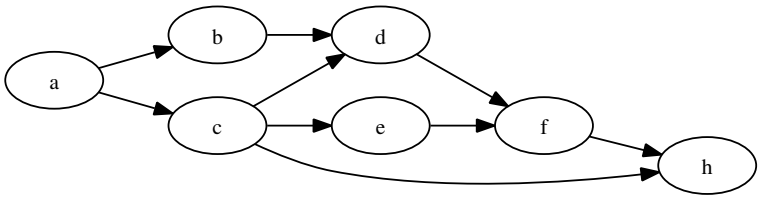


Figure 2.8: Visual representation of example DAG G_C

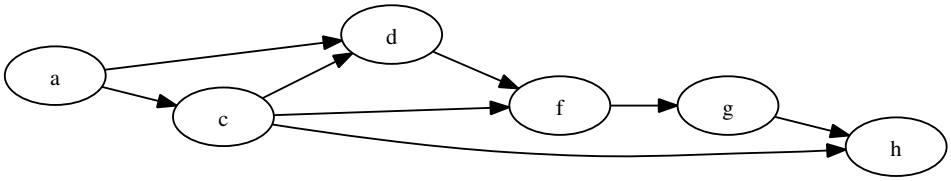


Figure 2.9: Visual representation of example DAG G_D

Definition 2.14 (*Graph homeomorphism*)

Suppose that G_A, G_B, G'_A, G'_B are graphs that conform to the following:

i) G'_A (G'_B) is derived from G_A (G_B respectively) by a sequence of zero or more edge subdivisions.

ii) $G'_A \cong G'_B$

Then G_A and G_B are said to be homeomorphic

The notation used to denote that two graphs G_A and G_B are homeomorphic is $G_A \approx G_B$.

Consider the two DAGs in Figures 2.8 and 2.9. Note that the vertex naming need not have been the same, but for ease of discussion they have been made to match.

For each of the digraphs in the figures, for each vertex in the graphs, the in-degree and out-degree of the vertex can be determined. The in-degree of a vertex is the number of edges entering the vertex and the out-degree

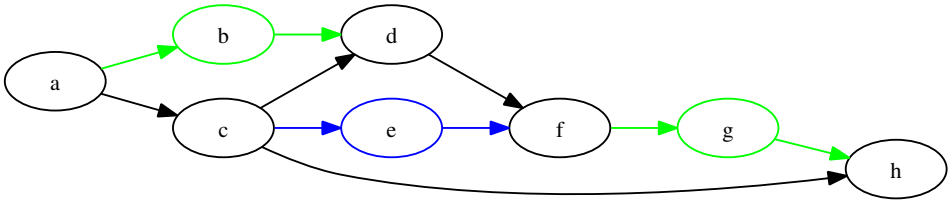


Figure 2.10: Visual representation of subdivided DAGs G'_C and G'_D

the number of edges leaving the vertex. This information is summarised in Table 2.1. Vertices that do not have values in the table are not in their respective graphs. This means that there is no vertex g in graph G_C and graph G_D does not have vertices b and e. Applying a subdivision in the respective graphs for these vertices will result in the graph given in Figure 2.10 which represents G'_C (edge subdivision denoted in blue) and G'_D (edge subdivisions denoted in green) with $G'_C \cong G'_D$. It then follows that G_C and G_D are homeomorphic, $G_C \approx G_D$.

	a	b	c	d	e	f	g	h
G_C	0 2	1 1	1 3	2 1	1 1	2 1		2 0
G_D	0 2		1 3	2 1		2 1	1 1	2 0

Table 2.1: In- and Out-degrees of vertices of G_C and G_D

2.3.2 Graph homomorphism

In the mathematical field of graph theory a graph homomorphism (homo - same, morphism - shape) is a mapping between two graphs that respects their structure. More concretely it maps adjacent vertices to adjacent vertices by preserving the edges. A homomorphism is defined in Definition 2.15 [Bang-Jensen and Gutin, 2007].

Definition 2.15 (*Homomorphism*)

Consider two graphs G_A and G_B. If there exists a mapping $\mathcal{F} : V_A \longrightarrow V_B$, such that $\{v_1, v_2\} \in E'_A \implies \{\mathcal{F}(v_1), \mathcal{F}(v_2)\} \in E'_B$ then $\mathcal{F}$ is referred to as a graph homomorphism between G_A and G_B, G_A is said to be homomorphic to G_B.

The notation used to denote that the graph G_A is homomorphic to G_B is $G_A \rightarrow G_B$.

2.4 Graph transformation

Graph transformation, also referred to as graph rewriting or graph reduction, is a technique used to transform one graph to another by following a set of rules, or algorithm [Heckel, 2006]. Graph transformations are used to generate, manipulate and evaluate graphs [Andries et al., 1999].

Definition 2.16 provides a general definition for a graph transformation [Andries et al., 1999]. The basic idea of the graph transformation process is to iteratively apply a rule, from a set of rules or as defined by an algorithm, to a graph, thereby transforming the original graph to a new graph.

Definition 2.16 (*Graph transformation*)

A graph transformation comprises of a set of graph rewriting rules of the form $\mathscr{L} \longrightarrow \mathscr{R}$.

- $\mathscr{L}$ *is the left-hand or pattern side of a particular rule. It may represent a vertex, an edge or a subgraph of* G, $\mathscr{L} \subseteq G$.
- $\mathscr{R}$ *is the right-hand or replacement side of a particular rule. As with* $\mathscr{L}$ *it may represent a vertex, an edge or a subgraph to be inserted into* G.

A rule is applied to G *by finding an (all) occurrence(s) of* $\mathscr{L}$ *in* G *and replacing it (them) with* $\mathscr{R}$ *thereby transforming* G.

Each rule in the set is applied to G.

The process of inserting $\mathscr{R}$ into graph G requires the vertices in $\mathscr{R}$ to be connected to what is left of G when $\mathscr{L}$ has been removed. After a graph transformation rule has been applied to a graph G, the graph will be rewritten as $G = (G - \mathscr{L}) + \mathscr{R}$.

To illustrate a graph transformation, consider the following set of graph rewriting rules for a digraph G:

Rule 1: $(L) \longrightarrow (L, update)$, where $(L) \in \mathcal{L}$

Rule 2: $\{((a,b),L)\} \longrightarrow \{((a,c),(e1\_1)),((c,b),(e1\_2))\}$, where $L \in \mathcal{L}$

Rule 3: $\{((X,Y),L_1),((Y,X),L_2)\} \longrightarrow \{((X,Y),L_1+L_2)\}$, where $X, Y \in V$ and $L_1, L_2 \in \mathcal{L}$

Rule 1 is a general rule for updating the edge n-tuple of all edges in G represented by a single label to a pair including the original label as first element of the pair and "updated" as the second element.

Rule 2 is an example of edge subdivision as illustrated by Figures 2.6 and 2.7. It is also defined to find a specific occurrence specified by $\mathscr{L}$ and replace it with a specific $\mathscr{R}$ in G. The rule could have been written in a general form to cater for all edge subdivisions. A general version of the rule is given by Rule 2.1.

Rule 2.1 (*Edge subdivision*)

$\{((X,Y),L)\} \longrightarrow \{((X,Z),L_1),((Z,Y),L_2)\}$, *where* $X, Y \in V$, $L \in \mathcal{L}$

In the edge subdivision rule, Z after transformation becomes an element of V, $V = V + Z$ and the edge pair represented by $\mathscr{L}$ is removed from E and the two edges represented by $\mathscr{R}$ are included in E, $E = E + ((X,Z),L_1) + ((Z,Y),L_2) - ((X,Y),L)$. This general transformation will need to be guided

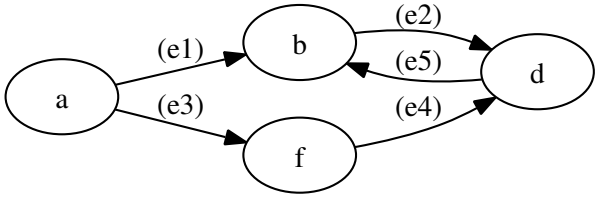


Figure 2.11: Initial graph G before transformation

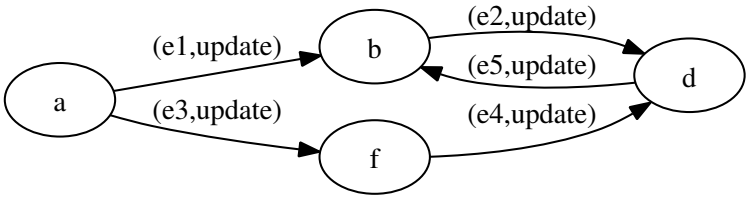


Figure 2.12: G after Rule 1 has been applied iteratively

by specific values for X, Y, Z, L, L_1 and L_2 such as in the example given by Rule 2 above.

Rule 3 represents a general rule for removing cycles from G which comprises of two vertices and two edges.

An example digraph to which the set of graph rewriting rules is to be applied is given in Figure 2.11. A sequence of figures will be presented showing how the rules when applied in the order given and iteratively for the particular rule will transform the graph G. After application of Rule 1, which updates the labels, to G, the graph transforms to the digraph given in Figure 2.12. The second rule, Rule 2, subdivides the walk $[a\ (e1, update)\ b]$ to the walk $[a\ (e1_1)\ c\ (e1_2)\ b]$ as shown in Figure 2.13. Application of Rule 3 could result in two final graphs if there is no specification as to what the specific values of the variables in the rule should be. These final graphs are given in Figures 2.14 and 2.15.

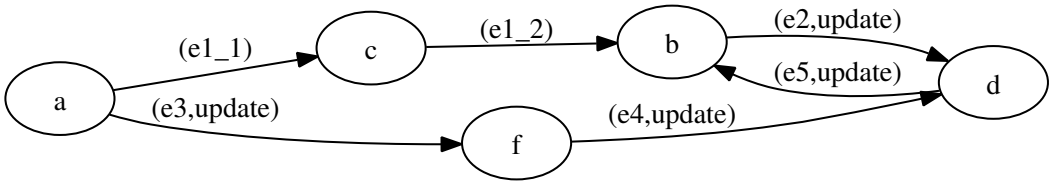


Figure 2.13: G after Rule 2 has been applied iteratively

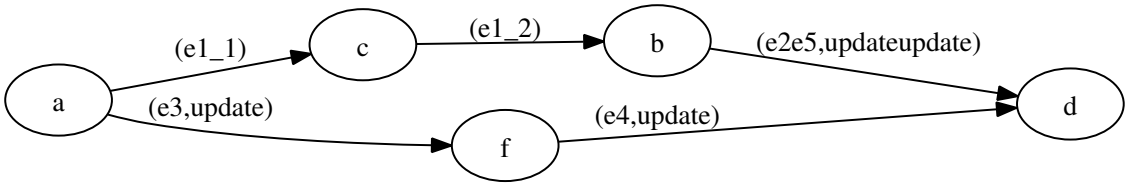


Figure 2.14: Final G after Rule 3 has been applied iteratively

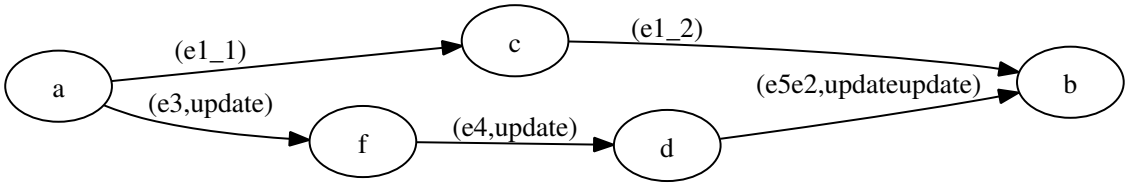


Figure 2.15: Alternative final G after Rule 3 has been applied iteratively

2.5 Conclusion

This chapter has provided a basic overview and definitions for concepts that are to be discussed or expanded on in further chapters of the thesis. Concepts to define basic graph types, graph matching and graph transformations were discussed.

The most important definition is the definition of an undirected graph, given by Definition 2.4, which defines a graph taking the desired properties of a graph into account as required in future chapters. This definition and the properties are then slightly modified to define a digraph as a special type of graph. The definition of a digraph will be used for specifying the algorithm defined in Chapter 5.

A broad overview of graph matching was given, defining both exact and inexact graph matching. The exact graph matching techniques are more relevant in this thesis, and therefore the discussion concentrated on these, only mentioning the inexact graph matching notion of a homomorphism for purposes of comparison. Exact graph matching includes determining whether two graphs are isomorphic and more specifically whether one is a subgraph isomorphism of the other. The special case of a graph isomorphism, referred to as a graph homeomorphism is presented as well in order to introduce the technique of edge subdivision which is to be used by the graph transformation algorithm in Chapter 5.

Chapter 3

Complexity Theory

3.1 Introduction

This chapter presents an overview of complexity theory. It contains well-known results in computer science and is provided to contextualise and present arguments in chapters that follow.

According to Black [2004a], complexity is defined as:

> *"The intrinsic minimum amount of resources, for instance, memory, time, messages, etc., needed to solve a problem or execute an algorithm."*

Complexity in this chapter will focus on time complexity, or processor usage, than it will on space complexity, or memory usage. Complexity will be discussed in terms of the Big-$\mathcal{O}$ notation in Section 3.2 and complexity classes in Section 3.5. As a precursor to complexity classes a decision problem will be defined and a discussion of Turing machines will be presented.

3.2 Big-$\mathcal{O}$ notation

Big-$\mathcal{O}$ is the notation most commonly used in mathematics to specify asymptotic complexity, or the rate at which a function, $f(n)$, grows [Drozdek, 2008]. In computer science, the Big-$\mathcal{O}$ notation is mostly used to classify the time complexity of an algorithm in terms of its execution time. The notation can also be effectively used to represent the space complexity of a data structure or algorithm in terms of its memory usage during execution.

The classification of algorithms and data structures in terms of the Big-$\mathcal{O}$ notation presents the worst-case time or space complexity for the algorithm or data structure [Harel, 1992]. Comparing the classifications of algorithms enables comparison between the algorithms to take place. These comparisons are typically on algorithms that perform similar functions, possibly with different underlying data structures either in terms of execution time

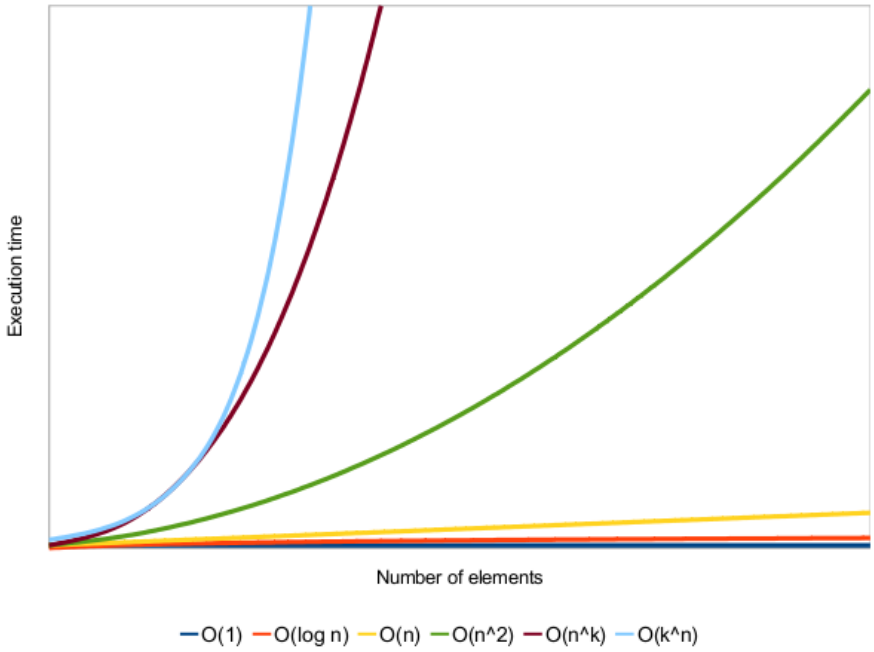


Figure 3.1: Common Big-$\mathcal{O}$ notation execution times

or memory usage during execution. In many cases, the algorithms perform adequately under normal conditions and the Big-$\mathcal{O}$ notation should be used taking the application in which the algorithm is used into consideration.

Common Big-$\mathcal{O}$ notation orders are given in Table 3.1. The notations are ordered beginning with the slower growing functions [Drozdek, 2008; Harel, 1992; Preiss, 1998]. The descriptions in the table explain the notation with respect to time and space complexities respectively.

A comparison of the execution times of the Big-$\mathcal{O}$ notation orders is given in Figure 3.1. As can be seen from the figure, moving through the execution order drastically influences the execution time. The further down the order the longer the execution time that is required for fewer elements. The classification of algorithms according to execution time for specific function orders relate directly to the complexity classes to which algorithms belong. These complexity classes will be discussed in Section 3.5. Space based complexities present a similar figure, with the vertical access representing memory usage instead of execution time.

3.3 Decision problems

A *decision problem* is a problem that can be answered with either a "yes" or a "no" as answer. The purpose of a decision problem to determine whether

Notation $\mathcal{O}(f(n))$	**Name**	**Description**
$\mathcal{O}(1)$	Constant	**Time:** The execution time remains the same for any number of input elements. **Space:** The memory used is independent of the size of the input.
$\mathcal{O}(\log n)$	Logarithmic	**Time:** The execution time initially rapidly increases and then flattens off as the number of input elements increases. **Space:** The memory usage stabilises for a large number of input elements.
$\mathcal{O}(n)$	Linear	**Time:** The execution time of the algorithms increases at the same rate as the number of input elements increases. **Space:** The memory used is directly proportional to the number of input elements.
$\mathcal{O}(n^2)$	Quadratic	**Time:** The execution time of the algorithm is directly proportional to the square of its number of input elements. **Space:** The memory usage is the number of input elements squared.
$\mathcal{O}(n^k), k > 1$	Polynomial	**Time:** The execution time reaches an upper bound for a polynomial expression in relation to the number of input elements. **Space:** The amount of memory needed to solve a problem is polynomial.
$\mathcal{O}(k^n), k > 1$	Exponential	**Time:** The execution time of the algorithm will increase by k for each additional input element. **Space:** The memory required to solve the problem will increase by $k^{p(n)}$ where $p(n)$ is a polynomial function of the space requirement for the input elements.

Table 3.1: Common Big-$\mathcal{O}$ notation function orders

a certain property holds or not. [Drozdek, 2008; Harel, 1992]

Specific instances of the problem require specific values as parameters and variables to be specified. A problem is *decidable* if there is a solution that answers the question for each instance, otherwise it is *undecidable* [Homer and Selman, 2011]. Problems are referred to as *tractable*, if there is an algorithm that will admit a solution in reasonable time, otherwise it is *intractable* [Harel, 1992]. The only difference between a problem being decidable or tractable is related to time. For a problem to be tractable, every instance must be solvable in polynomial time. For a problem to be decidable, it must be solvable for every instance with no time specification given.

A *deterministic algorithm* defines a unique sequence of steps that must be followed to achieve the result for the specified input. A *nondeterministic algorithm* is an algorithm that uses operations that take a "guess" at what decision is to be made. A nondeterministic algorithm solves a decision problem if there is a path in the decision tree that leads to a solution that answers the question. The algorithm is polynomial if it reaches a solution in the decision tree in $\mathcal{O}(n^k)$, where n is the size of the problem space [Drozdek, 2008].

3.4 Turing machines

A Turing machine, described by Alan Turing, is a theoretical model of a computing machine. A Turing machine consists of a read/write head and a linear tape comprising of cells in which symbols are written and read [Homer and Selman, 2011]. A Turing machine can be viewed as a computer with a fixed algorithm [Harel, 1992] and therefore the terms Turing machine and algorithm will be used interchangeably.

The definition for a deterministic single tape Turing machine is given by Definition 3.1 which has been adapted from definitions by Rayward-Smith [1986], Bovet and Crescenzi [2006] and Homer and Selman [2011].

Definition 3.1 (*Deterministic Turing machine*)

A deterministic Turing machine is a 6-tuple with $M = (Q, \sum, I, P, q_0, F)$, *where:*

1. Q *is a finite set of states,*
2. $\sum$ *is a finite set of symbols called the tape alphabet, of which the blank symbol,* $\llcorner$, *is always an element,*
3. I *is the set of input symbols with* $I \subseteq \sum \backslash \{\llcorner\}$
4. P *is the program defined by the partial function*

$$P : (Q \backslash F) \times \sum \rightarrow Q \times \sum \times \{L, R, 0\},$$

5. q_0 *is the initial state,*

6. $F \subseteq Q$ *is the set of final states.*

In the set $\{L, R, 0\}$, the L denotes a one cell move to the left by the Turing machine read/write head, R one cell move to the right and 0 no move. The program, P in state $q \in \{Q \backslash F\}$ taking a symbol $s \in \sum$, written as $P(q, s)$ is either undefined or a unique element of $Q \times \sum \times \{L, R, 0\}$. For $P(q, s) = (q', s', x)$ with $q' \in Q$, $s' \in \sum$ and $x \in \{L, R, 0\}$, the 5-tuple (q, s, q', s', x) will then appear in the program listing.

Intuitively, it would be expected that a multi-tape Turning machine would be more efficient than a single-tape Turing machine. In order to define a multi-tape Turing machine, with k tapes, the function defined in Definition 3.1 needs to be replaced by the following function [Bovet and Crescenzi, 2006; Homer and Selman, 2011]:

$$P : (Q \backslash F) \times \sum^k \rightarrow Q \times \sum^k \times \{L, R, 0\}^k,$$

Whether a multi-tape or single-tape Turing machine is used, complexity classes, including polynomial time classes, remain unaffected. Furthermore, every multi-tape Turing machine also has an equivalent single-tape Turing machine [Homer and Selman, 2011].

The definition of a nondeterministic Turing machine is given by Definition 3.2 [Rayward-Smith, 1986; Homer and Selman, 2011].

Definition 3.2 (*Nondeterministic Turing machine*)

A nondeterministic Turing machine is defined by the same 6-tuple as with a deterministic Turing machine except that the program P is defined by the function:

$$P : (Q \backslash F) \times \sum \rightarrow \mathscr{P}(Q \times \sum \times \{L, R, 0\})$$

There is a distinction between a deterministic and a nondeterministic Turing machine [Bovet and Crescenzi, 2006]. In a deterministic Turing machine at most one action can be performed when in a particular state and taking a specific symbol, while in a nondeterministic Turing machine more than one action may exist for the state taking the specific symbol. In the function of a nondeterministic Turing machine, $\mathscr{P}(Q \times \sum \times \{L, R, 0\})$ defines the power set of $Q \times \sum \times \{L, R, 0\}$. The power set represents all subsets of $Q \times \sum \times \{L, R, 0\}$, which can be written as:

$$P(q, s) = \{(q'_1, s'_1, x_1), (q'_2, s'_2, x_2), ..., (q'_n, s'_n, x_n)\} \text{ for } n \geq 1$$

If $n = 1$ or $P(q, s) = \emptyset$, then the Turing machine is deterministic [Rayward-Smith, 1986; Homer and Selman, 2011].

3.5 Complexity classes

A complexity class is a collection of problems that can be accepted by a Turing machine with the same resources. The two most common complexity measures used to represent the resources used by the Turing machine are time and space [Homer and Selman, 2011].

TIME is the number of steps it takes a Turing machine to execute for input n. *SPACE* denotes the amount of Turing machine cells used by the Turing machine to execute for input n [Bovet and Crescenzi, 2006].

Definition 3.3 defines the time complexity classes for the time required by a deterministic Turing machine and a nondeterministic Turing machine to solve a decision problem [Homer and Selman, 2011]. Definition 3.4 defines deterministic and nondeterministic space complexity [Bovet and Crescenzi, 2006].

Definition 3.3 (*Time complexity classes*)

A decision problem, with an input of size n and a time-constructible function t taking time $t(n)$ steps before halting, belongs to the complexity class:

- *DTIME*$(t(n))$*, if it is solved by a (deterministic) Turing machine in time* $\mathcal{O}(t(n))$*, and*
- *NTIME*$(t(n))$*, if it is solved by a nondeterministic Turing machine which runs in time* $\mathcal{O}(t(n))$

Definition 3.4 (*Space complexity classes*)

A decision problem, with input n and a space-constructible function s that uses exactly $s(\mid n \mid)$ tape cells before halting, belongs to the complexity class:

- *DSPACE*$(s(n))$*, if it is solved by a (deterministic) multitape Turing machine using* $\mathcal{O}(s(\mid n \mid))$ *memory, and*
- *NSPACE*$(s(n))$*, if it is solved by a nondeterministic multitape Turing machine which uses memory of* $\mathcal{O}(s(\mid n \mid))$

Further discussions in this thesis relate to time-bound complexity. Definition 3.3 defines two general time-bound complexity classes, *DTIME* and *NTIME*, for solving by a Turing machine.

Two common complexity classes in polynomial time, $t(n) = n^k, k \geq 1, k \in \mathbb{N}$, are P and NP respectively. P is the abbreviation for "polynomial time", while NP is the abbreviation for "nondeterministic polynomial time". Definition 3.5 provides a formal definition of P and NP respectively in terms

of the time complexity classes given by Definition 3.3 [Homer and Selman, 2011; Drozdek, 2008].

Definition 3.5 (*Complexity classes P and NP*)

- $P = \cup\{DTIME(n^k) | k \geq 1, k \in \mathbb{N}\}$ *and*
- $NP = \cup\{NTIME(n^k) | k \geq 1, k \in \mathbb{N}\}$

From Definitions 3.3 and 3.5 it can be deduced that a decision problem belongs to complexity class P, if it can be solved in polynomial time by a deterministic Turing machine (algorithm). Likewise, a problem belongs to complexity class NP if it is solvable by a nondeterministic Turing machine (algorithm) in polynomial time. Furthermore, a deterministic Turing machine and therefore P is contained in NP, that is P $\subseteq$ NP [Drozdek, 2008].

It is still an open question whether $P = NP$ or whether P $\subset$ NP. According to [Drozdek, 2008; Harel, 1992], there is evidence suggesting that no single member of a certain subclass of NP problems can be solved deterministically in polynomial time—i.e. that only nondeterministic polynomial solutions are possible. This is the so-called NP-Complete complexity class of problems. A formal definition for the NP-Complete complexity class is given in Definition 3.6 [Drozdek, 2008; Homer and Selman, 2011]. This definition also provides a means by which a decision problem can be shown to be NP-Complete [Bovet and Crescenzi, 2006].

Definition 3.6 (*Complexity class NP-Complete (NPC)*)

A decision problem is NP-Complete, if:

- *it is in NP; and*
- *every problem in NP can polynomially be reduced to this problem.*

From the definition of NP-Complete, it is clear that :

- NP-Complete problems are contained in NP; and
- if a polynomial time algorithm were to be found for one NP-Complete problem, then there would be a polynomial time algorithm for all problems in the NP-Complete complexity class by using polynomial reduction [Harel, 1992; Drozdek, 2008]. Refer to Definition 3.7 for an explanation of polynomial reducibility.

An alternative definition for NP-Complete is to state that a decision problem is NP-Complete if it is in NP and also in the set of NP-Hard problems. Further discussion regarding NP-Hard decision problems will be presented when the definition of NP-Hard is given in Definition 3.8.

The notion of a problem being NP-Complete is very important, as many examples of problems exist that are classified as NP-Complete [Homer and Selman, 2011]. Examples of such problems are those concerned with scheduling and matching [Harel, 1992].

Polynomial time reducibility defines how one decision problem can be transformed to another decision problem in such a way that the results of the two decision problems are the same [Bang-Jensen and Gutin, 2007; Homer and Selman, 2011; Bovet and Crescenzi, 2006]. The definition is presented by Definition 3.7.

Definition 3.7 (*Polynomial time reducibility*)

> *A decision problem D_1 is polynomially reducible to a decision problem D_2, $D_1 \leq^P D_2$, if a polynomial time Turing machine (algorithm) exists that transforms each instance of decision problem D_1 to an instance of D_2*

If $D_1 \leq^P D_2$ and if the complexity of D_2 is polynomial or worse, then the complexity of D_1 is no worse than that of D_2. If complexity of D_2 is less than polynomial, then D_1's complexity may be worse than that of D_2, but it will be at worst, polynomial—as per the transformation.

Nondeterministic polynomial time hard (NP-Hard) problems are at least as hard as the hardest problems in NP. The formal definition for the NP-Hard complexity class is given by Definition 3.8 . The definition makes use of the notion of polynomial reducibility of decision problems in NP [Homer and Selman, 2011; Bang-Jensen and Gutin, 2007].

Definition 3.8 (*Complexity class NP-Hard*)

> *A decision problem D is NP-Hard if and only if all NP problems are polynomially reducible to D.*

It can be shown that if the problem is NP-hard and also belongs to NP, then it is NP-Complete [Bang-Jensen and Gutin, 2007]. It can also be shown that all NP-Complete problems are NP-Hard problems, but not all NP-Hard problems are NP-Complete.

Efficient methods to solve NP-Complete and NP-Hard problems have not as yet been found. These problems are therefore classed as intractable. Many techniques have been developed to deal with algorithms that are intractable, including the use of approximation algorithms. An approximation algorithm is an inexact way of solving the problem, that may offer guaranteed performance with close to optimal solutions [Harel, 1992; Homer and Selman, 2011].

From the discussion of complexity classes presented in this section, the following facts are known for polynomial time Turing machine (algorithms) decision problems:

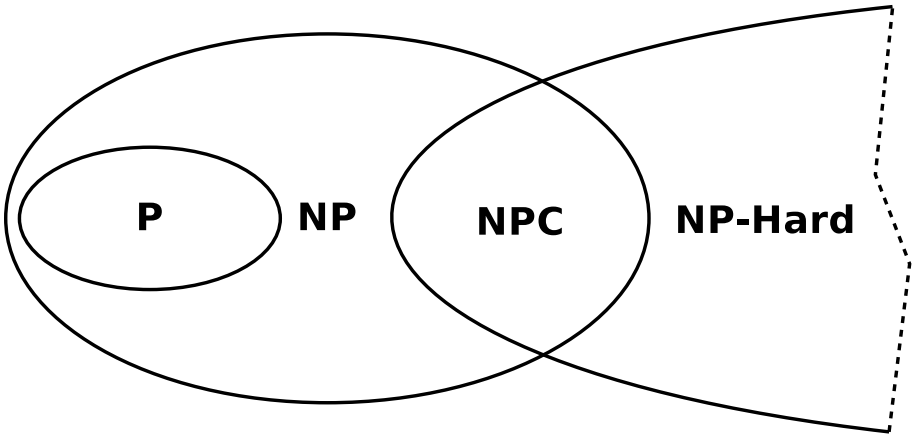


Figure 3.2: Relationship between polynomial time-based complexity classes

- P $\subseteq$ NP
- NP-Complete $\subseteq$ NP
- A decision problem D is NP-Complete if D is in NP and D is in NP-Hard

From these facts, the relationship between the polynomial time complexity classes can be diagrammatically presented as given in Figure 3.2.

3.6 Conclusion

This chapter has presented a brief overview of complexity theory in terms of the Big-$\mathcal{O}$ and complexity classes. The contents of the chapter is by no means an in-depth study of complexity theory, but provides an overview to understand the classification of the algorithms in the chapters that follow.

Chapter 4

Implementing Digraphs

4.1 Introduction

There are different techniques to implement digraphs and algorithms that can be applied to digraphs. As this chapter will be discussing the implementation of digraphs in terms of computer science concepts, it is necessary to specify the terminology to be used. Some people talk about nodes and arcs when talking about implementations of digraphs on a computer. These terms relate to vertices and edges respectively as already defined in Chapter 2. For the purposes of consistency, the mathematical terminology of vertices and edges will be used in the thesis when referring to both the graph theory and the implementation of the digraphs in a computer language.

The chapter will present an overview of implementation techniques and algorithms used in the implementation of the Graph Trans-morphism Algorithm, which is introduced in Chapter 5, and the Graph Comparison Framework, introduced in Chapter 6. The Graph Trans-morphism Algorithm makes use of a set-based representation of a graph to build a subgraph isomorphism of one graph in terms of the other. This subgraph isomorphism is used by the framework along with the original graphs for graph matching.

The implementation techniques presented in Section 4.2 will be contrasted in terms of their basic operations taking *only time complexity* into account. On the other hand, the algorithms and graph-based problems presented in Section 4.3 will be specified in terms of their *time complexity*, their *space complexity* and the respective *complexity classes* in which they fall.

4.2 Implementation techniques

There are two well-known techniques for implementing a *digraph* for computing purposes. The first is as an *adjacency matrix* and the second, by using *adjacency lists* [Bang-Jensen and Gutin, 2007]. Other techniques exist which take into consideration the disadvantages of the common techniques and try

		v_j					
		a	b	c	d	e	f
v_i	a	0	1	1	0	0	1
	b	0	0	1	0	0	1
	c	0	0	0	1	0	0
	d	0	0	0	0	1	0
	e	0	1	0	0	0	0
	f	0	0	0	0	0	0

Table 4.1: Adjacency matrix for digraph in Figure 2.2

to mitigate these for a particular situation. The majority of these techniques still rely on a basic matrix or list structure (or both) for representation.

A third technique, proposed by Barla-Szabo et al. [2004] is based on the notion of digraphs being represented as a set of triples. The technique has successfully been implemented in a toolkit called GraTe-Tk [Koopman, 2009] and will also be discussed as an implementation technique for digraphs.

4.2.1 Adjacency matrix

The *adjacency matrix* for a *digraph*, as defined in Definition 2.5, is given in Definition 4.1 [Bang-Jensen and Gutin, 2007; Diestel, 2005], with E' defined as a set of all the edge pairs of E. E' has been formally defined in Definition 2.9.

Definition 4.1 (*Adjacency matrix*)

For a digraph, $G = G(V, E)$, the adjacency matrix is an $n \times n$ matrix, where n represents the number of vertices in G ($n =\mid V \mid$). The matrix is given by $M_A(G) = [m_{A_{i,j}}], 1 \leq i, j \leq n$ where:

$$m_{A_{i,j}} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E' \\ 0 & \text{otherwise} \end{cases}$$

The *digraph* given in Figure 2.2, is represented as an adjacency matrix in Table 4.1. The resulting matrix is a 6×6 matrix with the rows representing the source vertices, $v_i \in V$, and the columns the destination vertices, $v_j \in V$. If there is an edge $(v_i, v_j) \in E'$, then $m_{A_{i,j}} = 1$. If there is no edge between the vertices then $m_{A_{i,j}} = 0$. Note that the principal diagonal of the adjacency matrix (that is $m_{A_{i,j}}$ where $i = j$) will always be 0 as the digraph definition does not allow for loops.

The problem with this representation is that for a sparse digraph with many vertices, many entries in the matrix will be 0 thereby wasting space

v_i \ e_j	(a,b)	(a,c)	(a,f)	(b,c)	(c,d)	(d,e)	(b,f)	(e,b)
a	-1	-1	-1	0	0	0	0	0
b	1	0	0	-1	0	0	-1	1
c	0	1	0	1	-1	0	0	0
d	0	0	0	0	1	-1	0	0
e	0	0	0	0	0	1	0	-1
f	0	0	1	0	0	0	1	0

Table 4.2: Incidence matrix for digraph in Figure 2.2

in the data structure representation. Even the example given in Table 4.1 only covers 8 out of the potential $(6 \times 6) = 36$ edges. This includes the six edge representations along the principal diagonal which are 0.

Incidence matrix

The adjacency matrix should not be confused with an incidence matrix, which can be used as yet another matrix-based representation. The definition of an incidence matrix is given in Definition 4.2 [Bang-Jensen and Gutin, 2007; Diestel, 2005; Drozdek, 2008]. The rows of the matrix represent the vertices in V and the columns the edges in E'.

Definition 4.2 (*Incidence matrix*)

For a digraph, $G = G(V, E)$, the incidence matrix is an $m \times n$ matrix given by $M_I(G) = [m_{I_{i,j}}]$ where $m =\mid V \mid$, $n =\mid E' \mid$, $1 \leq i \leq m, 1 \leq j \leq n$ and:

$$m_{I_{i,j}} = \begin{cases} -1 & \text{if } e_j = (v_i, x) \in E', x \in V \\ 1 & \text{if } e_j = (x, v_i) \in E', x \in V \\ 0 & \text{otherwise} \end{cases}$$

A directed edge leaving a vertex v_i, the source, is represented by -1 in the matrix. An incoming edge to a destination vertex v_i is represented by 1 and if there is no edge for the combination (v_i, x) or (x, v_i), a 0 is used. The incidence matrix representation of the digraph given in Figure 2.2 is presented by the incidence matrix in Table 4.2.

4.2.2 Adjacency list

An *adjacency list* is represented by an array of size n (n is the number of vertices of the digraph) of lists. In the majority of representations, the array represents the *source* vertex of the edge and the corresponding list all

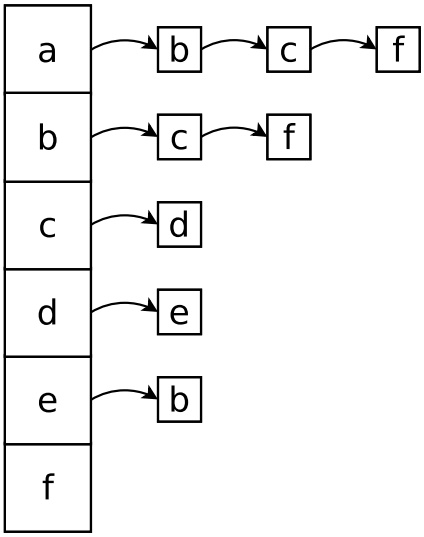


Figure 4.1: Adjacency list: source

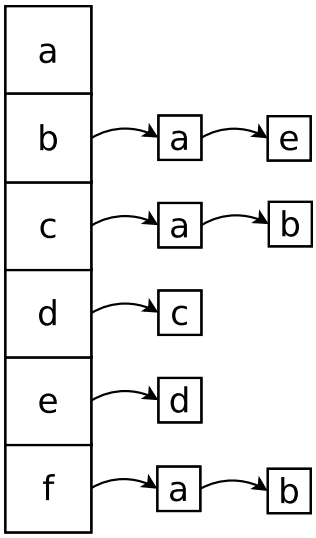


Figure 4.2: Adjacency list: destination

the *destination* vertices from that source vertex, $(source, destination) \in E'$. This maps directly to the entries with a -1 in the incidence matrix for the same graph. A second adjacency list representing the 1 entries of the incidence matrix for the particular graph will also adequately represent the graph [Bang-Jensen and Gutin, 2007]. The digraph in Figure 2.2 translates to the the adjacency list representation given in Figure 4.1 where the array of vertices represents the *source* vertices of an edge and the elements in the corresponding linked list the *destination* vertices. Figure 4.2 is the representation of *destination* vertices in the array and their corresponding *source* vertices in the lists.

4.2.3 Set of triples

Barla-Szabo [2002] defines a *digraph* as a set of arrows. Each arrow is represented by a triple comprising of the elements: *source*, *label* and *destination*. The *source* represents the start vertex of the arrow, *destination* the end vertex, and *label* the arrow nomenclature. The order of the elements of the triple are important in the definition of an arrow. An arrow begins at the *source*, has a *label* and ends at the *destination* as shown in Figure 4.3 [Barla-Szabo, 2002; Barla-Szabo et al., 2004]. The digraph in Figure 2.2

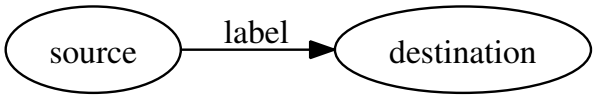


Figure 4.3: An arrow $(source, label, destination)$

translates to the following set of arrows:

$$\begin{aligned} G \quad = \quad & \{(a,(e1),b), \\ & (a,(e2),c), \\ & (a,(e3),f), \\ & (b,(e4),c), \\ & (c,(e5),d), \\ & (d,(e6),e), \\ & (b,(e7),f), \\ & (e,(e8),b)\} \end{aligned}$$

For the purposes of the discussions to be presented in this thesis, the representation of a digraph in terms of triples will be standardised to comply with the definition as presented by Definition 4.3.

Definition 4.3 (*Digraph - Set of triples*)

A digraph is characterised by a set of triples, $G = \{t_1, t_2, ..., t_n\}$.
Each triple, $t_i = (source, destination, label), 1 \leq i \leq n$, represents an edge of the graph, with $source \neq destination$.
The source and destination elements represent the start and end vertices of the edge respectively. Each edge is identified by a label.

From this definition, the graph representation of the digraph in Figure 2.2 is given as:

$$\begin{aligned} G \quad = \quad & \{(a,b,(e1)), \\ & (a,c,(e2)), \\ & (a,f,(e3)), \\ & (b,c,(e4)), \\ & (c,d,(e5)), \\ & (d,e,(e6)), \\ & (b,f,(e7)), \\ & (e,b,(e8))\} \end{aligned}$$

These two representations, $(source, label, destination)$ and $(source, destination, label)$, of a digraph are isomorphic and can therefore be used interchangeably as shown by the taxonomy of directed graph representations presented by Barla-Szabo et al. [2004, page 263].

The set of triples definition of a digraph can also be mapped onto the digraph definition presented in Definition 2.5. From Definition 4.3 it can be deduced that:

- The number of edges defined in the digraph is n. This means that $\mid E \mid$ for E as defined in Definition 2.5 must also be n.
- The set of all vertices in the graph is the union of all the source and destination vertices for all triples. The resultant set is equivalent to V as defined in Definition 2.5.
- A triple of the form $(source, destination, label)$ can be rewritten in the form $((source, destination), label)$ without loosing meaning by applying the left associative operator [Barla-Szabo et al., 2004] . This representation can subsequently be mapped to the form defined by Definition 2.5 for an edge pair in E. The first element of the pair represents the edge, $(source, destination) \in \mathcal{E}$. The second element of the pair represents the label, $label \in \mathcal{L}$.
- The definition does not allow for loops due to the requirement $source \neq destination$.
- A graph may be empty, $G = \{\} = \emptyset$. This was one of the required properties specified in Section 2.2.

It follows that the "Set of triples" representation of a digraph given by Definition 4.3 is an equivalent representation to the digraph representation as presented by Definition 2.5.

An advantage of defining a digraph as a set of triples is that well defined and behaved set operations, such as $\cup, \cap, -$ can be performed on the set. A minor disadvantage of the set of triple representation is that a graph comprising of one or more unconnected vertices cannot be represented. Each vertex needs to be connected to at least one other vertex. An empty graph can however be represented using the set of triple notation.

4.2.4 Comparison

This comparison between the three implementation techniques is on the data structure level. The focus is on the construction, initialisation and destruction of the digraph as well as on inserting, deleting and finding vertices and edges in the respective data structures. Algorithms for graph traversal, searching, matching are discussed in Section 4.3.

For the purposes of comparison a digraph comprises of v vertices and ϵ edges. Any digraph cannot have more than v^2 edges, that is $0 \leq \epsilon \leq v^2$. It is assumed that the implementation of the set data structure, used to implement the set of triples, is based on a data structure defined within the binary tree data structure classification hierarchy. Drozdek [2008] mentions that implementing a set as a red-black tree speeds up insertion and deletion to $\mathcal{O}(\log n)$. According to The C++ Resources Network [2013], the order for these two operations in the C++ STL is $\mathcal{O}(\log n)$, leading one to believe that the C++ STL makes use of some binary tree representation.

Table 4.3 presents the time complexity of data structure specific operations for each of the implementation techniques. A distinction is made between the construction of the data structure and the initialisation of the data structure. The construction of the data structure results in memory for the data structure to be allocated. The initialisation of the data structure is the action of giving the memory allocated during construction initial values. A distinction is also made between operations for inserting, deleting and finding a vertex and the operations for finding an edge [Preiss, 1998; Drozdek, 2008].

It should be noted that the table presents general Big-$\mathcal{O}$ notation orders for the different operations. It is possible to improve on the Big-$\mathcal{O}$-notation order by applying clever tweaks when implementing the algorithm. For example, finding the adjacent list of vertices in an adjacency list is $\mathcal{O}(1)$, finding a specific vertex will be $\mathcal{O}(v/\epsilon)$. The order sequence in which the vertices are specified within the data structure also plays a role, for example finding an edge between two vertices in an adjacency list is $\mathcal{O}(v/\epsilon)$ if the vertices in the vertex array are unsorted and $\mathcal{O}(\log(v/\epsilon))$ is they are sorted.

From the time complexity orders presented in Table 4.3 it can clearly be seen that the majority of the operations exhibit $\mathcal{O}(n)$ time complexity. All operations in the adjacency list implementation technique are of $\mathcal{O}(n)$ time complexity, except finding a vertex which performs in constant time. Most adjacency matrix representation operations require $\mathcal{O}(n)$ or less. Insertion, deletion and finding of edges in the adjacency matrix is a constant time lookup. It is only initialisation and destruction that performs worse and in polynomial time. If the data structure is not continuously deleted and reinstated, this performance is not a problem as it will only take place at system startup and system shutdown. The set of triples implementation fares well, since the majority of the operations are in linear time, while those that will be used the most in the basic manipulation of the data structure are in logarithmic time, $\mathcal{O}(\log n)$. Insertion and deletion of individual vertices in the set of triple implementation does not apply. Overall, the set of triple implementation technique performs better than the other two techniques for basic operations of creation, destruction, deletion, insertion and finding specific elements.

The space usage in memory of each of the implementation techniques

		Graph implementation technique		
Operation		Adjacency matrix	Adjacency list	Set of triples
construction		$\mathcal{O}(1)$	$\mathcal{O}(\upsilon)$	$\mathcal{O}(1)$
destruction		$\mathcal{O}(\upsilon^2)$	$\mathcal{O}(\upsilon + \epsilon)$	$\mathcal{O}(\epsilon)$
initialisation		$\mathcal{O}(\upsilon^2)$	$\mathcal{O}(\upsilon)$	$\mathcal{O}(\epsilon)$
insertion	vertex:	$\mathcal{O}(\upsilon)$	$\mathcal{O}(\upsilon)$	n/a
	edge:	$\mathcal{O}(1)$	$\mathcal{O}(\epsilon/\upsilon)$	$\mathcal{O}(\log \epsilon)$
deletion	vertex:	$\mathcal{O}(\upsilon)$	$\mathcal{O}(\epsilon)$	n/a
	edge:	$\mathcal{O}(1)$	$\mathcal{O}(\epsilon/\upsilon)$	$\mathcal{O}(\log \epsilon)$
find	vertex:	$\mathcal{O}(\upsilon)$	$\mathcal{O}(1)$	$\mathcal{O}(\epsilon)$
	edge:	$\mathcal{O}(1)$	$\mathcal{O}(\upsilon/\epsilon)$	$\mathcal{O}(\log \epsilon)$
next	vertex:	$\mathcal{O}(\upsilon)$	$\mathcal{O}(\upsilon)$	$\mathcal{O}(\epsilon)$
	edge:	$\mathcal{O}(\upsilon)$	$\mathcal{O}(\upsilon + \epsilon)$	$\mathcal{O}(\epsilon)$

Table 4.3: Comparison of graph operations with regards to the implementation technique

can also be compared. Space used by an adjacency matrix is $\mathcal{O}(\upsilon^2)$ while an adjacency list uses $\mathcal{O}(\upsilon+\epsilon)$. The space used by a set of triples is $\mathcal{O}(3\times\upsilon)$. In summary, adjacency lists and set of triples implementations exhibit a linear space usage order, $\mathcal{O}(n)$. The adjacency matrix implementation exhibits a quadratic space usage order, $\mathcal{O}(n^2)$.

For sparse graphs, an implementation technique with a linear space order is better. A sparse graph is a graph in which ϵ tends to be less than $\mathcal{O}(\upsilon)$ in terms of space usage [Diestel, 2005]. The situation for dense graphs, in which ϵ tends to be closer to υ^2 [Preiss, 1998], the adjacency matrix implementation technique already makes provision for representation for exactly υ^2 elements. The adjacency list implementation may be at worst having a space usage that is polynomial in order, while the set of triples would still be quadratic, but on average be worse off than the adjacency matrix implementation.

4.3 Problems and algorithms

This section mentions problems commonly associated with digraph-based structures and briefly introduces algorithms for solving those problems. The problems and algorithms are specified in terms of their space and time complexity as well as the complexity class they belong to. The algorithms under consideration are those that enable finding paths between vertices, traversing the digraphs and finding a match between two digraphs . The categories are presented in the sections that follow. Within each is a list of problems or algorithms that relate to the category.

As in the previous section, υ will represent the number of vertices in the digraph and ϵ the number of edges.

4.3.1 Finding paths and traversal

Breadth First Search (BFS) - BFS is based on the notion of beginning at a vertex and visiting all vertices that are siblings of the vertex before visiting its children. [Bang-Jensen and Gutin, 2007; Korf, 1985].

Time complexity: $\mathcal{O}(\upsilon + \epsilon)$

Space complexity: $\mathcal{O}(\upsilon)$

Complexity Class: P

Depth First Search (DFS) - DFS is based on the notion of beginning at a vertex and visiting all vertices directly reachable from the vertex before the visiting the siblings of the vertex. The algorithm makes use of backtracking [Drozdek, 2008; Bang-Jensen and Gutin, 2007; Korf, 1985].

Time complexity: $\mathcal{O}(\epsilon)$

Space complexity: $\mathcal{O}(\upsilon)$

Complexity Class: P

DFS with iterative deepening (DFSID) - The DFSID algorithm requires as parameters the source and destination vertices, as well as the digraph in which the paths are to be searched for. The output of DFSID is the set of paths between the specified source and destination vertices [Luger, 2009; Korf, 1985].

Time complexity: $\mathcal{O}(\epsilon)$

Space complexity: $\mathcal{O}(\upsilon)$

Complexity Class: P

Graph accessibility problem (GAP) - The GAP algorithm answers the question: "Is there is path between two vertices?" [Homer and Selman, 2011; Kriegel, 1986]

Time complexity: The complexity of the find edge operation was presented in Table 4.3 and is dependent on the implementation of the digraph.

Space complexity: $\mathcal{O}(log^2\upsilon)$

Complexity Class: P

Shortest path (SP) - Find the shortest path between two given vertices in the graph.

Time complexity: $\mathcal{O}(\upsilon + \epsilon)$

Space complexity: $\mathcal{O}(\upsilon^2)$

Complexity Class: P

Dijkstra's shortest path - The algorithm finds the distances from a given vertex of a weighted graph to the other vertices in the graph [Bang-Jensen and Gutin, 2007; Drozdek, 2008].

Time complexity: $\mathcal{O}(\upsilon^2)$, but when using fibonacci heaps it reduces to $\mathcal{O}(\epsilon + \upsilon \log \upsilon)$

Space complexity: $\mathcal{O}(\upsilon)$

Complexity Class: P

Travelling Salesman Problem (TSP) - Finds a hamiltonian cycle in a weighted graph with minimal cost [Bang-Jensen and Gutin, 2007; Harel, 1992; Sutcliffe, 2009].

Time complexity: $\mathcal{O}(\upsilon!)$. An approximation algorithms exist of $\mathcal{O}(\upsilon^3)$

Space complexity: $\mathcal{O}(\upsilon)$ when bounded by conditions.

Complexity Class: The classic TSP is NP-Complete, variations exist that are NP-Hard.

4.3.2 Matching

In Section 2.3 definitions for graph matching were discussed in terms of graph theory. These definitions included graph isomorphisms, graph automorphisms and subgraph isomorphisms. This section looks at graph matching problems. Graph matching problems are decision problems which ask the question whether one graph is an isomorphism, automorphism or subgraph automorphism of another. The graph automorphism problem is similar to the graph isomorphism problem in that the graphs are identical.

The subgraph isomorphism problem is a decision problem for which the question is: *For two graphs G and H, does G contain a subgraph that is isomorphic to H?*. This problem is known to be NP-Complete [Black, 2004b; Bang-Jensen and Gutin, 2007; Harel, 1992].

The graph isomorphism problem is a generalisation of the subgraph isomorphism problem and is conjectured to be NP-Complete [Aaronson et al., 2013]. This, however, is an open problem since no-one has been able to prove that is it NP-Complete [Johnson, 2005; Bang-Jensen and Gutin, 2007]. Nevertheless, it is known that the problem is contained in NP [Aaronson et al., 2013]. There are no known polynomial time algorithms that can solve the problem. There are, however, polynomial-time algorithms which are not NP-Complete that solve the problem when certain restrictions are placed on the graphs [Homer and Selman, 2011, Section 10.5]. The best time complexity

algorithm to date to solve the graph isomorphism problem is based on an algorithm by Babai and Luks [1983]. By adapting this algorithm, Zemlyachenko was able to achieve a time complexity upper bound of $2^{\mathcal{O}(\sqrt{v \log v})}$ [Johnson, 2005; Monroe, 2012].

4.4 Conclusion

This chapter provided an overview of techniques used to implement digraphs as well as algorithms and problems associated with digraphs that have a bearing on the study presented in this thesis. The techniques for digraph implementation namely, adjacency matrix, adjacency list and set of triples, were compared with each other taking both space and time complexities into account. Algorithms and problems relating to graph traversal and matching were presented along with their respective space and time complexities as well the complexity classes to which they belong.

Forthcoming chapters will show how a variant of the graph matching problem will be solved, relying on graphs represented as a set of triples and the DFSID algorithm. The variant is not an NP-Complete problem and will be solved in polynomial time. This enables the result of this study to be applied to real world contexts.

Chapter 5

Graph Trans-morphism Algorithm

5.1 Introduction

The subgraph isomorphism problem, discussed in Section 2.3.1, answers the question whether a particular graph is a subgraph isomorphism of another graph. The solution to this problem is classified as NP-Complete, thereby classifying it as a problem that is unsolvable in polynomial time. Refer to Chapter 3 for a discussion on Complexity Theory and Section 4.3.2 for a classification of the subgraph isomorphism matching algorithm.

This chapter proposes an algorithm to derive a subgraph isomorphism for a given digraph using the specifications as presented by another digraph. Instead of asking the question, "Is digraph G_M a (subgraph) isomorphism of digraph G_I?", a problem which is known to be NP-Complete, the algorithm *derives* a (subgraph) isomorphism. This (subgraph) isomorphism, digraph G_C, is derived using the information presented in digraph G_M and G_M is transformed to build digraph G_C using the structure of G_I as a template. The question then becomes a statement or assertion: "Digraph G_C is a (subgraph) isomorphism of digraph G_I, where G_C was derived by applying a transformation on G_M to structurally represent G_I.".

Prior to presenting an overview of the algorithm in Section 5.3, the terminology used when referring to the specific digraphs used by the algorithm is presented. A more detailed view of the algorithm is also presented, after which the application of the algorithm and its results are discussed in Section 5.4 by applying the algorithm to a toy application. The toy application is solely used for the purpose of explanation.

5.2 Terminology

In the introduction, three digraphs were mentioned, G_I, G_M and G_C. These digraphs are referred to as the "**I**deal", "**M**odel" and "**C**omplier" respectively and will be written as I, M and C which correspond to G_I, G_M and G_C. The ideal represents a specification to which an implementation (or approximation) of the specification, referred to as the model, should adhere. In a perfect world, the model should exactly match the ideal. In terms of graph theory, the model is then isomorphic to the ideal. In a less than perfect world, the model may be a subgraph isomorphism of the ideal. Unfortunately this is not always realistic, as the model does not necessarily represent the information of the ideal in a similar format. A transformation applied to the model, to mould it into the format as presented by the ideal, is therefore required in order for the comparison to take place. The transformed model is referred to as the complier.

The ideal, I, and the model, M, digraphs are defined within a domain of the universal set of all digraphs $\mathcal{D}$, that is $I, M \in \mathcal{D}$. As the complier, C, is a transformation of M to a representation of I in terms of structure, it follows that C is also in domain $\mathcal{D}$, that is $C \in \mathcal{D}$.

5.3 Algorithm

The algorithm provides a means to facilitate digraph matching of digraph M and digraph I. By definition digraph matching requires common vertices and a mapping, $\mathcal{F} : (v_i, v_j) \in E'_M \longrightarrow (\mathcal{F}(v_i), \mathcal{F}(v_j)) \in E'_I$, between the edges of the digraphs. This means that to enable matching with regards to the algorithm, at least the condition $V_M \subset V_I$ must be true. If this condition is not true, then there is no possibility of a match between I and M. If this condition is true and there is a $\mathcal{F}$ then M is a subgraph isomorphism of I. Refer to Definition 2.12. In many cases, due to the nature of M, there is no such obvious $\mathcal{F}$ between the edges of M and those of I. The algorithm provides a means to build this mapping by applying a series of transformations on M to build C such that it is directly comparable to I. The outcome of the algorithm is a complier for which $V_M \subset V_C$ and $V_C \subset V_I$ and it is guaranteed that there is a mapping between the edges of C and those of I. In fact this mapping results in $E'_C \subset E'_I$ making C a subgraph isomorphism of I. The representation of M in terms of C is now directly comparable to I.

The purpose of the algorithm is therefore to transform the digraph representing the model to a digraph referred to as the complier by taking the structure of the ideal into account. The premise of the algorithm is that for digraphs $I, M \in \mathcal{D}$ there exists a function $\mathcal{T} : \mathcal{D} \times \mathcal{D} \longrightarrow \mathcal{D}$, such that $C = \mathcal{T}(I, M)$, $C \in \mathcal{D}$. The algorithm therefore takes two digraphs, I and

M, as parameters and returns a third digraph, the derived complier C.

5.3.1 Algorithm overview

A high-level representation of the Graph Trans-morphism algorithm ($\mathcal{T}$) is provided by Algorithm 1.

Algorithm 1 Graph Trans-morphism Algorithm ($\mathcal{T}$) — Overview

Require: $I, M \in \mathcal{D}$
Ensure: $C \in \mathcal{D}$
1: $C = \emptyset$
2: **for** every *source* and *destination* vertex combination in M **do**
3: determine all paths in I for this combination
4: **if** paths are found in I **then**
5: add the paths to C
6: **end if**
7: **end for**
8: **return** C

Line 1 of the algorithm initialises digraph C to the empty graph. This is a valid assignment as per Definition 2.4, a generalisation of Definition 2.5. The for-loop defined by lines 2 to 7, iterates through every possible combination of *source* and *destination* vertices that have been defined in M. Using this *source* and *destination* vertex combination from M, line 3 computes all paths in I from the *source* vertex to the *destination* vertex.

Lines 4 to 6 include the paths into C if found in I. This can be achieved by applying the transformation given by Rule 5.1 for each path found to digraph C.

Rule 5.1 (*Add a path to digraph G*)

$$[v_1\ \mathcal{L}_1\ v_2\ \mathcal{L}_2\ v_3\ \ldots\ v_{i-1}\ \mathcal{L}_{i-1}\ v_i] \longrightarrow$$
$$V_G = V_G \cap \{v_1, v_2, \ldots, v_{i-1}, v_i\} \text{ and}$$
$$E_G = E_G \cap \{((v_1, v_2), \mathcal{L}_1), ((v_2, v_3), \mathcal{L}_2), \ldots, ((v_{i-1}, v_i), \mathcal{L}_{i-1})\}$$

Once all combinations of *source* and *destination* in M have been exhausted, C represents a digraph, possibly disjoint, of the information represented by M in terms of the representation defined by I. C is also in domain $\mathcal{D}$.

5.3.2 Possible outcomes of algorithm $\mathcal{T}$

One or more identifiable outcomes of algorithm $\mathcal{T}$ exists with regards to the resultant complier, C. Details of these outcomes will be presented in Chapter 7 where the outcomes will be illustrated using examples. In this

section the possible outcomes will be mentioned and a short description of each will be presented.

Outcome 1: C may have parallel edges

Using the definition of a digraph as given by Definition 2.5, it is possible that C contains multiple edges which may even be parallel. It is guaranteed that parallel edges will not have the same edge labels due to the set theoretic nature of the definitions of digraphs, both in terms of Definition 2.5 and the set of triples definition presented in Definition 4.3.

Transformation Rule 5.2 can be applied to C to remove parallel edges by concatenating the labels. If this transformation is done after the algorithm has been run, it must also be applied to I. Applying the transformation to I prior to running the algorithm, will eliminate the need to apply it to both I and C after the execution of the algorithm.

Rule 5.2 (*Remove parallel edges from digraph G*)

$$(v_i, v_j, L_1), (v_i, v_j, L_2) \longrightarrow (v_i, v_j, (L_1, L_2))$$

It is now guaranteed that C will not contain any parallel edges. The resultant digraph of C, after rewriting, will still comply with both Definition 2.5 and the set of triples digraph definition as presented in Section 4.2.3.

Outcome 2: C may comprise of disjoint digraphs

There is no guarantee that C is fully connected, implying that the complier may comprise of a number of disjoint digraphs. In many cases this problem can be solved by inserting a vertex into either or both I and M and linking them accordingly for each of the digraphs so that there is a guaranteed common vertex between the ideal and the model. If this addition of what will referred to as a "grounding vertex" does not fully resolve the complier from comprising of disjoint graphs then the model is not sufficiently commensurate[1] with the ideal. Further details regarding graph comparisons will be discussed in Chapters 6 and will be applied in Chapter 7.

Outcome 3: C may be the empty set

A complier that is an empty set, that is $C = \emptyset$, means that there are no common edges between I and M. In this case the test, as defined by graph matching in Definition 2.9, of $E'_I \cap E'_M$ producing the empty set could have been used as an indication thereof before running the algorithm. This test naturally does not take the graph labels into account and is solely based on the matching of graph shape in terms

[1] *commensurate* means corresponding in size or degree; in proportion.

of connected vertices. A stronger indication of incompatibility between I and M is if $V_I \cap V_M = \emptyset$, this would mean that there are no common vertices, let alone edges between the two digraphs.

Outcome 4: C may be an exact copy of I

A resultant complier that is an exact copy of the ideal, that is $V_C = V_I$ and $E_C = E_I$, indicates that M is fully compliant with I.

It does not follow that if C is an exact copy of I that M is an exact copy I, because:

i there might be vertices, and therefore possibly edges as well, in M that are not in I and therefore not in C either; and

ii the labels of edges in M may be different from those in I and by inference C.

Other than M being fully compliant with I, very little more can be said about M until the comparison framework has been introduced in Chapter 6. If however the following conditions were true, $V_M = V_I$ and $E'_M = E'_I$, M would be an exact copy of I, except for the possibility of a difference in edge labels. It would also not have been necessary to execute the algorithm to derive C in this case.

Outcome 5: C may be contained in I

The resultant digraph C is a subset of I, that is $V_C \subset V_I$ and $E_C \subset E_I$.

The compliance of M to I is dependent on the overlap between C and I. The quantification of this overlap will be discussed further in Chapter 6. A similar argument as with Outcome 4 holds in that it cannot be inferred that M is contained in I.

5.3.3 Algorithm detail

For this presentation of the algorithm, the graphs I, M and C are represented as a set of triples. Refer back to Section 4.2.3 for a detailed discussion of the set of triple representation for digraphs. The derivation of the sets V, E and E' used in the previous definitions for digraphs represented by Definition 2.5 are given in Definition 5.1 for the set of triple representation of a digraph defined by Definition 4.3.

Definition 5.1 (V, E and E' for a set of triples)

For any digraph G, represented as a set of triples, the sets V, E and E' can be determined by applying the respective formula that follow.

$$\begin{aligned} V &= \{u, v \mid (u, v, \mathcal{L}) \in G\} \\ E &= \{((u, v), \mathcal{L}) \mid (u, v, \mathcal{L}) \in G\} \\ E' &= \{(u, v) \mid (u, v, \mathcal{L}) \in G\} \end{aligned}$$

A more detailed version of the algorithm is presented by Algorithm 2. The algorithm comprises of two components:

i the nested loops, presented in lines 4 and 5 which generate all combinations of vertices *source* and *destination* in M; and

ii the search strategy algorithm, called in line 9, that determines all paths between the *source* and *destination* vertex combinations in I.

Algorithm 2 Graph Trans-morphism Algorithm ($\mathcal{T}$) — $C = \mathcal{T}(I, M)$

Require: $I, M \in \mathcal{D}$
Ensure: $C \in \mathcal{D}$
1: $P_{set} = \emptyset$
2: $sourceSet = \{u \mid (u, v, w) \in M\}$
3: $destinationSet = \{v \mid (u, v, w) \in M\}$
4: **for** $i = 1$ to $\mid sourceSet \mid$ **do**
5: **for** $j = 1$ to $\mid destinationSet \mid$ **do**
6: $source = sourceSet[i]$
7: $destination = destinationSet[j]$
8: **if** $source \neq destination$ **then**
9: $P_{set} = P_{set} \cup \text{DFSID}(source, destination, I)$
10: **end if**
11: **end for**
12: **end for**
13: $C = \mathscr{T}(P_{set})$
14: **return** C

Line 1 of the algorithm is equivalent to line 1 of Algorithm 1. In this case P_{set} is a placeholder for the resultant complier graph. P_{set} is a set of sets of triples. Each set of triples in P_{set} represents a path. Line 1 initialises the set of all paths P_{set} to the empty set so that each set of triples that are found to represent a path in I can be inserted into P_{set} by applying the set union, $\cup$, operator.

Lines 2 and 3 determine sets of unique vertices representing the source vertices and destination vertices of M respectively.

Lines 4 and 5 setup all combinations of *source* and *destination* vertices of M using the *sourceSet* and the *destinationSet*. Each loop iterates from 1 to the size, or cardinality, of the respective set.

The respective *source* and *destination* vertex assignments in lines 6 and 7 use the i^{th} vertex in the *sourceSet* and combines it with the j^{th} vertex in the *destinationSet* to ensure that all combinations of *source* and *destination* are covered by the search algorithm when searching through I for possible paths representing the information given by the two vertices.

Line 8 ensures that the *source* and *destination* vertices are not the same. By definition, it is not necessary to check for loops as the digraphs are defined not to have loops. This test eliminates at most $\mid V_M \mid$ calls to the search algorithm in line 9.

The search strategy applied in line 9 is Depth-First Search with Iterative Deepening (DFSID) [Luger, 2009]. A description of the algorithm was presented in Section 4.3.1. The DFSID algorithm requires as parameters two vertices, the *source* (s) and *destination* (d) vertices derived from M, as well as the digraph in which the paths are to be searched for, namely I. The output of the DFSID algorithm is a set of sets of triples with each set of triples representing a path between the specified *source* and *destination* in I. The output is unified with all previously found paths in P_{set}. Any duplicate paths will not be included in the updated P_{set} due to the nature of sets not allowing duplicate entries.

Once the execution of both for-loops has completed, the set of all paths found, P_{set}, is transformed in Line 13 from a set of sets of triples to a set of triples by applying function, $\mathscr{T}$. Function $\mathscr{T}$ is discussed in Section 5.3.4 as a graph transformation. The resultant, C, set of triples representation of a digraph is returned. This digraph is a representation of the information in M that has been found in I and moulded into I's form. The model in the form of the complier is now directly comparable to the ideal. The results of this comparison will be presented in Chapter 6.

5.3.4 Discussion in terms of graph theory

The discussion of the algorithm in terms of the graph theory, presented in Chapter 2, focusses on graph transformations and graph matching. The fundamental requirement is to be able to match I and M. In many cases, even though both digraphs are within the same domain, they are impossible to match sensibly. The algorithm, $\mathcal{T}$, derives a third digraph C which is a transformation of the information represented by M into the shape, or structure, as represented by I.

Graph transformations

The graph transformation rules being applied, do not come from a fixed set of rules. The rules are guided by the *source* and *destination* vertices defined in M and the DFSID algorithm which takes these vertices and searches for paths in I in which these vertices define the source and the destination of the path.

The most fundamental of the rules applied is to calculate the path between two given vertices in a given digraph. Multiple applications of Rule 5.3, which finds a single path between two vertices, to find all paths between the vertices matches the post-condition of the DFSID algorithm

called in Line 9 of Algorithm 2.

Rule 5.3 (*Find a path between v_i and v_j in digraph G*)

$$(v_i, v_j, \emptyset) \longrightarrow \{(v_i, v_1, L_1), (v_1, v_2, L_2), ..., (v_n, v_j, L_n)\}$$

The application of this rule to digraphs I and M, with $v_i, v_j \in V(M)$, results in a path in I, such that $P_I = [v_i, L_1, v_1, L_2, ..., v_n, L_n, v_j]$ with $v_i, v_j, v_1, ..., v_n \in V_I$ and $L_1, ..., L_n \in \mathcal{L}_I, \mathcal{L}_I \in E_I$. The length of P_I, denoted by $\mid P_I \mid$, is n where $n > 0$. This is only true if a path between the two vertices in digraph I exists. If there is no path in I between v_i and v_j, then $\mid P_I \mid = 0$. Rule 5.3 can also be seen as a recursive application of the edge subdivision rule given by Rule 2.1.

Once all paths have been found by the DFSID algorithm, they are concatenated to P_{set}. This is represented by the set union operation in Line 9 of Algorithm 2. Rule 5.4 gives this concatenation in terms of a graph transformation rule. With the digraphs being represented as sets, concatenating an empty set onto an existing set will have no effect on the existing set.

Rule 5.4 (*Join a path to graph G*)

$$G, \{(v_i, v_1, L_1), (v_1, v_2, L_2), ..., (v_n, v_j, L_n)\} \longrightarrow$$
$$G \cup \{(v_i, v_1, L_1), (v_1, v_2, L_2), ..., (v_n, v_j, L_n)\}$$

Function $\mathscr{T}$, which is called in Line 13 of Algorithm 2, can be represented by the transformation given by Rule 5.5. This transformation converts a set of sets of triples to a set of triples. Any duplicate t_{ij_k} triples are automatically removed from the resulting set. Rule 5.5 can be seen as a set of triple version of the general rule given by Rule 5.1.

Rule 5.5 (*Transform a set of sets of triples to a set of triples*)

$$\{\{t_{11_{k_1}}, t_{12_{k_1}}, \ldots, t_{1j_{k_1}}\}, \{t_{21_{k_2}}, t_{22_{k_2}}, \ldots, t_{2j_{k_2}}\}, \ldots, \{t_{i1_{k_i}}, t_{i2_{k_i}}, \ldots, t_{ij_{k_i}}\}\} \longrightarrow$$
$$\{t_{11_{k_1}}, t_{12_{k_1}}, \ldots, t_{1j_{k_1}}, t_{21_{k_2}}, t_{22_{k_2}}, \ldots, t_{2j_{k_2}}, \ldots, t_{i1_{k_i}}, t_{i2_{k_i}}, \ldots, t_{ij_{k_i}}\},$$
with $t_{ij_{k_i}} = (v_{ij_{k_i}}, u_{ij_{k_i}}, (L)_{ij_{k_i}})$
where:
i *represents the* i^{th} *set of triples in the set of sets of triples; and*
j_{k_i} *represents the* j^{th} *triple in the set of triples of* k_i *triples.*

Due to the nature of sets, it will not be necessary to apply Rule 5.2 to remove parallel edges from the resultant graph, in this case to digraph C. All labels for the edges in C have been derived from I only and therefore a duplicate triple cannot be inserted. It may however be necessary to include the labels of corresponding edges in M with those in C. Rule 5.6 presents a graph transformation rule to update labels of edges. This rule is similar to the first example rule presented in Section 2.4 used to illustrate graph transformations. It is more specific in the requirements with regards to the edge specifications.

Rule 5.6 (*Transfer the label of an edge from digraph G to digraph H*)

$(v_i, v_j, L_1) \in G, (v_i, v_j, L_2) \in H \longrightarrow (v_i, v_j, (L_2, L_1)) \in H$

Application of the rule to the results of algorithm $\mathcal{T}$ would mean that the labels of digraph C are augmented with the labels of digraph M. It is not necessary for them to be augmented with digraph I as well, since the algorithm preserves the labels of I in C.

Graph matching

Assume that I and M are representations of digraphs in the same domain. If it were to be found that $E'_I = E'_M$, then M and I would be identical except for possible differences in the labels of their respective edges. It would also follow that the two graphs I and M are isomorphic to each other, $I \cong M$. Similarly, if $E'_M \subset E'_I$ then M is a subgraph isomorphism of I. Unfortunately, as stated before, I and M may be in the same domain and may represent similar concepts, but structurally they are completely different and therefore cannot be directly matched using an exact matching technique.

The application of algorithm $\mathcal{T}$ facilitates the matching of M to I by building a digraph C which represents the information of M in terms of the structure of I. The digraphs C and I are now directly comparable using the exact graph matching techniques of isomorphism and subgraph isomorphism presented in Section 2.3. The resultant digraph C is either, the empty set, isomorphic to I or a subgraph isomorphism of I. These possible resultants of C were previously stated in Section 5.3.2 as Outcomes 3, 4 and 5 to Algorithm 1 respectively. Each resultant of C will be individually discussed in the sections that follow.

C is the empty set

If $C = \emptyset$ then there are no paths between each *source* and *destination* pair from digraph M in digraph I. There is therefore no comparison possible between M and I as neither a subgraph isomorphism nor an isomorphism of M in terms of I could be built.

C is isomorphic to I

Recall from Definition 2.9 that C is isomorphic to I if $\mid V_C \mid = \mid V_I \mid$ and there exists a function $\mathcal{F} : V_C \longrightarrow V_I$ such that $(v_1, v_2) \in E'_C \iff (\mathcal{F}(v_1), \mathcal{F}(v_2)) \in E'_I$. As discussed in Section 4.2.3, V_C and V_I can be derived from C and I as the union of all vertex elements in E'_C and E'_I respectively. Outcome 4 states that C may be an exact copy of I and therefore $\mid V_C \mid = \mid V_I \mid$. The function $\mathcal{F}$ that maps a vertex of C onto a vertex of I exists and for $v_i \in V_C$ and $\mathcal{F}(v_i) \in V_I$, $v_i = \mathcal{F}(v_i)$. For $(v_1, v_2) \in E'_C \iff (\mathcal{F}(v_1), \mathcal{F}(v_2)) \in E'_I$ it follows that $(\mathcal{F}(v_1), \mathcal{F}(v_2)) = (v_1, v_2)$. C is therefore isomorphic to I, $C \cong I$.

C a subgraph isomorphism of I

As C was derived from I using the information presented in M, the edges in C may form a subset of those in I, Outcome 5 in Section 5.3.2. It can now be established that if $E'_C \subset E'_I$ then $V_C \subset V_I$. It has already been established that if C was isomorphic to I then a function $\mathcal{F}$ exists with $(v_1, v_2) \in E'_C \iff (v_1, v_2) \in E'_I$. From Definition 2.12 it therefore follows that under the circumstances where $V_C \subset V_I$. and a function $\mathcal{F}$ exists, C is a subgraph isomorphism of I.

From Definition 2.15, it can also be said that C is homomorphic to its equivalent subgraph in I. Algorithm $\mathcal{T}$ effectively builds C to be homomorphic by preserving the edges in I in the complier. Furthermore, it can be said that C is homeomorphic, according to Definition 2.14, to a subgraph of M. This subgraph is the graph that contains the vertices that are both in C and M.

Digraph M in the form of digraph C is now directly comparable with digraph I. The comparison of digraph C with digraph I and other comparisons of interest, such as digraph M with digraph C will be discussed in Chapter 6.

5.3.5 Discussion with reference to complexity theory

It is known, as shown in Section 4.3.2, that the graph isomorphism and subgraph isomorphism problems belong to the NP-Complete or NP-Hard complexity class. This means that there is no efficient means to solve these problems.

The decision problem,"Is M a (subgraph) isomorphism of I?", has no efficient solution and an alternative technique needs to be developed to deal with the problem. In this case, algorithm $\mathcal{T}$, builds a digraph C which represents M in such a way that C is a (subgraph) isomorphism of I. The problem now becomes "C, which represents M, is a (subgraph) isomorphism of I.", if C is not the empty digraph. This problem is related to the original decision problem. The asymptotic time and space complexities of the algorithm is given in the paragraphs that follow:

The asymptotic time complexity of the algorithm presented by Algorithm 2 is given by $\mathcal{O}(n^6 \log n)$ which is the worst of all the different parts that make up the algorithm. This can be calculated as follows. As before, ϵ denotes the number of edges and υ the number of vertices:

Source and destination set initialisation — lines 2 and 3: The initialisation of each of the sets is $\mathcal{O}(\upsilon)$, the number of vertices in the *sourceSet* and *destinationSet*.

Nested loops — lines 4 and 5: These present the highest order of time complexity of $\mathcal{O}(\upsilon^2)$.

DFSID — line 9: The complexity of the DFSID algorithm is given in Section 4.3.1 and is $\mathcal{O}(\epsilon)$, where $\epsilon = \mid G \mid$, the number of triples in digraph G.

Assignment and comparison — lines 1 to 3, 6 to 9, and 13: are negligible with a time complexity of $\mathcal{O}(1)$.

Union — line 9: Using the `set_union` algorithm in C++ STL would result in a time complexity of $\mathcal{O}(\epsilon)$. The actual implementation of the algorithm appends the result of the algorithm using an iterator and the set `insert` operator which has a time complexity of $\mathcal{O}(\log \epsilon)$ as shown in Table 4.3. Inserting multiple times results in a complexity of $\mathcal{O}(\epsilon \log \epsilon)$ to simulate a union operation.

Function $\mathscr{T}$: There are $\epsilon_{P_{set}} = \sum_{i=1}^{j} k_i$ edges in total in P_{set}, using the notation from Rule 5.5. This gives a time complexity of constructing the set of triples from the set of sets of triples as $\mathcal{O}(\epsilon_{P_{set}})$ which is at worst a constant.

The overall complexity of the algorithm is derived by taking all the complexity contributions and multiplying them together. The result of this calculation is:

$$\begin{aligned} \text{Complexity of } \mathcal{T} &= \mathcal{O}(\upsilon \cdot \upsilon^2 \cdot \epsilon \cdot 1^9 \cdot \epsilon \log \epsilon \cdot \epsilon_{P_{set}}) \\ &= \mathcal{O}(\upsilon^3 \epsilon^3 \log \epsilon) \\ &\equiv \mathcal{O}(n^{3+3} \log n) \\ &= \mathcal{O}(n^6 \log n) \end{aligned}$$

As the algorithm exhibits polynomial asymptotic time complexity, the algorithm will run on a deterministic Turing Machine in polynomial time and is therefore in class P.

The asymptotic complexity of algorithm $\mathcal{T}$ in terms of space is dependent on the implementation of the underlying set data structure. The representation used by the algorithm is a a set of triples in which case there are always ϵ edges specified for the graph and each edge has a constant size k. The space complexity is given by $\mathcal{O}(\epsilon k)$ which is equivalent to $\mathcal{O}(n)$.

5.4 Explanation of the algorithm using a toy application

To illustrate how the algorithm works, consider the following two digraph specifications for I and M in domain $\mathcal{D}$ respectively.

$$\begin{aligned} I \quad = \quad & \{(a,b,(e1)),(a,c,(e2)), \\ & (a,f,(e3)),(b,c,(e4)), \\ & (c,d,(e5)),(d,e,(e6)), \\ & (b,f,(e7)),(f,g,(e8)), \\ & (g,h,(e9)),(h,i,(e10)), \\ & (h,e,(e11)),(i,d,(e12))\} \end{aligned}$$

$$\begin{aligned} M \quad = \quad & \{(a,b,(e1)),(a,d,(e2)), \\ & (d,f,(e3)),(b,f,(e4)), \\ & (b,j,(e5))\} \end{aligned}$$

5.4.1 Derivation of C using M and I by inspection

For this toy example, the matching can easily be done by inspection. Firstly by considering all edges in M and determining whether there are corresponding edges in I. There are exactly two edges that can be considered to match in terms of the definition of a subgraph isomorphism given by Definition 2.12, namely: $(a,b,(e1))$ in both I and M; and $(b,f,(e7))$ in I to $(b,f,(e4))$ in M. These matches are shown as green edges on the graphical representations of both I and M in Figures 5.1 and 5.2 respectively.

Just matching corresponding edges is not necessarily fully representative of all possible matches. So a second consideration is to calculate all combinations of *source* and *destination* vertices in M and to then inspect I to determine whether, by a process of applying consecutive edge subdivisions, there are paths between these vertices. From the graphical representations this type of inspection is easy for a human to do. It is done by finding paths between pairs of vertices. Writing down this process requires the pairs of vertices to be well defined. The discussion that follows provides a way in which a human will derive a matching graph to I for M.

The set of all *source* vertices in M is given by {a,d,b} and {b,d,f,j} for the *destination* vertices of M. The set of all combinations of *source* and *destination* pairs of vertices of M is given by the following set of pairs.

{(a,b),(a,d),(a.f),(a,j),(d,b),(d,d),(d,f),(d,j),(b,b),(b,d),(b,f),(b,j)}

The loops may be eliminated as per the definition of a digraph in Definition 4.3, leaving the following set of pairs.

{(a,b),(a,d),(a.f),(a,j),(d,b),(d,f),(d,j),(b,d),(b,f),(b,j)}.

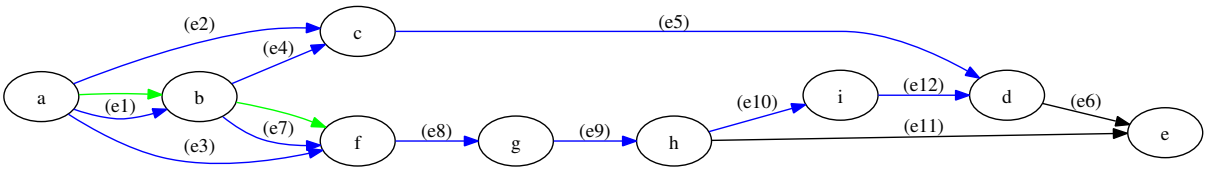


Figure 5.1: Example Ideal — $I \in \mathcal{D}$

For each pair in the set, the respective paths between the pairs in digraph I can be determined and is given by the following list of paths. The edges making up these paths are highlighted in blue in Figure 5.1.

(a, b): $P_{I_{(a,b)}} = [a\ (e1)\ b]$, an exact match.

(a, d): There are 4 paths between a and d in I. These are given by:

$P^1_{I_{(a,d)}} = [a\ (e2)\ c\ (e5)\ d]$, one edge subdivision.

$P^2_{I_{(a,d)}} = [a\ (e1)\ b\ (e4)\ c\ (e5)\ d]$, two edge subdivisions.

$P^3_{I_{(a,d)}} = [a\ (e1)\ b\ (e7)\ f\ (e8)\ g\ (e9)\ h\ (e10)\ i\ (e12)\ d]$, five edge subdivisions.

$P^4_{I_{(a,d)}} = [a\ (e3)\ f\ (e8)\ g\ (e9)\ h\ (e10)\ i\ (e12)\ d]$, four edge subdivisions.

(a, f): $P^1_{I_{(a,f)}} = [a\ (e1)\ b\ (e7)\ f]$, one edge subdivision; and $P^2_{I_{(a,f)}} = [a\ (e3)\ f]$, an exact match.

(a, j): No path, j is not in V_I.

(d, b): No path, d is a successor vertex in I to b.

(d, f): No path, d is a successor vertex in I to f.

(d, j): No path, j is not in V_I.

(b, d): There are 2 paths from b to d in I. These are given by:

$P^1_{I_{(b,d)}} = [b\ (e4)\ c\ (e5)\ d]$, one edge subdivision.

$P^2_{I_{(b,d)}} = [b\ (e7)\ f\ (e8)\ g\ (e9)\ h\ (e10)\ i\ (e12)\ d]$, four edge subdivisions.

(b, f): $P_{I_{(b,f)}} = [b\ (e7)\ f]$, an exact match.

(b, j): No path, j is not in V_I.

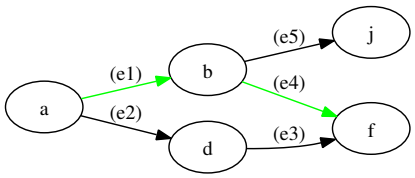


Figure 5.2: Example Model — $M \in \mathcal{D}$

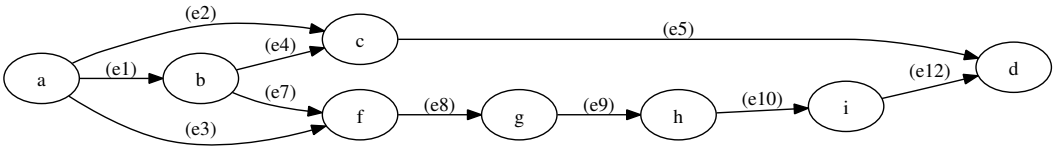


Figure 5.3: Example Complier — $C \in \mathcal{D}$

5.4.2 Building C with algorithm $\mathcal{T}$

To illustrate how algorithm, $\mathcal{T}$ defined by Algorithm 2, can be applied to digraphs I and M in order to be able to compare the information represented in M with that of I, the algorithm is called in the following manner: $C_{result} = \mathcal{T}(I, M)$, with I and M as defined above and represented by Figures 5.1 and 5.2. An iteration-by-iteration execution of the algorithm can be found in Appendix A, Section A.1. The resultant complier, C_{result}, after successful completion of the algorithm is defined by the following set of triples and represented graphically in Figure 5.3.

$$\begin{aligned} C_{result} \quad = \quad & \{(a, b, (e1)), (a, c, (e2)), (a, f, (e3)), (b, c, (e4)), (b, f, (e7)), \\ & (c, d, (e5)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\} \end{aligned}$$

By comparing the set of triple representations of the ideal and complier as well the respective figures for these graphs, Figures 5.1 and 5.3, it can clearly be seen that $V_C \subset V_I$ and $E'_C \subset E'_I$. There is also a direct mapping, a function $\mathcal{F}$, between the edges of C and those of I. The graph in Figure 5.3 representing the complier, matches the blue highlighted edges of the ideal in Figure 5.1 exactly. It can therefore be concluded that C is a subgraph isomorphism of I.

5.4.3 By inspection vs. algorithmic computation

When viewing graphs, most people think of them as a collection of paths rather than edges and vertices. This is especially true when looking for a way to "move" from one vertex to another. A possible thought process for viewing graphs in terms of paths and doing manual matching using these

paths was presented in Section 5.4.1. This method is only feasible when the graphs are small. Larger graphs of around 50 and more edges, become more complex for a human to process and an automated method, such as using an algorithm, is required.

While humans are more likely to recognise patterns that repeat easier and therefore not determine the solution again, algorithms tend to recalculate similar answers over and over again unless they have been programmed otherwise. The paths presented in Section 5.4.1 can easily be changed using a transformation to the set of triples representing the complier presented in Section 5.4.2. A transformation rule to convert a path to a set of triples is given by Rule 5.7.

Rule 5.7 (*Convert a path to a set of triples*)

$[v_1\ L_1\ v_2\ L_2\ v_3 ... v_n\ L_n\ v_{n+1}] \longrightarrow \{(v_1, v_2, L_1), (v_2, v_3, L_2), ..., (v_n, v_{n+1}, L_n)\}$, *for* $n > 1$

Applying this rule to each of the paths that were determined by inspection and calculating the union of all the resultant sets of triples results in the following digraph.

$$\{(a, b, (e1)), (a, c, (e2)), (a, f, (e3)), (b, c, (e4)), (b, f, (e7)),$$
$$(c, d, (e5)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}$$

This is the same result as what the algorithm calculates for the complier C_{result} given in Section 5.4.2. The calculations showing the transformations using Rule 5.7 and the union of the results are available in the Appendix, Section A.2, of Appendix A.

5.5 Conclusion

In this chapter an algorithm, given by Algorithm 2, has been presented which builds a (subgraph) isomorphism of an ideal given a model. A discussion regarding the possible outcomes of the Graph Transformation Algorithm which returns a (subgraph) isomorphism, referred to as the complier, was presented which will be discussed in more detail with examples in Chapter 7.

The execution of the algorithm has been illustrated using a toy application. A trace of the execution of the algorithm has been presented along with a method of solving the problem by inspection to illustrate how the algorithm works along with the results given by the algorithm. The resultant complier of the algorithm will be used for the comparison of the digraphs which will be discussed in Chapter 6.

Chapter 6

Graph Comparison Framework

6.1 Introduction

In Chapter 5 an algorithm, the Graph Trans-morphism Algorithm ($\mathcal{T}$), was presented which takes two digraphs, the ideal and model, as input and produces a digraph referred to as the complier. The complier is a representation of the information in the model in terms of the structure of the ideal. This chapter introduces a way in which these three digraphs can be compared in order to determine to what extent the model represents the ideal. To facilitate the comparison a framework is presented which is referred to as the Graph Comparison Framework.

There are many nuances to the meaning of what a framework is. The Oxford English Dictionary [OED Online, 2013] defines a framework as:

> *a. A structure made of parts joined to form a frame; esp. one designed to enclose or support; a frame or skeleton.*
> *b. In extended use: an essential or underlying structure; a provisional design, an outline; a conceptual scheme or system*

From this definition, it can be deduced that a framework is a structure that comprises of parts. For the purposes of the thesis, a framework will be viewed as a structure which can be traversed from its start-point to its end-point by visiting specific parts, which will be referred to as components. This traversal will take place in a predefined order. Each traversal of the structure will result in a specific way in which the ideal, model and complier digraphs are viewed and/or manipulated. By applying multiple traversals to the digraphs and analysing the outcomes of each, different aspects regarding the comparison of the graphs can be seen and a holistic comparison can be built.

The Graph Comparison Framework consists of two primary components: the visualisation component and the comparison component. Both these components rely on ideal and model input to the Graph Trans-morphism Algorithm, presented in Chapter 5, and the complier output calculated by the algorithm.

In the sections that follow, a high-level overview of the framework will be given. After this, a detailed discussion of each of the components will be presented. To illustrate how the framework can be used, the toy application that was introduced in Section 5.4 will be used. The application of the framework in this context will be discussed.

6.2 Framework overview

The Graph Comparison Framework makes use of the results of the Graph Trans-morphism Algorithm, $\mathcal{T}$, presented in Chapter 5. The algorithm therefore is the entry point into the framework. Both the inputs and the output of the algorithm are either further manipulated or used as is by the components, defined after $\mathcal{T}$, in the framework. Each component can therefore be seen as an entity that receives inputs and produces output. Figure 6.1 provides a high-level view of the framework showing the components of the framework and how the components are related to one another.

The components labelled Visualisation and Comparison, with the broken border, are high-level descriptors for the functionality represented by their respective child components. These function in a similar manner to interfaces in Java and virtual classes in C++ and will be referred to as virtual components.

Each of the paths through the framework provides another view of the inputs and output of the Graph Trans-morphism Algorithm. These views, independently or as a combination provide insights into the similarities and differences between digraphs I and M.

6.3 Comparison component

The comparison component of the framework presents the digraphs in such a way that they can be compared by the application of an algorithm or algorithms to the digraph representations. Application of algorithms will highlight features in the digraphs that may not be apparent by inspecting the digraph representations either visually or in their raw data form. The component can further be used to inspect and confirm or refute what may have been seen in the data or visual representation of the digraph.

The sections that follow discuss different digraph comparison techniques. The first is a way of quantifying and identifying the similarities and differences between the digraphs called the *difference comparison* component in

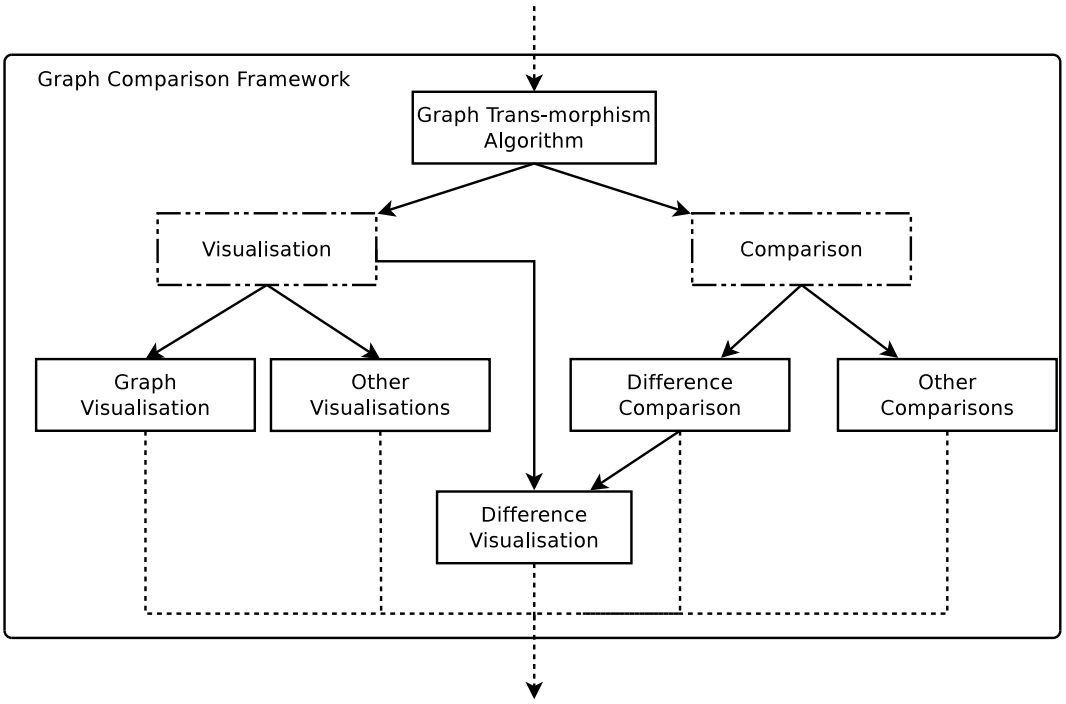


Figure 6.1: High-level view of the Graph Comparison Framework

the proposed Graph Comparison Framework. It is discussed in Section 6.3.1. Section 6.3.2, leaves room for other possible comparison components that could be applied to the digraphs before presenting an overview of another possible comparison component for digraphs, referred to as the graph edit distance.

6.3.1 Difference comparison component

The difference comparison component considers quantities related to both I, M and C as well as to a relevant selection of differences between these digraphs. The first category of quantities only considers the sets of triples for each of the digraphs. The second considers the relationships between these digraphs in order to determine to what extent they are similar or different. The difference comparison component therefore quantifies different aspects of the digraphs to enable determining the extent to which the model on the one hand, and the complier representation of the model on the other, matches the ideal.

None of the quantities take the labels of edges into account. This means that for each triple representing an edge of the digraph the label tuple is removed, leaving the edge to be characterised by a $(source, destination)$ pair.

The quantities are classified further into two categories, a quantity for the edges and quantity for the vertices representing each digraph.

Quantifying I, M and C

For any digraph G, the number of vertices and the number of edges of the digraph can be determined by calculating the cardinality of the respective sets. That is, the number of vertices in the digraph is given by $\mid V \mid$ and the number of edges by $\mid E' \mid$, where V and E' are determined as per Definition 5.1. Both these formulae will result in the number of distinct vertices and edges being calculated due to the nature of sets. Two or more parallel edges will be reckoned as a single edge. This is due to the label entity of the triple not being taken into account when determining the set of pairs, represented by E', for which the cardinality is calculated.

Applying the formulae to I, M and C will result in six cardinality-based quantities, a vertex and edge quantity per digraph. These quantities can be compared with each other to give an indication of the similarities in term of magnitude between the digraphs for vertices and edges. Cardinality-based quantities only consider relative size of the digraphs with each other in terms of vertices and edges. It it conceivable that two digraphs may have exactly the same number of vertices and/or edges and not be similar to each other at all. In order to quantify the possible matching between digraphs, it is necessary to consider the differences between the digraphs and quantify these.

Quantifying differences between I, M and C

Quantifying the difference between digraphs represented as a set of triples will make use of the set-theoretic difference of one set to another, also referred to as the relative compliment. A general definition for the set-theoretic difference between two sets is presented in Definition 6.1. Set-theoretic difference is denoted by the backslash ($\setminus$), that is the set theoretic difference of A and B is given by $A \setminus B$. An alternative notation is to use the minus-sign ($-$). This notation however gets confused with other differences and therefore the backslash will be used instead.

Definition 6.1 (*Set-theoretic difference*)

If A and B are two sets, then the set-theoretic difference of A and B is the set of elements in A, but not in B.

$$x \in A \setminus B \iff x \in A, x \notin B$$

To illustrate the application of Definiton 6.1, consider sets A and B of positive integers and the resulting set-theoretic difference sets. A is the set

of all positive integers less than 10. B the set of all positive even integers, up to and including 10.

$$\begin{aligned} A &= \{1,2,3,4,5,6,7,8,9\} \\ B &= \{2,4,6,8,10\} \\ A \setminus B &= \{1,3,5,7,9\} \\ B \setminus A &= \{10\} \end{aligned}$$

The set of all elements in A but not in B, $A \setminus B$, is the set of all odd positive integers less than 10. The set of $B \setminus A$ is the set of all elements in B which are not in A. There is only one such element, which is 10.

The set-theoretic difference when applied to two digraphs can be applied either directly on the digraph definitions when represented as a set of triples or by applying the difference on individual sets derived from the set of triple representation. Application of the set-theoretic difference on the set of triples directly is discussed in the section described as Application A below. Individual application of the set-theoretic difference on vertices and unlabeled edges is represented by Application B.

Application A: Set-theoretic difference on the digraph level

The set-theoretic difference applied to the set of triple representation of digraphs will determine all the edges in one digraph which are not in the other. When considering three digraphs, there are six possible difference combinations to take into account. Not all combinations are significant or relevant. Those of lesser significance or relevance will not be used to determine the match of the model to the ideal. The six difference combinations will be described in order to establish which are the most significant and will be used by the Difference Comparison Component.

$I \setminus M$: Represents all the information in I that is not covered by M. The closer the resultant set represents I, the less likely it is that C will provide a favourable comparison with I. If the set theoretic difference between I and M is the empty set, that is $I \setminus M = \emptyset$, then M and I are automorphic. As graph automorphism, refer to Definition 2.10, is a more extreme case of graph isomorphism, it can also be said that I is isomorphic to M. The converse is not necessarily true. If $I \cong M$ then according to Definition 2.9 for graph isomorphism, only the cardinality of the vertex sets of I and M need to be the same. This does not mean that the vertex sets of each of the graphs are identical in terms of their vertices as well.

For the purposes of the application of graph isomorphism in the framework, the graph isomorphism is refined and made more specific. To

determine graph similarities and differences using the *set of triples*, not only the vertex set cardinalities are compared, but the content of the sets as well. A partial definition for digraph isomorphism using set-theoretic differences for digraphs implemented as a *set of triples*, is given by Definition 6.2. If by Definition 2.9, $G_A \cong_\ell G_B$, then it cannot be inferred that $G_A \setminus G_B = \emptyset$. Definition 6.2 explicitly caters for when the vertex sets are exactly the same and not for the cardinalities of the sets being the same.

Definition 6.2 (*Labelled graph isomorphism — set of triples (i)*)

For two digraphs G_A and G_B, both represented by a set of triples,

$$G_A \setminus G_B = \emptyset \implies G_A \cong_\ell G_B$$

$I \setminus C$: Represents the information in I that is not in C. This will give an indication of what is in I and not represented by M. If this is the empty set, then I and C match 100% and are automorphic to each other. It further follows that M matches the information presented in I.

$M \setminus I$: All extraneous (or additional) information in M that is not required for a comparison with I. This quantity is similar to $M \setminus C$ in results. The latter quantity is deemed more informative for reasons given below and therefore $M \setminus I$ will not be used.

$M \setminus C$: All extraneous (or additional) information in M that is not required for a comparison with C. It is not necessary to look at both $M \setminus C$ and $M \setminus I$, one of the difference quantities should suffice. As C is derived from M, the most significant of the two quantities is the difference between M and C. This is because it will give an indication of what information in M is not required to build C.

$C \setminus I$: As C is derived as a (subgraph) isomorphism of I, this should always be the empty set. For comparison purposes, this difference quantity is insignificant and will be excluded.

$C \setminus M$: Information that has been added to C from I that is not reflected in M. The result here gives an indication of what could be inferred to be in M but not explicitly defined.

The difference combinations, in the order of significance, are:

$$I \setminus C,\ I \setminus M,\ M \setminus C^1 \text{ and } C \setminus M^2$$

$I \setminus C$ is the most significant difference combination because it will give the best indication of how well M, in terms of C complies with I. If the result of the difference is the empty set, then C is isomorphic to I and M complies with I. If this is not the case, then C is a subgraph isomorphism of I and the difference quantity will represent the extent to which C is a subgraph isomorphism of I.

The difference combination of $I \setminus M$ on its own only gives an indication of what has not been covered in M that is in I. This information is by no means insignificant and should not be ignored and may be used in an application domain to guide adjustments that need to be made to the model. The comparison of $I \setminus C$ with $I \setminus M$ however, will show to what extent the algorithm $\mathcal{T}$ transformed M to I in the form of C.

The difference combination of $M \setminus C$ represents the information in M and not in C and therefore not in I. This information in M is extraneous to what is in I and may need to be considered in an application domain when in the process of developing a model. Similarly, $C \setminus M$ represents the information in C (and therefore also in I) but not in M. The information represented by the resultant set of the difference combination has been inferred in C from the information present in M.

The differences for the combination $I \setminus M$ can be calculated before the execution of algorithm $\mathcal{T}$. All the other difference combinations of relevance here rely on the result of the complier.

Application B. Set-theoretic difference applied to sets of vertices and edges without labels

On a finer level of granularity, if the set-theoretic difference was applied to vertices and edges without labels independently, the results should be more interesting and useful. The set difference combinations discussed in the paragraphs referred to as Application A are still relevant and the meanings remain the same. These will be used and applied to sets of vertices and edges independently.

For each difference combination, two quantities of the difference will be calculated. The first is the difference between the vertices of each set and the second is in terms of the edge pairs which exclude the labels. Definition 6.3

[1]The difference combination $M \setminus C$ represents the information in M that is not reflected in C. This information is also not in I. As the information is neither used in C or I, it may be that it is not necessary that it is in M. This result could incorrectly be seen as *overfitting*. For the purposes of the thesis, the term ***extraneous*** will be used when referring to the information that is in M but not in C.

[2]The difference combination $C \setminus M$ represents the information in C that is not in M. For the information to be included in C there must have been a path between vertices in I beginning and ending at a vertex in M that included the additional information not in M in C. This result may incorrectly be referred to as *underfitting* of M in relation to both I and C. For the purposes of this thesis, the term ***inferred*** will be used when referring to the additional information that has been included in C that is not in M.

provides the formulae used to determine the sets of vertices and edges that are used to compute the respective difference quantities. This definition makes use of the vertex and edge sets of digraph G, that is, it makes use of V and E' respectively. When G_A (G_B respectively) is defined as a set of triples, then Definition 5.1 shows how to derive V_A and E'_A (V_B and E'_B respectively).

Definition 6.3 (*Difference sets*)

For two digraphs, G_A and G_B, the set-theoretic difference sets for vertices and edges are given by:

$$\begin{aligned} V_{G_A \setminus G_B} = V_{A \setminus B} &= V_A \setminus V_B \\ E'_{G_A \setminus G_B} = E'_{A \setminus B} &= E'_A \setminus E'_B \end{aligned}$$

The cardinality of the resultant difference sets from Definition 6.3 provides a quantification of the difference, referred to as a difference quantity, for each of the difference combinations in terms of vertices and edges.

The definition of graph isomorphism for digraphs represented as sets of triples given by Definition 6.2 can be adapted not to include edge labels as is the case in Definition 2.9. The updated definition, which is a generalised version of Definition 6.2, is presented in Defintion 6.4. From these two definitions it follows that $(G_A \cong_\ell G_B) \implies (G_A \cong_{u\ell} G_B)$.

Definition 6.4 (*Unlabelled graph isomorphism — set of triples (ii)*)

For two digraphs G_A and G_B, if

i $V_A \setminus V_B = \emptyset$; *and*

ii $E'_A \setminus E'_B = \emptyset$

$\implies G_A \cong_{u\ell} G_B$.

Quantifying the difference sets requires taking the cardinality of each of the resultant difference sets, that is $|\, V_A \setminus V_B \,|$ for vertices and $|\, E'_A \setminus E'_B \,|$ for edges.

Significance of the quantities

The quantities, calculated by taking the cardinality of the specific sets, provide size-based values for each of the sets. A comparison of set size only does not give a clear indication of the similarities and differences between the different quantities due to the varying magnitudes of the quantities. With the ideal, I, being what is being strived for, a relative quantity in terms of the ideal can be calculated, This relative quantity, referred to as a ratio, will

enable effective comparison of the quantities. Each ratio is uniquely identified by a ratio descriptor, $R(\mathcal{X}, \mathcal{Y})$. $\mathcal{X}$ represents the quantity or difference quantity for which the ratio is being described and $\mathcal{Y}$ the digraph relative to which the ratio is being calculated. The ratio is calculated by dividing the cardinality of $\mathcal{X}$ with the cardinality of $\mathcal{Y}$.

For ratios where a distinction is made between vertices and edges, the ratio descriptor comprises of two ratios, one for vertices ($V_{\mathcal{X}}^{ratio}$) and one for edges ($E_{\mathcal{X}}^{ratio}$). Equations 6.1 and 6.2 of Definition 6.5 provide the formulae to calculate each of the quantities of I, M and C in relation to I, that is for ratio descriptors $R(I, I)$, $R(M, I)$ and $R(C, I)$. Equations 6.3 and 6.4, provide the formulae for calculating the ratios for the significant set difference quantities previously discussed in relation to I.

Definition 6.5 (*Ratio formulae in terms of I*)

Formula for a single quantity in relation to I - *For a digraph G, the ratios for the vertex and edge quantities are given by:*

$$V_G^{ratio} = |\, V_G \,| \,/\, |\, V_I \,| \tag{6.1}$$
$$E_G^{ratio} = |\, E'_G \,| \,/\, |\, E'_I \,| \tag{6.2}$$

Formula for a set difference quantity in relation to I - *For digraphs G_A and G_B, the ratios for the vertex and edge difference quantities are given by:*

$$V_{A\setminus B}^{ratio} = |\, V_{A\setminus B} \,| \,/\, |\, V_I \,| \tag{6.3}$$
$$E_{A\setminus B}^{ratio} = |\, E'_{A\setminus B} \,| \,/\, |\, E'_I \,| \tag{6.4}$$

The difference sets $C \setminus M$ and $M \setminus C$ may also reveal interesting results when their respective vertex and edge cardinalities are compared in terms of a ratio of the first set specified in the combination, that is for ratio descriptors $R(C \setminus M, C)$ and $R(M \setminus C, M)$, as shown in Equations 6.5 and 6.6 in Definition 6.6.

Definition 6.6 (*Ratio formulae in terms of the first operand*)

Formula for the difference quantity in relation to the set itself - *For digraphs G_A and G_B, the ratios for the vertex and edge difference quantities are given by:*

$$V_{A\setminus B}^{ratio} = |\, V_{A\setminus B} \,| \,/\, |\, V_A \,| \tag{6.5}$$
$$E_{A\setminus B}^{ratio} = |\, E'_{A\setminus B} \,| \,/\, |\, E'_A \,| \tag{6.6}$$

The calculated values for all ratios, $R(\mathcal{X}, \mathcal{Y})$, will all be greater than or equal to 0. In most cases the value will be in the range from 0 to 1. Ratios greater than 1 could indicate that over-specification exists. For each ratio, details regarding the significance of the ratio in terms of its result is given in Appendix C. A summary of this detail is presented in Table 6.1.

Descriptor	**Description**	**Summary**
$R(I, I)$	I relative to I	The ratios for both vertices and edges will always be 1.
$R(M, I)$	M relative to I	Relative cardinality of M to I. A ratio of 1 does not necessarily mean that I and M match. It does mean that relative to one another, they are the same size.
$R(C, I)$	C relative to I	The ratio is always less than or equal to 1. If it is equal to 1 then C is isomorphic to I, otherwise C is a subgraph isomorphism of I.
$R(I \setminus C, I)$	$I \setminus C$ relative to I	C is a subgraph isomorphism of I. The resultant set will provide information regarding what C does not cover with regards to the content of I. The ratios will always be less than or equal to 1.
$R(I \setminus M, I)$	$I \setminus M$ relative to I	The ratios will always be less than or equal to 1.
$R(M \setminus C, I)$	$M \setminus C$ relative to I	A ratio that tends towards 0 is desirable. It indicates to what extent M is compatible with C and therefore I.
$R(C \setminus M, I)$	$C \setminus M$ relative to I	The ratio will always be less than or equal to 1. It should tend towards 0, thereby indicating the extent to which inferences have been made to the information in M with building C.
$R(M \setminus C, M)$	$M \setminus C$ relative to M	The ratio will always be less than or equal to 1. As the ratio tends to 0, the better the representation of M in terms of C.
$R(C \setminus M, C)$	$C \setminus M$ relative to C	$C \neq \emptyset$ must hold. A ratio closer to 0 means that C is a good representation of M. A ratio that tends towards 1 indicates the level of inference of elements in C that has taken place.

Table 6.1: Summary of the significance of the ratios $R(\mathcal{X}, \mathcal{Y})$

Ratios $R(M \setminus C, I)$ and $R(C \setminus M, I)$ provide an indication of how well M morphs to C and also whether and where any necessary adjustments should be made to M. The resultant vertex and edge sets of $M \setminus C$ provide information regarding what is in M that is not being used for the comparison. $C \setminus M$ provides information about what has been inferred as being in M

and this can be checked for validity.

6.3.2 Other comparison components

There are many other possible comparison components that can be applied to the results of the Graph Trans-morphism Algorithm. These components can easily be included in the framework thereby extending the framework. One such other component is a component that measures the number of 'edits' necessary to transform the digraph.

Graph edit distance comparison component

Graph edit distance is a generalisation of string edit distance, also referred to as Levenshtein distance [Myers et al., 2000]. Graph edit distance is an inexact graph matching technique [Zaslavskiy, 2010]. This is in contrast to the outcomes of algorithm $\mathcal{T}$ which provides a mechanism for exact matching of C and I by building a subgraph isomorphism, C, of M in terms of I.

The Levenshtein distance algorithm considers inserts, deletions and changes to characters of the source string, resulting in the target string. The application of the edit distance technique to graphs, requires inserts, deletions and changes to vertices and edges. It is also necessary to encode the graph.

Different encodings of graphs exist. Wilson and Hancock [2004] make use of a weighted adjacency matrix to encode graphs to strings. Using an adjacency matrix for two graphs, G_A and G_B in order for them to be comparable, will result in two adjacency matrix representations each of size $s \times s$ where $s = \mid V_A \cup V_B \mid$. For sparse graphs, this encoding results in a lot of computational overhead. An alternative encoding is to consider converting vertices, edges or connections to strings [Gao et al., 2010]. This encoding will work well for sparse graphs. Fiscus et al. [2006] make use of the graph edit distance to align words of sentences where the sentences are represented by DAGs with specified begin and end vertices.

The application of the Levenshtein distance algorithm to digraphs I, M and C could give an indication of the matching properties of the combinations I with M, I with C, and C with M. A distance of 0 between two digraphs that have been encoded as strings, represents an exact match. The closer the distance is to 0, the smaller the number of inserts, deletes and changes that need to be done to change one digraph to another.

The graph edit distance component, being a tried and tested technique for matching graphs, has been included for completion of the study. The technique is an inexact matching technique and provides an indication of whether graphs match or not. This information can be used to determine whether it is necessary to consider a more exact technique or not, specifically when the graph edit distance results in a bad match. For the purposes of this study a more expressive technique is required which can pinpoint at a

finer level of granularity whether there is a match and what constitutes the match.

6.4 Visualisation component

The visualisation component of the framework presents the digraphs in such a way that they can be visually compared. In many cases, as the size of the graph increases, the less likely it is to pick up similarities and differences by comparing visual representations of the entire digraph. Visualising smaller sections of the graphs is possible. As an alternative to direct graph visualisation, visualisation of the results presented in Section 6.3, the Comparison component, can be applied.

In many cases, existing graph visualisation tools can be used and will present adequate results. More specialised visualisations will require more specialised algorithms which will need to be custom written. In this section the focus is on using existing tools to visualise graphs or results from calculations applied to the graphs.

6.4.1 Graph visualisation component

The graph visualisation component makes extensive use of GraphViz which is an open source graph visualisation software suite of layout programs [AT&T Labs Research and Contributors, 2013]. Each GraphViz layout program takes as input a graph representation. The output of the program is a visual representation of the graph as specified by the layout program in a graphical file format such as: Enhanced Postscript (eps), Portable Document Format (pdf), and many more.

The layout programs in GraphViz most commonly used by the graph visualisation component are those that represent graphs in a hierarchical structure on the one hand, and those that rely on so-called "spring models" to represent graphs, on the other.

The layout program used specifically for digraphs in a hierarchical representation is called `dot`

A "spring model" views each edge as a spring and assigns it a specific strength value. This value is used to determine the length of the edge placement of the corresponding source and destination vertices. `neato` or `fds` are two layout programs in GraphViz that rely on spring models. `neato` makes use of a global energy function which needs to be minimised to determine the placement of vertices and edges in the graph. `fds` uses a force reduction function for placement. For large graphs of more than 100 vertices it is best to use `sfds` which is referred to as a "multiscale version" of `fds`.

`twopi` and `circo`, two other layout programs in the suite, present radial and circular visualisations of the graph. If no root node has been specified in `twopi`, a node is chosen at random. Examples of each of the layouts

available in GraphViz is given in Figure 6.2. More information regarding the mentioned layout programs can be found on the GraphViz website at: `http://www.graphviz.org`.

A useful visualisation strategy is to view the graph using dot from left to right to see what could be perceived as possible "entry points". Consider the digraph presented in Figure 6.2 for `dot`. From the figure is it clear that vertices p and q may be considered "entry points" to the digraph. Using this representation could be useful in identifying potential "grounding points" for the digraph when the complier is represented by a set of disjoint digraphs. Refer to Section 5.3.2 for an explanation and Section 7.3 for an example of the compliers being represented by disjoint digraphs.

If the edge from vertex q to vertex r, in Figure 6.2, is reversed and the digraph is generated again using `dot`. The result is given in Figure 6.3. The vertices p and q no longer exhibit "entry point" tendencies, but vertex a does and vertex p may be seen as a minor "entry point".

6.4.2 Difference visualisation component

The visual representation of the ratios, $R(\mathcal{X}, \mathcal{Y})$, requires a visualisation that allows for multivariate data. A radar chart (also referred to as a web, spider or star chart) is well suited for this type of data [Heerand et al., 2010]. The tool used to create the charts is the Draw tool that forms part of the Open Office suite. More information regarding Open Office can be found at: `http://www.openoffice.org`.

Each ratio representing $I, M, C, I \setminus C, I \setminus M, M \setminus C$ and $C \setminus M$, all in relation to I, will be represented by equi-angular radii, or spokes, on the radar chart. The respective vertex and edge quantities in terms of their ratios to I are plotted on the chart, resulting in two closed shapes. The relationships between the ratios can be compared to determine compliance in terms of similarities and differences.

If digraphs C, M and I are exactly the same, they are said to match perfectly. The vertices and edges charts in such a perfect match situation would have the form as shown by the dashed line in Figure 6.4. It shows ratio values of 1 for ratios representing I, M and C and values of 0 for the difference quantities such as $I \setminus M$, etc.

The solid line in Figure 6.5 represents the maximum values for each of the ratios. This information is a visual depiction of the information presented in Table 6.1. It is important to note that the ratio may exceed 1.5, but for illustration purposes the radar chart has been limited to 1.5. It is possible that the ratios $R(M, I)$ and $R(M \setminus I, I)$ exceed the value of 1. This may occur when M over specifies I and contains additional vertices and/or edges not in I. The complier will never be larger than I, it will either be exactly the same size as I, that is isomorphic to I, or it will be smaller than I, a subgraph isomorphism of I. As the complier becomes smaller in size, the

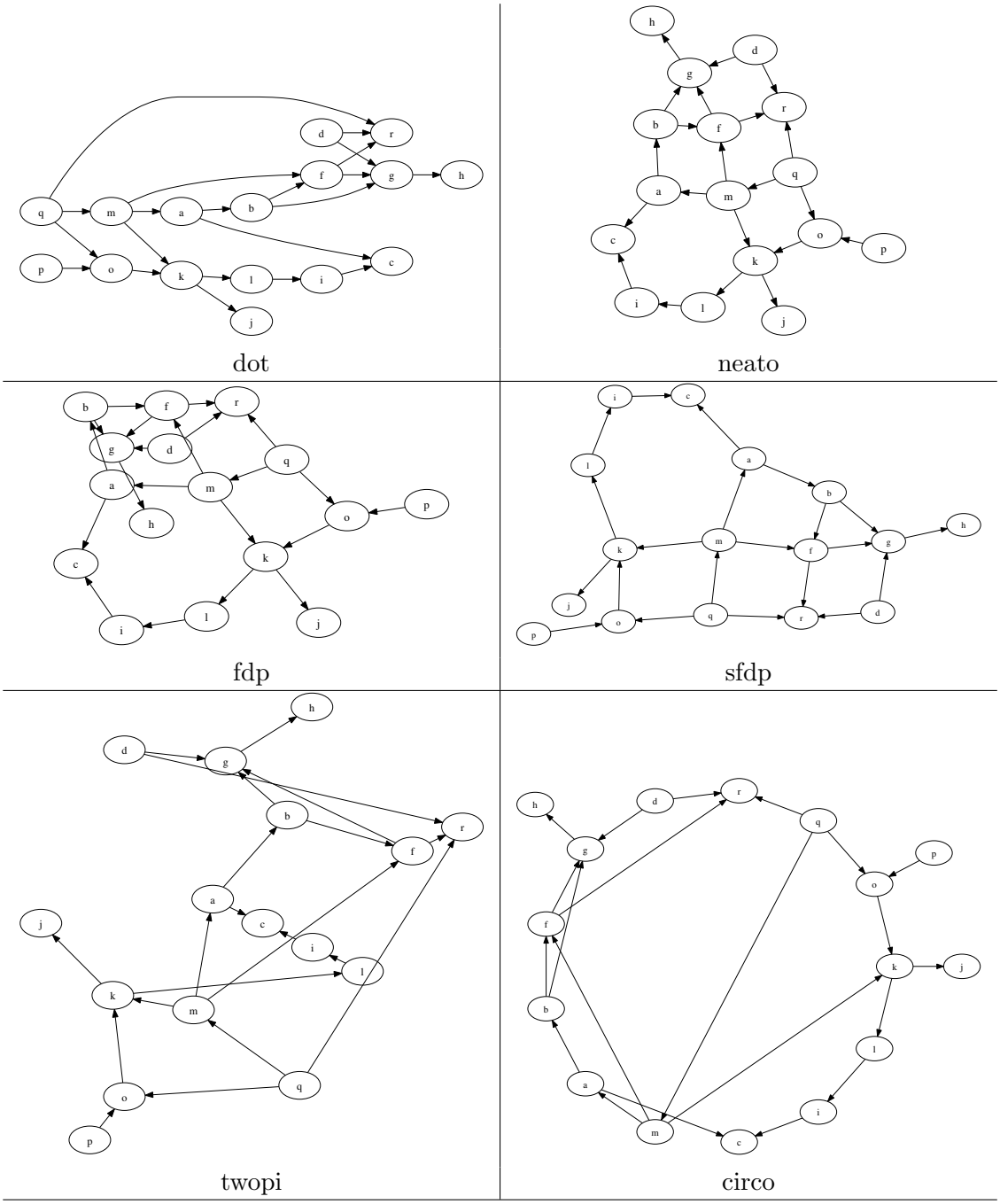


Figure 6.2: Examples of digraph layouts generated by GraphViz

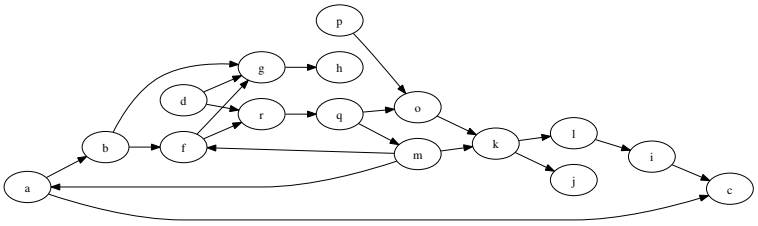


Figure 6.3: Manipulating using `dot` layout

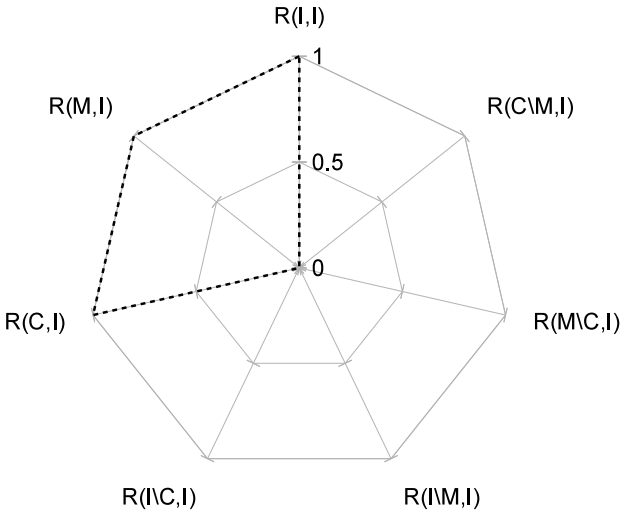


Figure 6.4: Radar chart illustrating a perfect match of ratios in relation to I

ratio $R(C, I)$ will tend towards 0 and the ratio $R(I \setminus C, I)$ will moved along the spoke in the figure towards 1. The ratio $R(I \setminus M, I)$ moves from 0 on the spoke towards 1 when there is very little that is common between I and M. The ratio $R(C \setminus M, I)$ will exhibit similar tendencies as $R(I \setminus M, I)$ but in relation to the relative size of C.

A situation may occur, as described by Outcome 4 in Section 5.3.2, where the complier is an exact copy of I yet the model is not an exact copy of I. In this situation the information represented in M maximally covers the information presented in I and the ratio $R(C, I)$ will be 1. The ratio $R(I \setminus C, I)$ will be 0, but the ratio for $R(I \setminus M)$ may be greater than 0. This situation will be illustrated in Section 7.5, Representation 2.

Using the radar chart to visualise, under perfect matching conditions, the ratios $R(M \setminus C, M)$ and $R(C \setminus M, C)$ results in a radar chart as given in Figure 6.6. The ratios $R(M, I)$ and $R(C, I)$ are also included to keep the perspective with regards to the ideal of the model and the complier.

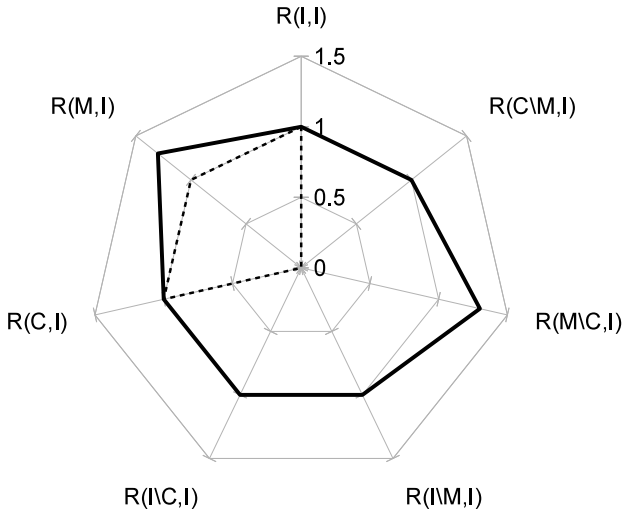


Figure 6.5: Radar chart illustrating a worst case of ratios in relation to I

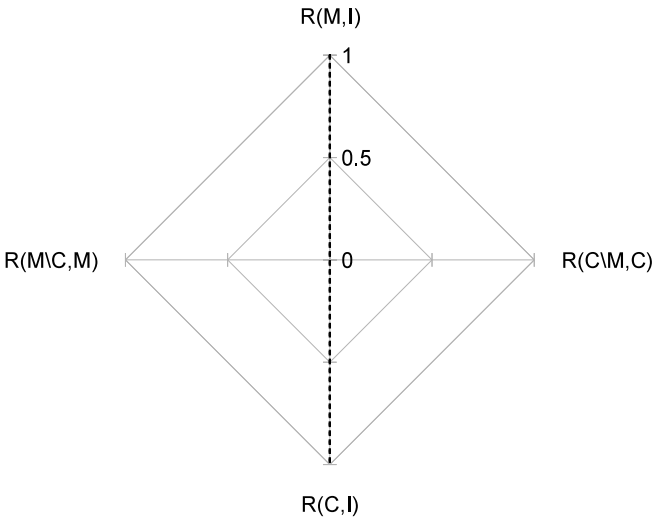


Figure 6.6: Radar chart illustrating a perfect match of ratios $R(M \setminus C, M)$ and $R(C \setminus M, C)$

6.4.3 Other visualisation components

Other than the visualisations already presented by the Graph Visualisation and Difference Visualisation components, other visualisation possibilities also exist. These additional visualisations can make use of existing tools in combination with either the algorithm or the comparison component results or both. Herman et al. [2000] presents further information visualisation techniques where the information is represented as graphs and Bennett et al. [2007] applies layout aesthetic techniques to digraphs. Both these techniques can be incorporated into components and placed in the framework to enhance the framework with regards to graph visualisation. The incorporation of techniques such as these and others is left for future work.

6.5 Applying the framework to the toy application

The application of the graph visualisation, difference comparison and difference visualisation components of the framework will be shown in terms of the toy application that was introduced in Section 5.4. In this section it is assumed that the Graph Trans-morphism Algorithm component of the framework has already been executed for the specific ideal and model digraphs.

For each of the components, what can be observed or derived from the information given will be enumerated in order to refer back to the observation if it can also be established from one of the other components.

6.5.1 Visual representation of the graphs

Visual representations of the digraphs I, M and C have already been presented in Figures 5.1, 5.2 and 5.3 respectively. These figures were generated using the `dot` layout in GraphViz. From these independent figures it is very difficult to see what the similarities and differences are between digraphs I and M except for what had already been highlighted in green. The similarities between C and I are easier to identify on the individual figures.

Figure 6.7 presents these same digraphs in terms of their respective E' sets using the `dot` layout. All three digraphs, I, M and C are drawn together in the same diagram.

From Figure 6.7 the match between the complier and the ideal can easily be seen. They match wherever the green and blue arrows, representing their respective edges, appear in parallel. A blue arrow with no parallel green arrow points to information in the ideal which is not only absent from the model, but cannot be inferred from the model and is therefore also absent from the complier. Vertex a is clearly the designated "entry point" in all the digraphs.

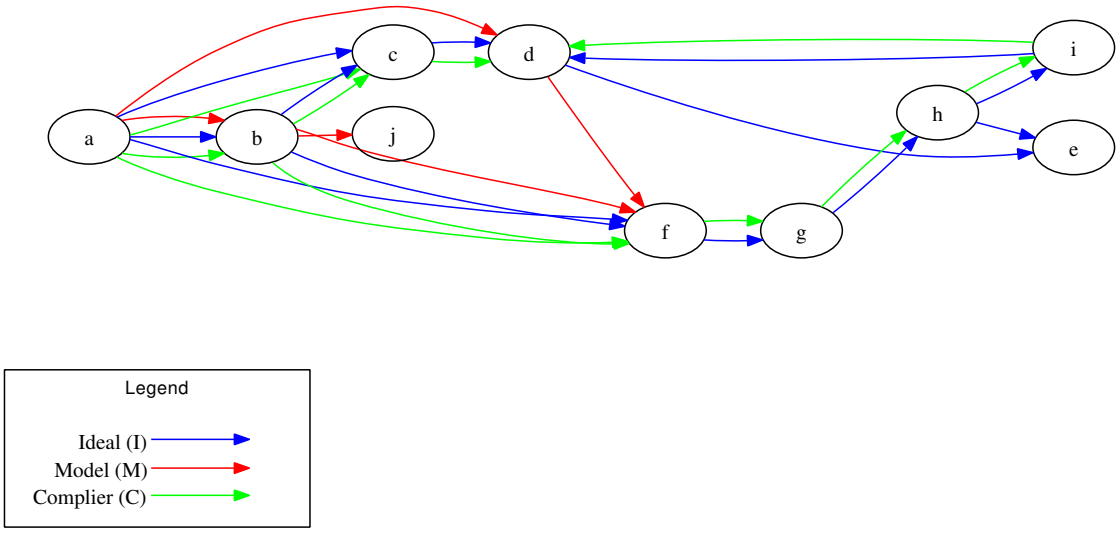


Figure 6.7: Visual comparison of I, M and C using `dot`

Redrawing the digraphs in Figure 6.7 with `twopi` and specifying a as the "entry point" or "root" in GraphViz, results in the superimposed graphs given in Figure 6.8. From this figure it can clearly be seen that vertex j is unique to the model M. Likewise, vertex e is unique to the ideal I, and so are the edges that link to e from vertices h and d.

From Figures 6.7 and 6.8 the following summarising observations can be made:

***Observation 1*- Match between I and C:** The perceived match between I and C is high as can be seen from the parallel green and blue lines.

***Observation 2*- "Entry point" node:** The natural vertex to act as "entry point" to digraphs I, M and C is vertex a.

***Observation 3*- Vertex e only in I:** Vertex e is not in M, nor is it in C and therefore neither are the edges that link it to h and d.

***Observation 4*- Vertex j is only in M:** Vertex j is not in I and therefore also not in C.

6.5.2 Results of the difference combinations

The toy application's vertex and edge sets for digraphs I, M and C are presented in Table 6.2. The table also indicates the vertices and edges of the significant set differences, namely $I \setminus C$, $I \setminus M$, $M \setminus C$ and $C \setminus M$.

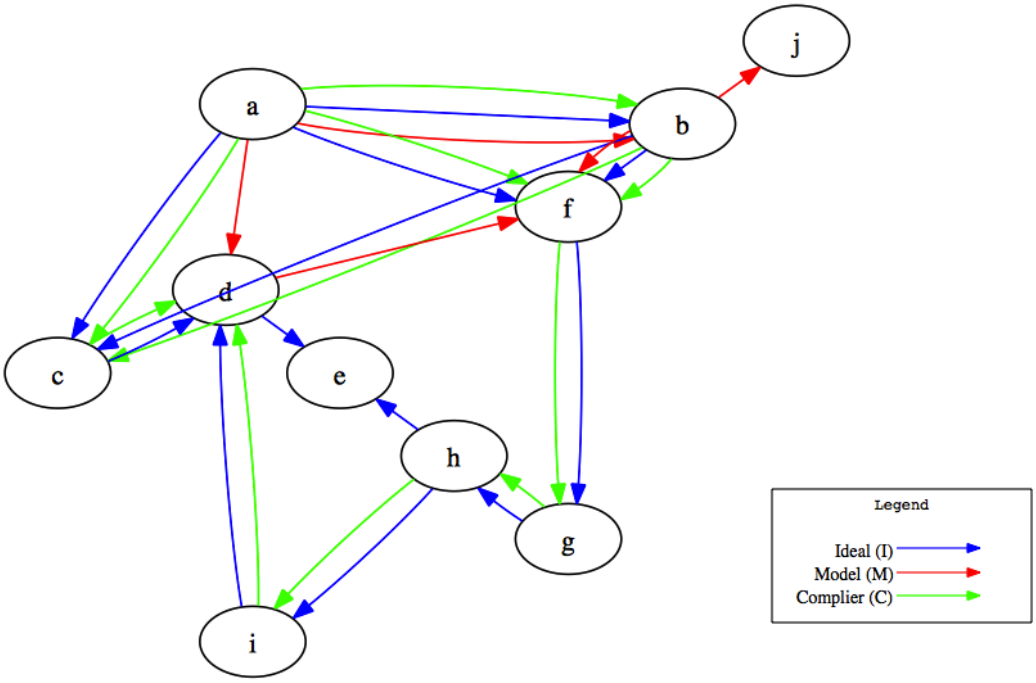


Figure 6.8: Visual comparison of I, M and C using `twopi`

Quantity	Variable	Set representation for the Toy Application
I	V_I E'_I	$\{a, b, c, d, e, f, g, h, i\}$ $\{(a, b), (a, c), (a, f), (b, c), (b, f), (c, d), (d, e), (f, g),$ $(g, h), (h, e), (h, i), (i, d)\}$
M	V_M E'_M	$\{a, b, d, f, j\}$ $\{(a, b), (a, d), (b, f), (b, j), (d, f)\}$
C	V_C E'_C	$\{a, b, c, d, f, g, h, i\}$ $\{(a, b), (a, c), (a, f), (b, c), (b, f), (c, d), (f, g),$ $(g, h), (h, i), (i, d)\}$
$I \setminus C$	$V_{I\setminus C}$ $E'_{I\setminus C}$	$\{e\}$ $\{(d, e), (h, e)\}$
$I \setminus M$	$V_{I\setminus M}$ $E'_{I\setminus M}$	$\{c, e, g, h, i\}$ $\{(a, c), (a, f), (b, c), (c, d), (d, e), (f, g),$ $(g, h), (h, e), (h, i), (i, d)\}$
$M \setminus C$	$V_{M\setminus C}$ $E'_{M\setminus C}$	$\{j\}$ $\{(a, d), (b, j), (d, f)\}$
$C \setminus M$	$V_{C\setminus M}$ $E'_{C\setminus M}$	$\{c, g, h, i\}$ $\{(a, c), (a, f), (b, c), (c, d), (f, g), (g, h), (h, i), (i, d)\}$

Table 6.2: Vertex and edge sets for the quantities

The information in this table confirms what was seen by the inspection in the digraph visualisations.

The set difference $I \setminus C$ results in the sets $V_{I \setminus C} = \{e\}$ and $E'_{I \setminus C} = \{(d, e), (h, e)\}$ as shown in Table 6.2. This can be clearly seen in Figure 6.8 and confirms *Observation 3*.

The vertex set difference of $M \setminus C$ confirms *Observation 4*, that vertex j is not an element of the complier and consequently it will not be an element of the ideal either.

The difference set $C \setminus M$ results in vertices and edges that have been inferred from the information in M in relation to I and have been included in C.

From the difference combinations, *Observations 3* and *4* can be confirmed and a further observation made, namely:

***Observation 5*- 4 vertices and 8 edges inferred from M:** A relatively large number of vertices and edges can be inferred from information in M. This can be seen from $C \setminus M$. These resultant sets are similar in cardinality to the sets of $I \setminus M$.

6.5.3 Determining the ratios

The values presented in Table 6.3 are derived from the sets in Table 6.2 by calculating the cardinalities for each of the sets. The respective cardinalities for vertices and edges for each of the quantities, represented by their ratio descriptor introduced in Table 6.1, is given in the second and third columns. The fourth and fifth columns, for the first seven rows represents the ratio of the quantity with respect to the quantity of I. Rows eight and nine present the respective quantities in terms of M and C respectively. These ratios are calculated using the formulae given by Definitions 6.5 and 6.6.

Ratio $(R(\mathcal{X}, \mathcal{Y}))$	$\mid V_{\mathcal{X}} \mid$	$\mid E'_{\mathcal{X}} \mid$	$V_{\mathcal{X}}^{ratio}$	$E_{\mathcal{X}}^{ratio}$
$R(I, I)$	9	12	1.00	1.00
$R(M, I)$	5	5	0.56	0.42
$R(C, I)$	8	10	0.89	0.83
$R(I \setminus C, I)$	1	2	0.11	0.17
$R(I \setminus M, I)$	5	10	0.56	0.83
$R(M \setminus C, I)$	1	3	0.11	0.25
$R(C \setminus M, I)$	4	8	0.44	0.67
$R(M \setminus C, M)$	1	3	0.20	0.60
$R(C \setminus M, C)$	4	8	0.50	0.80

Table 6.3: Set quantity cardinalities and ratio calculation results for the toy application

The vertex and edge ratios for $R(I, I)$ will always be 1. The ratio for M, R(M,I), merely gives an indication of M's cardinality in terms of I and not much should be read into the value. If this ratio is 1, it does not mean that M and I represent the same information. It just indicates that I and M have the same number of vertices and/or edges. A value greater than 1 will indicate that M is "bigger" than I and a value less than 1, that M is "smaller". The ratio for C, R(C,I), is more significant. It indicates the extent to which C is a subgraph isomorphism of I. This ratio quantifies what was seen by *Observation 1*.

The ratios of the difference quantities cannot be seen in isolation from one another. All ratios for difference quantities must tend towards zero and the closer they all are to zero, the better the match between I and M. *Observation 6* captures this requirement.

***Observation 6* - All set difference ratios must tend to 0:** The closer the set difference ratios are to 0 the better the match. A perfect match is indicated when all set difference ratios are 0.

The ratios for $R(I \setminus C, I)$ quantifies the difference sets for vertices and edges and what was seen in *Observation 3*. The only difference between I and C is in terms of one vertex and two edges giving favourable vertex and edge ratios.

$R(C \setminus M, I)$ confirms *Observation 5*, namely that C contains a relatively large number of edges and vertices that are not explicitly in M but have to be inferred from I. Because of this relatively high degree of inferencing, the edge and vertex sets of C tend towards becoming subsets of the corresponding edge and vertex sets of I and as a consequence, the $R(I \setminus C, I)$ ratios are relatively low.

The small ratios for $R(M \setminus C, I)$, indicates that M is not over specifying I. This relates to *Observation 4*. For small values, the individual elements of the sets can be considered and the need for them can be individually accessed. A large ratio indicates that M may not be related to the ideal at all and an investigation regarding the suitability of the comparison needs to be conducted, or M can be refined in terms of contents.

6.5.4 Plotting the ratios on a radar chart

The information presented in Table 6.3 is difficult to compare due to its format. Figure 6.9 presents the information for ratios in terms of I as a radar chart. The radar chart plots the vertex (in green), and edge (in blue) ratios with respect to I.

From Figure 6.9 it can be clearly seen that C almost fully represents I by investigating $R(I \setminus C, I)$, *Observation 4*. This however needs to be viewed in perspective with $R(C \setminus M, I)$, which is quite high for both vertices and edges

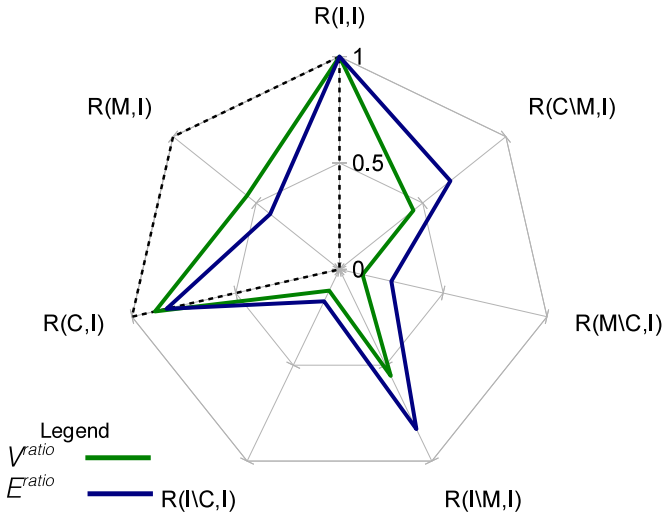


Figure 6.9: Radar chart - Toy application, R(I,I) to $R(C \setminus M, I)$

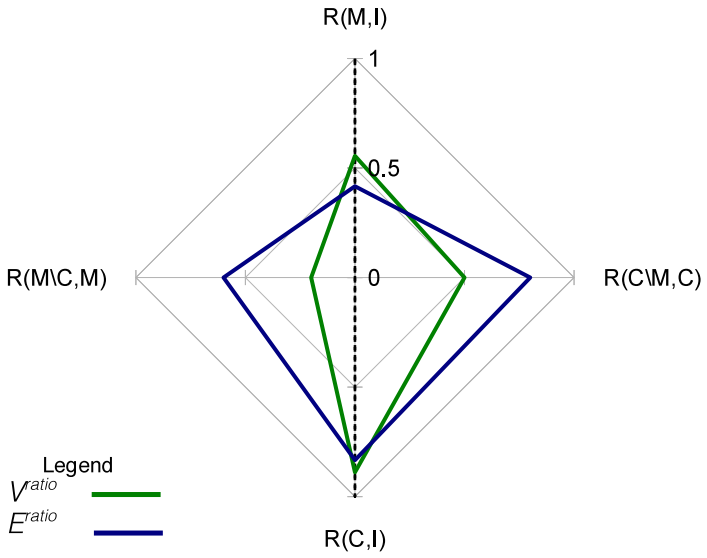


Figure 6.10: Radar chart - Toy application, R(M,I), R(C,I), $R(M \setminus C, M)$ and $R(C \setminus M, C)$

showing that quite a bit of inference took place during the construction of C, *Observation 5*. The values for $R(I \setminus M, I)$ are high, especially for edges and is given in *Observation 7* below. In this example, the under specifying nature of M resulted in greater inferences made in C. The extraneous elements in M, $R(M \setminus C, I)$, do not seem to have a major impact on the matching as the ratios are quite low.

***Observation 7* - M under specifies I:** The differences in vertices and edges of I and M indicate that there is not enough in M to match with I.

Figure 6.10 presents a similar situation for ratios R(M,I), R(C,I), $R(M \setminus C, M)$ and $R(C \setminus M, C)$. For both $R(M \setminus C, M)$ and $R(C \setminus M, C)$ a match would be indicated if the ratios were to be 0. In both cases the edge ratios are high. The vertex ratio for $R(M \setminus C, M)$ indicates less of an over specification than was initially thought when considering the information presented in Figure 6.9.

6.5.5 Considering the graph edit distance

The graph edit distance comparison component was introduced in Section 6.3.2 where it was mentioned that in order to use the Levenshtein algorithm it is necessary to encode the graphs using a string representation. The most obvious encoding to use is the adjacency matrix representation. As stated previously in Section 4.2.1, for dense graphs this method works fine, but for sparse graphs the overheads both in storage required and computation could be significant.

An alternative is to encode the edges of the digraphs as a string of vertices ordered according to their *source* and *destination* pairs. For large graphs, this representation can also result in a long string. A third method that is proposed for use to determine the graph edit distance for comparison in the toy application is to use the ordered set of edges represented by E' for each of the graphs.

The encodings of I for the toy example in terms of each of these representations are given below.

Adjacency matrix: The encoding of the adjacency matrix takes each row of the matrix and concatenates it with the next to form one long string of `0` and `1` characters. The adjacency matrix for I is given by the following 10×10 matrix. The vertices used to setup the matrix are in $V_I \cup V_M$.

	a	b	c	d	e	f	g	h	i	j
a	0	1	1	0	0	1	0	0	0	0
b	0	0	1	0	0	1	0	0	0	0
c	0	0	0	1	0	0	0	0	0	0
d	0	0	0	0	1	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	1	0	0	0
g	0	0	0	0	0	0	0	1	0	0
h	0	0	0	0	1	0	0	0	1	0
i	0	0	0	1	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0

The matrix is encoded by concatenating each row with the next which results in the following string. Each 0 and/or 1 is delimited by a space.

0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Vertex string: The string of vertices encoding requires the set E' for the digraph to be ordered, first with respect to the *source* vertex and then the *destination* vertex. E' for I is given by the following set of pairs of vertices representing the edges:

$$\begin{aligned} E'_I &= \{(a,b),(a,c),(a,f),(b,c),(b,f),(c,d),\\ &\quad (d,e),(f,g),(g,h),(h,e),(h,i),(i,d)\} \end{aligned}$$

Removing the syntax and using just the vertices from left to right will result in the encoding for I given below.

a b a c a f b c b f c d d e f g g h h e h i i d

Ordered E': The encoding for an ordered E' is similar to the string of vertices representation in that it uses E' and the ordered (*source*, *destination*) pairs. Each pair is delimited by a space to enable easy parsing of the input string. The encoding in terms of an ordered E' is given by the following string:

(a,b) (a,c) (a,f) (b,c) (b,f) (c,d) (d,e) (f,g) (g,h) (h,e) (h,i) (i,d)

The results of the graph edit distance for each of the encodings for the combinations of I and M, I and C, and M and C are given in Table 6.4.

Digraph combination	**Adjacency matrix**	**Vertex string**	**Ordered E'**
I and M	10	17	10
I and C	2	4	2
M and C	10	13	8

Table 6.4: Graph edit distance for the toy application

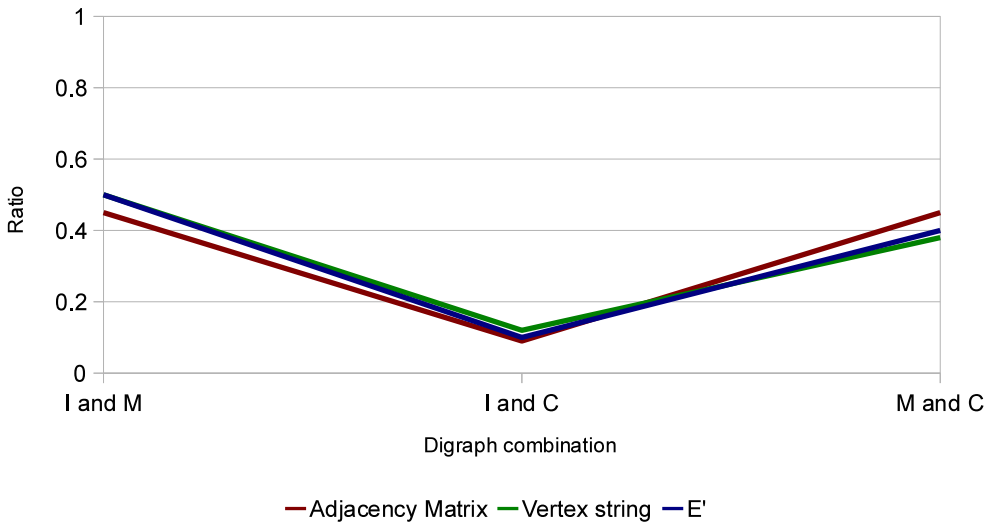


Figure 6.11: Graph edit distance ratios for the toy application

Calculating the ratio of each combination per encoding and plotting these ratios on a line graph, gives an indication of the similarities between the encodings. Figure 6.11 presents these ratios visually as a line graph.

From the line graph it is clear that there is not much difference between the ratios. This indicates that one edit distance encoding is as good as another for this application. This being the case, the execution time to determine the edit distance per encoding should be taken into consideration. As the digraphs increase in size, so will the execution time. For the adjacency matrix the execution time increase will be more dramatic than that of the other two encodings. The encoding using E' in the majority of cases should perform the best as it will always have the least number of comparisons to execute.

The "dip" in line graph given in Figure 6.11 can be attributed to C being a good representation of M in terms of I. This means that there are fewer edits — inserts, deletes and changes — required to transform I into C.

6.5.6 Interpretation

The complier digraph C that was built as a subgraph isomorphism of I by algorithm $\mathcal{T}$ matches well with the ideal I, *Observation 1*. This can be seen in the ratios for R(C,I) and $R(I \setminus C, I)$. Ratio $R(I \setminus C, I)$ also corresponds to *Observation 3*. Unfortunately, C over specifies M by inferring too much. Quantities $R(I \setminus M, I)$ and $R(C \setminus M, I)$ (and *Observation 7*) attest to this. There is not enough overlap between I and M to expect a good match, and the ratios for $R(C \setminus M, I)$ show that C is over specifying M, *Observation 5*. Both these quantities and *Observation 4* which is related to $R(M \setminus C, I)$, can be used to reconsider M and to determine how M can be restructured to decrease ratio $R(C \setminus M, I)$.

Observation 2 can only be determined by looking at the visual representations of the digraphs. *Observation 6* on the other hand can only be made by considering the ratios, either in number form as a radar chart visualisation of the numbers.

6.6 Conclusion

A framework, referred to as the Graph Comparison Framework, was presented that makes use of the inputs to and output from the Graph Transformation Algorithm introduced in Chapter 5. The framework comprises of two main components and can be extended further. These components concentrate on the visualisation of the digraphs as well as the comparison of these digraphs.

Two visualisations were discussed, the first presents the digraphs using different layouts from which similarities and differences may be spotted visually. The second makes use of the results from the difference comparison

component and represents these results on radar charts for more detailed comparison.

As with the visualisations, two comparisons were discussed. The first comparison looks at exact matching between the digraphs and presents the information regarding the differences between the digraphs as difference ratios. The second comparison component which was briefly discussed is an inexact matching technique and considers the number of edits required to match two digraphs.

All these techniques were applied to the toy application which was introduced in Chapter 5. The application of the techniques and the results particular to this application were discussed. In chapters to follow, the Graph Comparison Framework will be applied to the analysis of the outcomes of algorithm $\mathcal{T}$ (Chapter 7) and to real-world scenarios in Chapter 11.

Chapter 7

Analysis of the Outcomes of $\mathcal{T}$

7.1 Introduction

In Chapter 5 an algorithm was presented which accepts two digraphs as parameters, the model and ideal, and returns a third digraph, the complier, which is the information represented in the model in terms of the structure of the ideal. Section 5.3.2 in the chapter identified possible outcomes of the complier. Each of these identified outcomes will be discussed using simple digraphs for the ideal and model which best illustrate the outcomes of, or complier returned by, the algorithm. The discussion around each of the outcomes that are illustrated with the specific toy application ideals and models will further be guided by the Graph Comparison Framework presented in Chapter 6.

It is important to note that the outcomes of algorithm $\mathcal{T}$ are not mutually exclusive. Figure 7.1 gives all the possible combinations for the outcomes. It is for example possible for Outcomes 2, 4 and 5 to have instances of Outcome 1. Disjoint digraphs, Outcome 2, may also be subgraphs (Outcome 5) of I. In both these cases it may be necessary to take rectifying action and refine at least the model.

For each of the sections that follow the algorithm $\mathcal{T}$ will be applied to a given ideal and model in order to illustrate the possible resultant compliers for each of the outcomes. It is hoped that in the real world the majority of the resultant compliers will comply with Outcome 5 so that the techniques presented by the framework in Chapter 6 can be applied and the results thereof be used to improve the model until Outcome 4 is met. Each section, as necessary, will also discuss possible rectifying actions that may need to be taken to improve the matching of M to I.

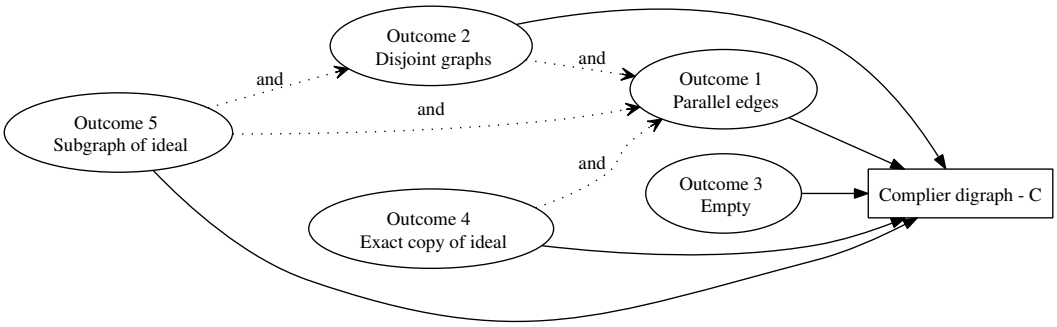


Figure 7.1: Outcome combinations of algorithm $\mathcal{T}$

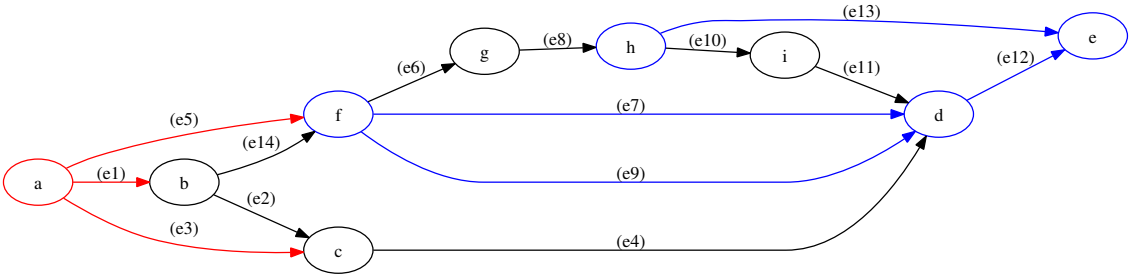


Figure 7.2: Outcome 1 - I

7.2 Outcome 1 - Parallel edges

The complier will only have parallel edges if the ideal has parallel edges that have not been removed by applying Rule 5.2 to the ideal before running the algorithm. Consider the ideal as given by Figure 7.2 with the parallel edges, $(f, d, (e7))$ and $(f, d, (e9))$, and the model as given by Figure 7.3. It is clear from the two figures that the ideal and model are not fully comparable in their current form. The only edge pairs in E' for each graph that overlap are $(f, d), (d, e)$ and (h, e). These have been marked in blue in the respective digraphs.

The subgraph, marked in blue, in M is a subgraph isomorphism of I. With some correlation between M and I, there will be at least as much correlation between C and I.

After running the algorithm, $\mathcal{T}$, the resultant complier C is given in

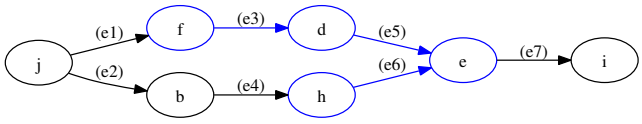


Figure 7.3: Outcome 1 - M

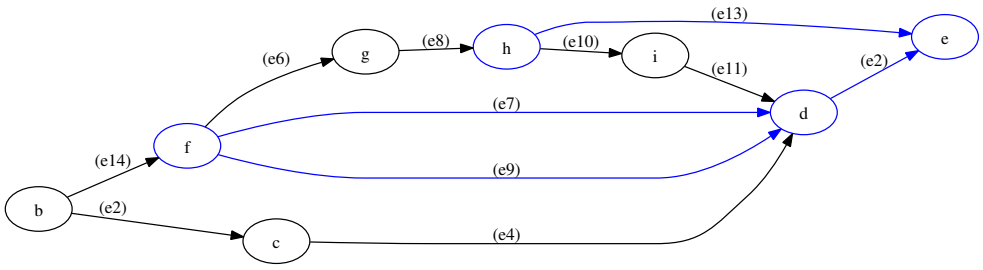


Figure 7.4: Outcome 1 - C

Figure 7.4. The edges which overlapped in I and M are also highlighted in blue in digraph C. A visual inspection of I and C reveals that there is only one vertex in I which is not in C. This vertex is a. With regards to edges, those characterised by the labels $(e1), (e3)$ and $(e5)$ are in I but not in C. For convenience, these have been highlighted in red in I. It follows, that C is a subgraph isomorphism of I because all edges between vertex b and e that are in C are also in I. This can be seen when comparing the complier in Figure 7.4 with the ideal in Figure 7.2. The information represented by the complier which is derived from the information in the model is now comparable to the ideal.

As can be seen in Figure 7.4, parallel edges in the ideal are transferred to the complier after the successful execution of algorithm $\mathcal{T}$. Application of Rule 5.2, to a digraph with parallel edges will result in a digraph with a single edge in which the labels of the edges have been concatenated. Application of the rule to the ideal before running the algorithm, will result in a complier without parallel edges. If the rule was not applied before the complier was determined, then the rule must be applied to both I and C. For the example under consideration, application of the rule to the edges $(f, d, (e9))$ and $(f, d, (e7))$ will result in a merged edge $(f, d, ((e9), (e7)))$ in the digraph being considered.

The blue edges in the figures representing I, M and C indicate edges that are common between the three digraphs. When comparing the labels of the common edges between M and either I or C, it is clear that the labels are not necessarily common between the digraphs even though the vertices may be. Application of Rule 5.6 transfers the label of an edge from one digraph and concatenates it to the label of an edge of another graph which has the same *source* and *destination* vertices but a different label. For example, the transferral of the common edge labels from M to C by the application of the rule results in updated edges between the vertices highlighted in blue in digraph C. The exact specifications of these edges are given by: (f,d,(((e9),(e7)),(e3))), (d,e,((e2),(e5))) and (h,e,((e13),(e6))) if it is assumed that the parallel edges have already been removed.

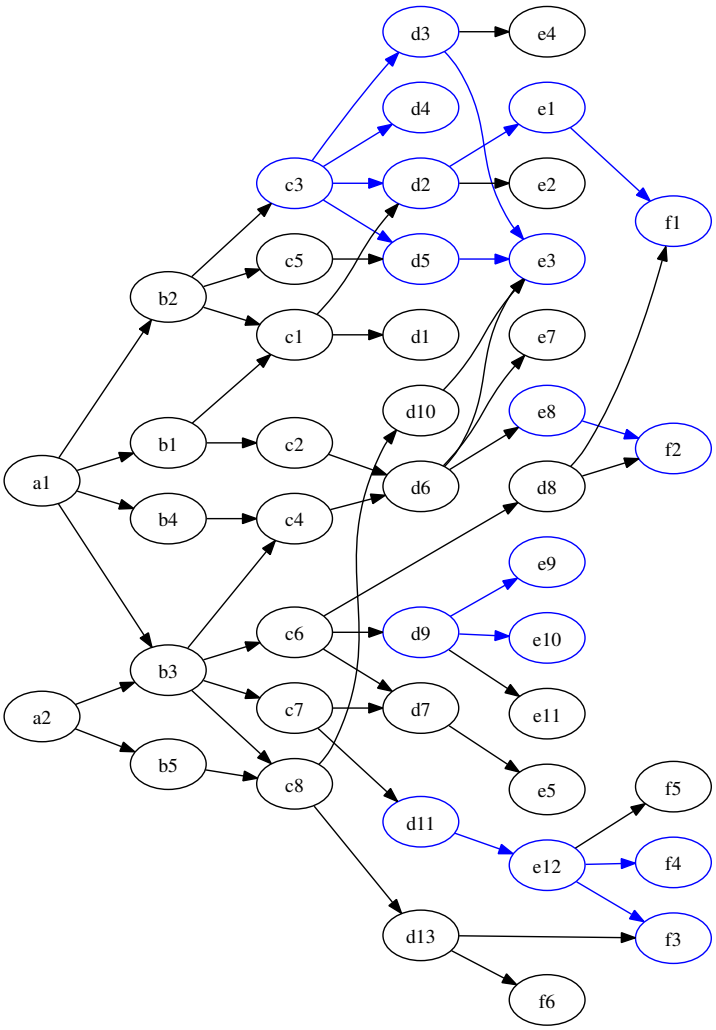


Figure 7.5: Outcome 2 - I

7.3 Outcome 2 - Disjoint digraphs

It is possible for the complier to be represented by a set of disjoint digraphs. In this case it is necessary to update at least the model and possibly re-evaluate the representation of the ideal in order to unify these disjoint digraphs without changing the information the digraphs represent. Consider the ideal and model digraphs given in Figures 7.5 and 7.6. These digraphs have been drawn using the sets E'_I and E'_M, the sets of $(source, destination)$ pairs without the labels, for I and M respectively. E'_n was defined in Definition 2.9.

After applying algorithm $\mathcal{T}$ to the ideal and model digraphs, the resultant complier digraph is given in Figure 7.7. Digraph C is a set of disjoint

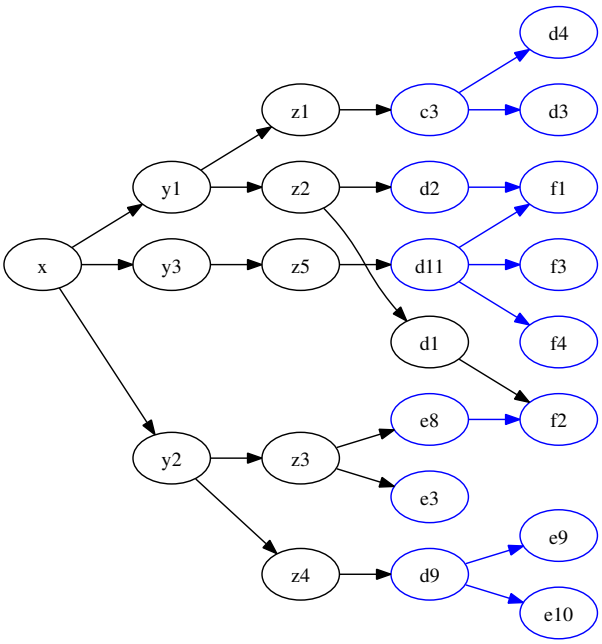


Figure 7.6: Outcome 2 - M

digraphs which map to the areas highlighted in blue on the respective I and M digraphs given in Figures 7.5 and 7.6. Each of these disjoint digraphs are subgraph isomorphisms of the ideal in their own right. The output given by the algorithm is presented in Section B.1 in the Appendix.

In some applications / contexts, a result such as this might prompt the user (of the framework) to question the accuracy of either the ideal or the model, or both. It seems that an "entry point" might be needed possibly into both the ideal and the model. This "entry point" should not detract from the information that the individual digraphs represent, but should ensure that

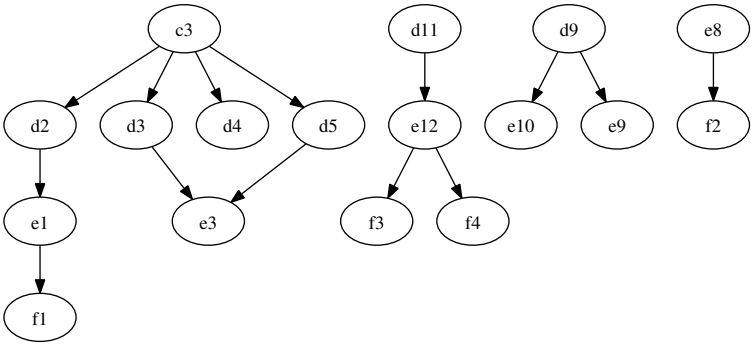


Figure 7.7: Outcome 2 - C

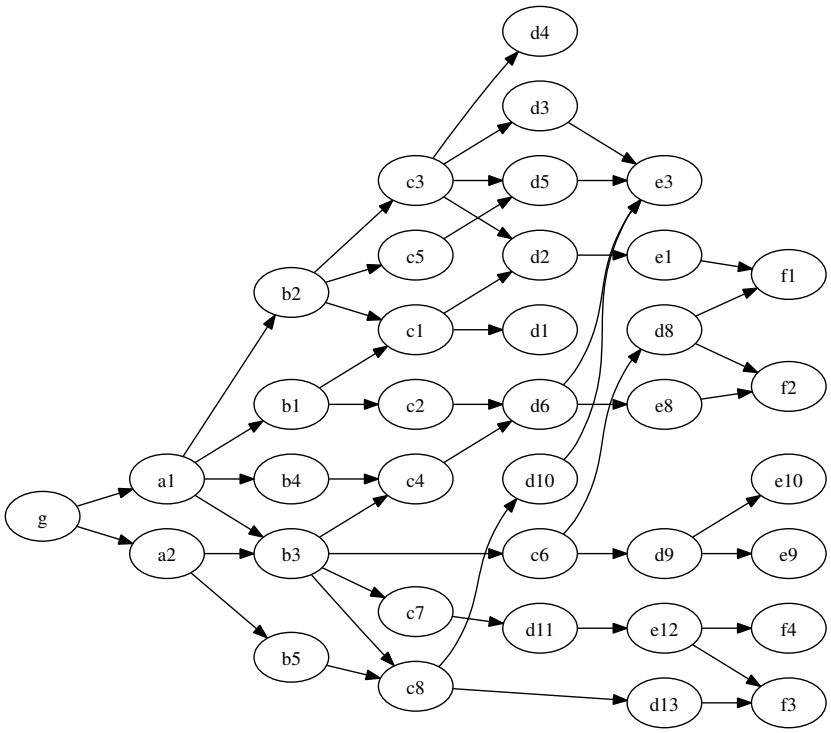


Figure 7.8: Outcome 2 - C updated

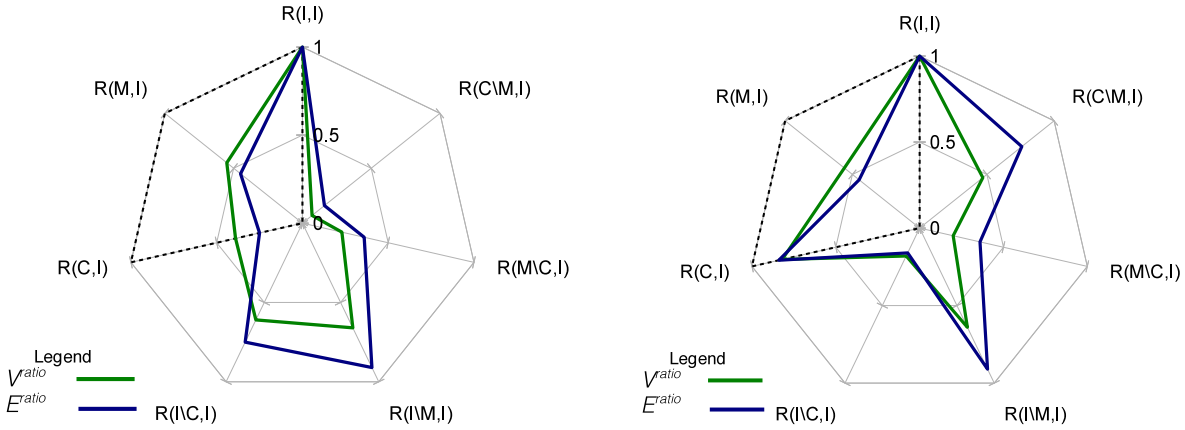
the domain, which both the ideal and model represents, is better covered.

Assume this "entry point" g is placed in I by including two edges that link node g with nodes $a1$ and $a2$ respectively. That is, the following two triples are concatenated to the representation of I, $(g, a1, \emptyset), (g, a2, \emptyset)$. A similar concatenation is applied to digraph M with the triple $(g, x, \emptyset)$ being included in the specification.

The execution of $\mathcal{T}$ with these updated I and M digraphs results in a complier as given by Figure 7.8. In some contexts, the subgraph isomorphism, C, of I might be considered to be a complier that is more representative of the ideal than the earlier version.

Applying the comparison component of the framework, particularly the difference quantities and comparing the prior to update and the updated radar charts, given in Figure 7.9, provides the following insights into the similarities and differences between the digraphs.

The addition of the "entry points" have little effect of the relationship of M to I. This can be seen by the fact that the ratio represented by $R(M, I)$ in the figures has not changed much. The ratio represented by $R(C, I)$ has shown a marked improvement after the addition of the "entry points" in I and M, as has the ratio of $R(I \setminus C, I)$. The addition of "entry points" will in some contexts result in a complier that is a more representative subgraph



Before update After update

Figure 7.9: Outcome 2 - Radar charts

isomorphism of the ideal than an earlier version without the "entry points" was.

The inclusion of the "entry points" and corresponding edges results in the ratio $R(C \setminus M, I)$ moving away from the ideal value of 0 towards 0.5 for vertices and 0.75 for edges. The difference can be seen on the radar chart in Figure 7.9 when comparing the before and after spoke for the ratio $R(C \setminus M, I)$. This is as a result of M under-representing I. The edges that result in this under-representation of M in terms of I are given by $C \setminus M$'s "After row" in Table 7.1. This row also represents the extent to which C has inferred information in M that has not been explicitly specified in M. Adapting M using this row will ensure that M better represents I.

From Table 7.1 it is evident that as C moves closer to a better match with I it moves further from matching with M. This is because C, after the update, is represented by a much larger graph. It has moved from a graph with 17 vertices and 14 edges to one with 37 vertices and 49 edges.

Inspection of the sets of $I \setminus M$ and $C \setminus M$ after the adaption to I and M has been made reveals that the sets are similar. The differences between the sets are highlighted in blue in Table 7.1.

Table 7.2 presents the results of the application of the graph edit distance to the digraphs before the update and after the update to digraphs I and M. From these values it is clear that C is a better representation of M with regards to I after the update has been made. It is also clear that the update degrades the match between M and C. This confirms the results presented in Figure 7.9 and Table 7.1.

In conclusion, C is a better representation of M in terms of I after the addition of the "entry point" vertices . Where C gained with the inclusion of the "entry point" vertices when compared with I, it lost in terms of

Quantity	Modification	Difference set
$I \setminus C$	Before	(a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5) (b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8) (b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c4,d6) (c5,d5) (c6,d7) (c6,d8) (c6,d9) (c7,d11) (c7,d7) (c8,d10) (c8,d13) (d10,e3) (d13,f3) (d13,f6) (d2,e2) (d3,e4) (d6,e3) (d6,e7) (d6,e8) (d7,e5) (d8,f1) (d8,f2) (d9,e11) (e12,f5)
	After	(c6,d7) (c7,d7) (d13,f6) (d2,e2) (d3,e4) (d6,e7) (d7,e5) (d9,e11) (e12,f5)
$I \setminus M$	Before	(a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5) (b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8) (b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d5) (c4,d6) (c5,d5) (c6,d7) (c6,d8) (c6,d9) (c7,d11) (c7,d7) (c8,d10) (c8,d13) (d10,e3) (d11,e12) (d13,f3) (d13,f6) (d2,e1) (d2,e2) (d3,e3) (d3,e4) (d5,e3) (d6,e3) (d6,e7) (d6,e8) (d7,e5) (d8,f1) (d8,f2) (d9,e11) (e1,f1) (e12,f3) (e12,f4) (e12,f5)
	After	The same as with the "Before" entry, but with the following two additional pairs: (g,a1) (g,a2)
$M \setminus C$	Before	(d1,f2) (d11,f1) (d11,f3) (d11,f4) (d2,f1) (x,y1) (x,y2) (x,y3) (y1,z1) (y1,z2) (y2,z3) (y2,z4) (y3,z5) (z1,c3) (z2,d1) (z2,d2) (z3,e3) (z3,e8) (z4,d9) (z5,d11)
	After	Exactly the same as the "Before" entry
$C \setminus M$	Before	(c3,d2) (c3,d5) (d11,e12) (d2,e1) (d3,e3) (d5,e3) (e1,f1) (e12,f3) (e12,f4)
	After	(a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5) (b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8) (b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d5) (c4,d6) (c5,d5) (c6,d8) (c6,d9) (c7,d11) (c8,d10) (c8,d13) (d10,e3) (d11,e12) (d13,f3) (d2,e1) (d3,e3) (d5,e3) (d6,e3) (d6,e8) (d8,f1) (d8,f2) (e1,f1) (e12,f3) (e12,f4) (g,a1) (g,a2)

Table 7.1: E' difference sets for outcome 2

Digraph combination	Before update	After update
I and M	54	56
I and C	42	9
M and C	22	47

Table 7.2: Graph edit distance for outcome 2

specifying M accurately. The set for $C \setminus M$ after the update provides the vertices that have been inferred in C from the information in M. Taking a closer look at these and making further changes to M will ensure that M is a better match to C. These kinds of changes requires human knowledge of the subject area that I is representing in order to modify M and increase the coverage of M in relation to C and I.

7.4 Outcome 3 - Empty resultant digraph

An empty resultant digraph, though not very likely in practice, is a theoretical possibility. Using the same digraph to represent the ideal as given in Figure 7.5, but a different model as presented in Figure 7.10, the resultant complier is the empty graph.

Again, the addition of an "entry point" will enable additional matchings to be found. The same triples are added to the ideal $((g, a1, \emptyset), (g, a2, \emptyset))$ and the model $((g, x, \emptyset))$ as were added to the ideal and model representations for outcome 2 in Section 7.3. The resultant complier for the updated I and M digraphs is given in Figure 7.11

The resultant values of the inexact matching technique represented by the graph edit distance is given in Table 7.3. The values for before update are either the cardinality of I, as seen by the first value in the first two rows, or the cardinality of M for the first value in the third row. The graph edit distances represented by the cardinality of the set was expected for the calculations which have C in them as $C = \emptyset$. The value for the first row being the cardinality of I indicates that there is very little that is similar between I and M and that it takes $\mid I \mid$ inserts, deletes or updates to convert I to M.

Digraph combination	**Before update**	**After update**
I and M	56	58
I and C	56	4
M and C	34	54

Table 7.3: Graph edit distance for outcome 3

The results for after update for the graph edit distance between I and M are as expected. It remains the cardinality of I where I now has two additional edges thereby explaining the difference of 2 between the after and before values. There is an improvement between I and C from before to after update. Before the update required 56 edits to transform I to C, while after the update only 4 edits were required. This indicates that the complier matches the ideal quite well. The situation is not that good between the

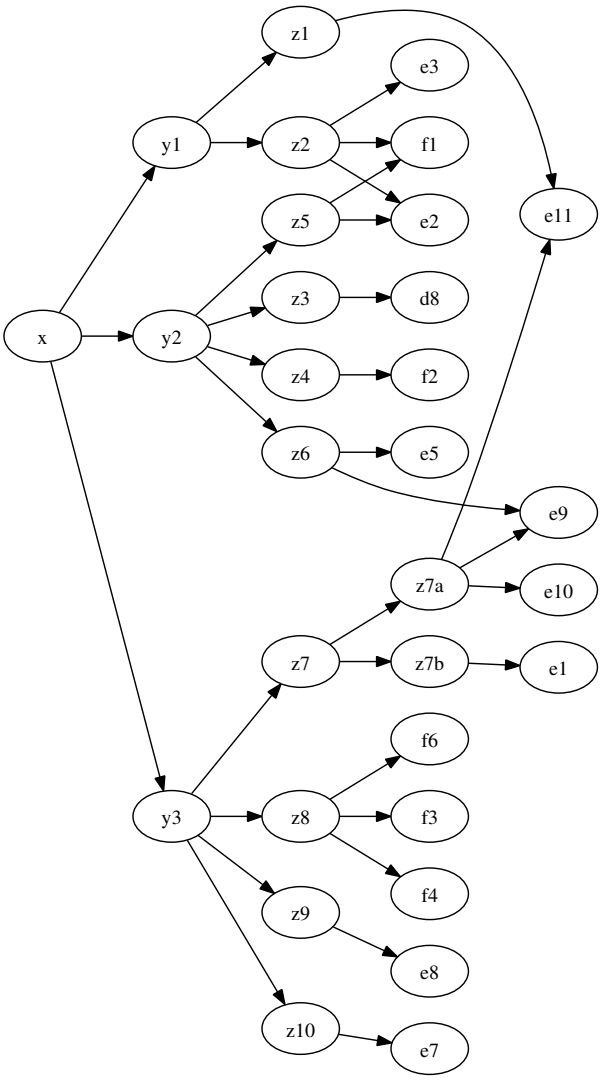


Figure 7.10: Outcome 3 - M

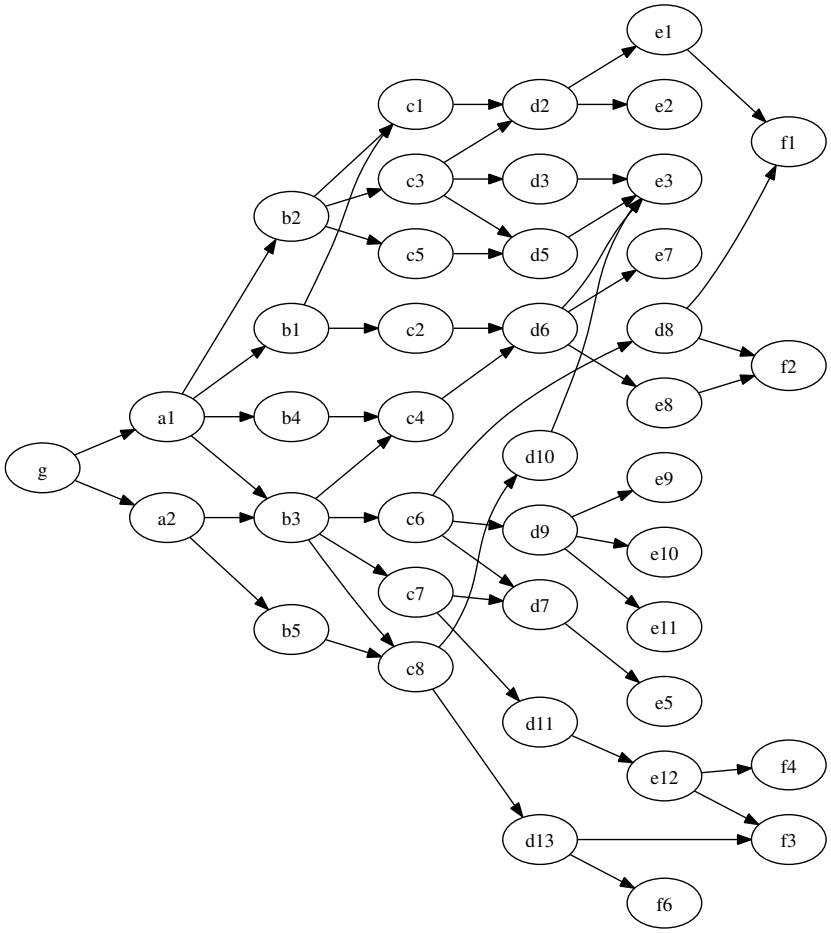
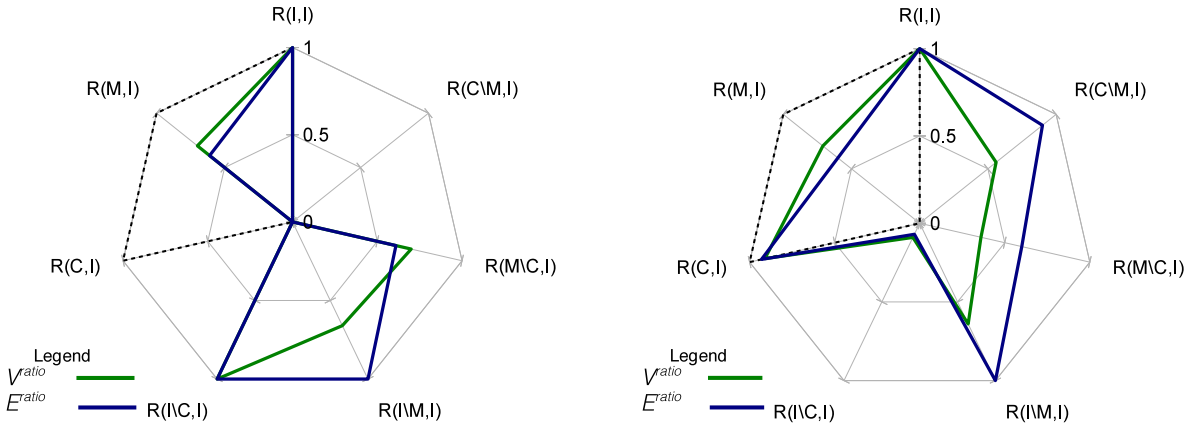


Figure 7.11: Outcome 3 - C updated



Before update After update

Figure 7.12: Outcome 3 - Radar charts

model and the complier.

Applying the comparison component of the framework, particularly the difference quantities and comparing the prior to update and the updated radar charts, given in Figure 7.12, provides the following insights into the similarities and differences between the digraphs.

The addition of the "entry point" has not made much difference between the ratios of before and after for $R(M, I)$, they have however made a marked difference to the ratio represented by C. Digraph C has almost an equivalent cardinality to I. Just ratio $R(C, I)$ alone cannot provide information regarding the match of M to I. It is important to investigate the difference ratios as well. Here again when I and C are compared, that is $R(I \setminus C, I)$, there is little difference between the two digraphs. This indicates that M represented in terms of C matches I well. There however is no improvement in M's relation to I. The ratio of $R(M \setminus C, I)$ indicates that even though there is little change in terms of edges, there is a better match in terms of vertices. As C was empty before the update took place, the result of $C \setminus M$ cannot be compared to the after update result. However, the after update result can be analysed on its own. There are many, 93%, edges that have been inferred in C that are not in M. The difference ratio for vertices is better at 56%.

In conclusion, the match between the model and ideal in terms of the complier was an empty graph. until the introduction of the "entry points". By investigating the exact vertices and edges in the quantities $M \setminus C$ and $C \setminus M$, what is extraneous in M and what has been inferred to be in M and included in C can be determined. This information can be used by a domain expert as input to improve the model.

7.5 Outcome 4 - Exact copy of the ideal

The complier may be an exact copy of the ideal, that is automorphic to the ideal by Definition 2.10, under two very different representations of the model. The first is if the model is automorphic to the ideal. The second is if the model is not automorphic to the ideal, but nevertheless such that the coverage of the ideal by the model is adequate to produce a complier that is automorphic to the ideal. Each of these representations will be discussed in the sections that follow.

7.5.1 Representation 1: M and I are identical

The first of these representations can easily be illustrated by running the algorithm with an ideal and model that are in fact exactly the same and therefore automorphic to each other.

Let the digraph representing the ideal given in Figure 7.5 be used as input parameters for the algorithm for both I and M. After the execution of the algorithm, $\mathcal{T}$, the resultant complier is given by the set of triples:

$$
\begin{aligned}
C_{\mathcal{T}(I,I)} \quad = \quad & \{(a1, b1, \emptyset), (a1, b2, \emptyset), (a1, b3, \emptyset), (a1, b4, \emptyset), (a2, b3, \emptyset), (a2, b5, \emptyset),\\
& (b1, c1, \emptyset), (b1, c2, \emptyset), (b2, c1, \emptyset), (b2, c3, \emptyset), (b2, c5, \emptyset),\\
& (b3, c4, \emptyset), (b3, c6, \emptyset), (b3, c7, \emptyset), (b3, c8, \emptyset), (b4, c4, \emptyset), (b5, c8, \emptyset),\\
& (c1, d1, \emptyset), (c1, d2, \emptyset), (c2, d6, \emptyset), (c3, d2, \emptyset), (c3, d3, \emptyset), (c3, d4, \emptyset),\\
& (c3, d5, \emptyset), (c4, d6, \emptyset), (c5, d5, \emptyset), (c6, d7, \emptyset), (c6, d8, \emptyset), (c6, d9, \emptyset),\\
& (c7, d11, \emptyset), (c7, d7, \emptyset), (c8, d10, \emptyset), (c8, d13, \emptyset),\\
& (d10, e3, \emptyset), (d11, e12, \emptyset), (d13, f3, \emptyset), (d13, f6, \emptyset),\\
& (d2, e1, \emptyset), (d2, e2, \emptyset), (d3, e3, \emptyset), (d3, e4, \emptyset), (d5, e3, \emptyset),\\
& (d6, e3, \emptyset), (d6, e7, \emptyset), (d6, e8, \emptyset), (d7, e5, \emptyset),\\
& (d8, f1, \emptyset), (d8, f2, \emptyset), (d9, e10, \emptyset), (d9, e11, \emptyset), (d9, e9, \emptyset),\\
& (e1, f1, \emptyset), (e12, f3, \emptyset), (e12, f4, \emptyset), (e12, f5, \emptyset), (e8, f2, \emptyset)\}
\end{aligned}
$$

This set of triples representing the complier can be directly matched to the representation of the ideal given in Appendix B.1. The ratios for $R(I \setminus C, I)$, $R(I \setminus M, I)$, $R(M \setminus C, I)$ and $R(C \setminus M, I)$ being 0 is expected and confirms the automorphism.

The order in which the triples in the model are specified also has no effect on the resultant complier. Running the algorithm with $C = \mathcal{T}(I, I)$ presents the same results as running the algorithm with $M_{Z \to A}$, where $M_{Z \to A}$ is I with the triples presented in reverse order, that is $C = \mathcal{T}(I, M_{Z \to A})$. It can therefore be concluded that if I and M are automorphic, then the order in which the triples are specified has no bearing on the complier C.

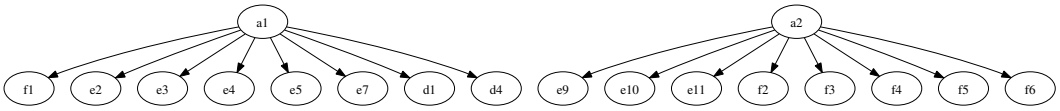


Figure 7.13: Outcome 4 - M

7.5.2 Representation 2 - adequate coverage given by M

The second model representation requires a model that adequately covers the ideal but without being automorphic to the ideal. Consider a model in which only the *source* vertex may be repeated and the *destination* vertices are all unique. The *destination* vertices must also comply with the requirement that they must be the furthest vertex in the path from the specific *source*. Such a model for the ideal given in Figure 7.5 is given by the following set of triples:

$$\begin{aligned} M \quad = \quad & \{(a1, f1, \emptyset), (a1, e2, \emptyset), (a1, e3, \emptyset), (a1, e4, \emptyset), \\ & (a1, e5, \emptyset), (a1, e7, \emptyset), (a1, d1, \emptyset), (a1, d4, \emptyset), \\ & (a2, e9, \emptyset), (a2, e10, \emptyset), (a2, e11, \emptyset), (a2, f2, \emptyset), \\ & (a2, f3, \emptyset), (a2, f4, \emptyset), (a2, f5, \emptyset), (a2, f6, \emptyset)\} \end{aligned}$$

The graphical representation of digraph M, refer to Figure 7.13, reveals that M comprises of two disjoint directed graphs which are actually trees. The first directed tree has a *root* of $a1$ and the other has the *root* $a2$. The output of the algorithm for this run is given in the Appendix in Section B.2. The resultant complier is exactly the same as the one specified by $C_{\mathcal{T}(I,I)}$ and given in Figure 7.5.

Further investigation of the radar chart in Figure 7.14 for this representation of Outcome 4 confirms that M covers I adequately so that C and I are indeed isomorphic. Both the ratios for $R(C, I)$ as well as $R(I \setminus C, I)$ in Figure 7.14 reflect the fact that I and C are exactly the same graph.

The ratio $R(M \setminus C, I)$ which is 0 in terms of vertices confirms that all vertices in M are in C as well. This means that there are no extraneous vertices in M, that is vertices in M which are not in I and therefore cannot be transferred to C. With regards to the edges, these are not exactly the same and differ by 29%. In a real-world application the subject area expert would need to make a decision regarding whether the edge ratio is significant or not. The ratio of $R(C \setminus M, I)$ indicates inference did indeed take place in the construction of the complier for both vertices and edges. For edges, 100% of the edges, that is all the edges have been inferred. When considering the vertices, 59% of the vertices have been inferred.

It should be noted that the value of the ratio $R(I \setminus M, I)$ for the vertices as shown in the radar chart in Figure 7.14 is unclear, and could incorrectly

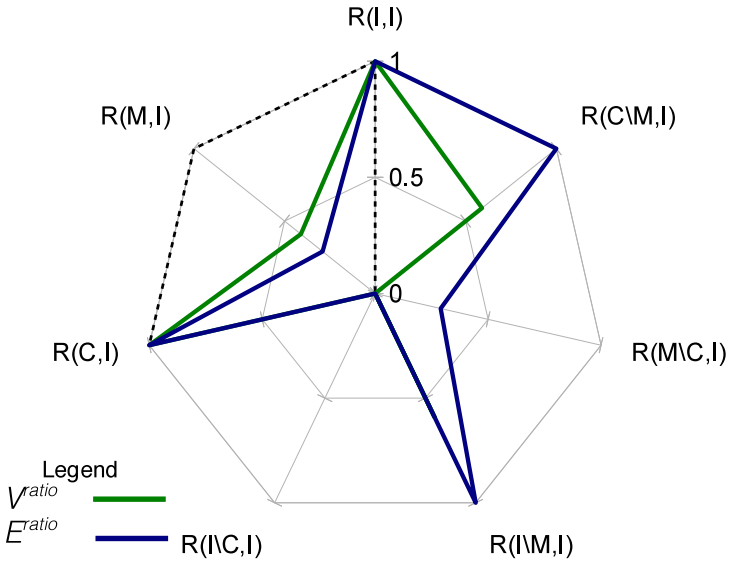


Figure 7.14: Outcome 4 - Radar chart

be construed as 0. This however is not the case. It is actually 0.59. Representing the vertex and edge ratios on individual radar charts removes the obscurity. This can be verified by viewing the radar chart for the vertices only of Outcome 4, as shown in Figure 7.15.

In conclusion, when the model is an exact copy of the ideal the complier will be an exact copy of the ideal. Both the model and the complier are automorphic to the ideal. The complier may also be automorphic to the ideal even if the model is not automorphic to the ideal. In this case the model must adequately cover the information presented in the ideal which results in all vertices and edges not explicitly specified in the model being inferred from the ideal and present in the complier.

7.6 Outcome 5 - Subgraph of the ideal

Applying the algorithm to the ideal and model digraphs as presented by Figures 7.5 and 7.10 in their respective updated forms, that is with the inclusion of the grounding vertex g in each, the resultant complier has already been given in Figure 7.11. The output of the algorithm is presented in Section B.3 of the Appendix.

A visual comparison of the ideal, model and complier is given in Figure 7.16. The blue lines represent the edges between the vertices of the ideal; the red lines represent the edges of the model; and the green lines are the edges of the complier. From the figure it is clear that the ideal and model are different in structure. The complier on the other hand is similar

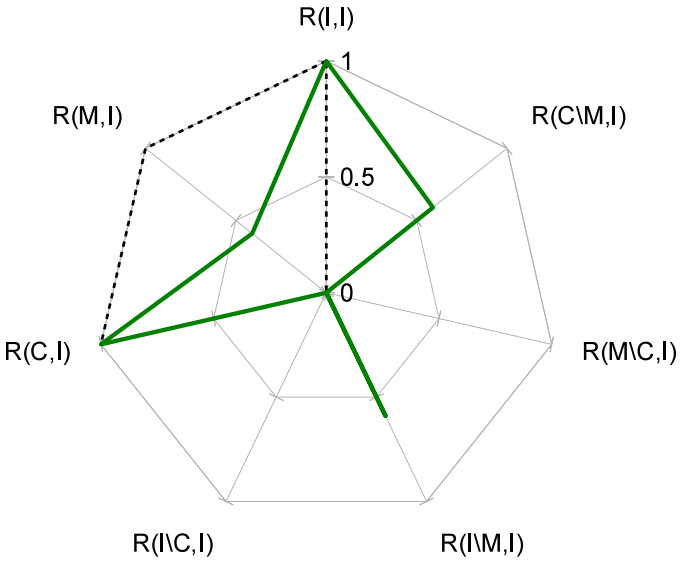


Figure 7.15: Outcome 4 - Radar chart - vertices

to the ideal. It is not an exact match, but comes very close. The majority of the blue edges of the ideal are shadowed by a green complier edge.

From the visual representation, focussing on the ideal and the complier, it can be seen that there is a correspondence between the blue edges of the ideal and the green edges of the complier. The edit distances for these graphs are exactly those presented in Figure 7.3 in the *After update* column. The edit distance for I and C indicates that I and C are very close even though M is vastly different from I and C.

Further investigation into the matching, particularly of the quantity $I \backslash C$ which is used to determine the extent to which the complier is a subgraph isomorphism of the ideal, indicates that the vertices $d1, d4, e4$ and $f5$ from the ideal are not part of the complier and the set of edges

$$\{(c1, d1, \emptyset), (c3, d4, \emptyset), (d3, e4, \emptyset), (e12, f5, \emptyset)\}$$

are not in the complier. The radar chart presented in Figure 7.17 provides an indication of the extent to which the other quantities play a role in the comparison of M to I in terms of C. For ratios $R(I \setminus M, I)$, $R(M \setminus C, I)$ and $R(C \setminus M, I)$, the edge results are by far worse than the vertex results. This indicates that even if the edges of I and M do not match well, the vertices may represent the information adequately for both I and M.

The radar chart for ratios $R(M \setminus C, M)$ and $R(C \setminus M, C)$ presented in Figure 7.18 confirms this. The edges in M that are not in C in relation to the edges in M are high. The same is true for the ratio $R(C \setminus M, C)$ in terms of edges. The respective ratios in terms of vertices presents a similar

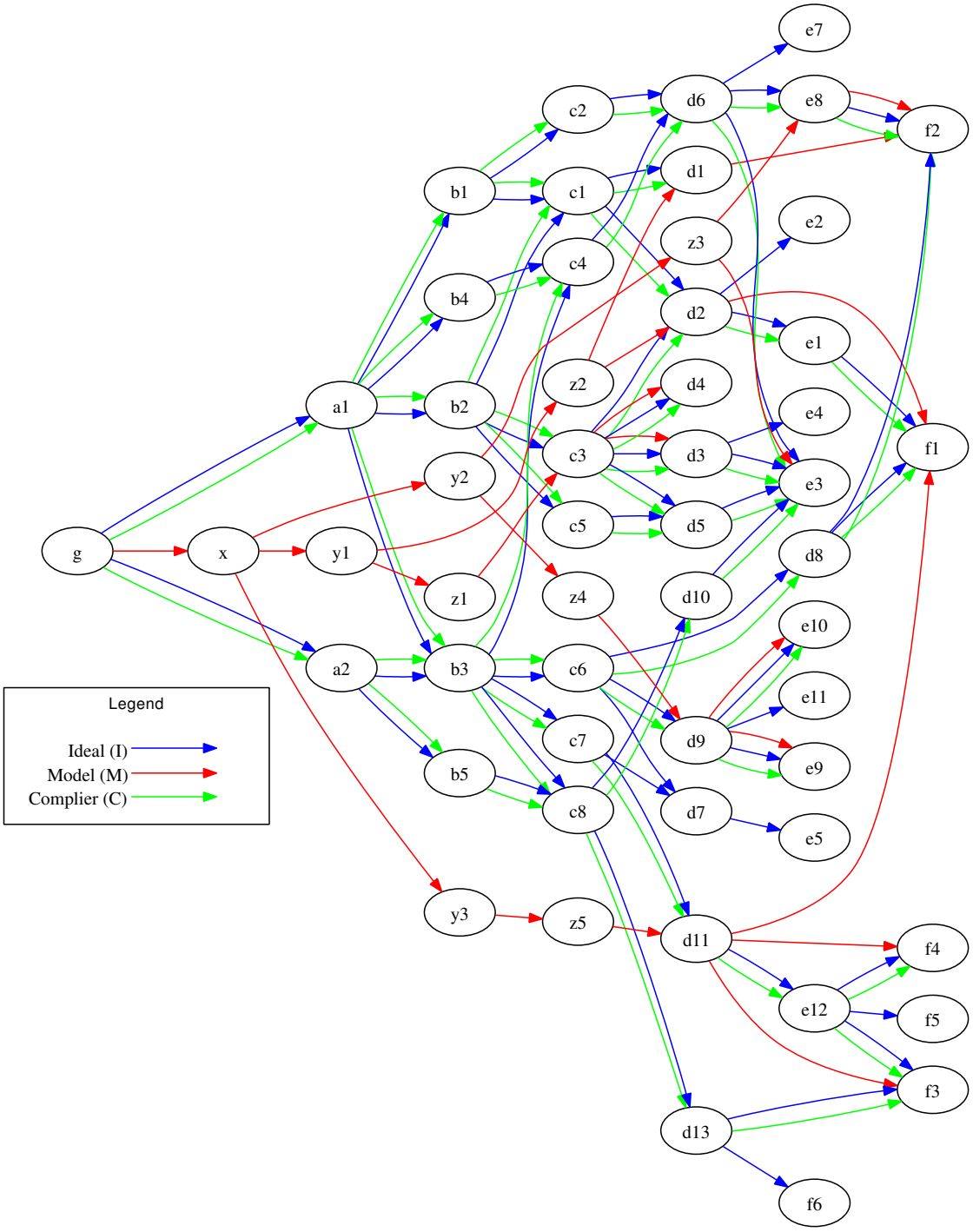


Figure 7.16: Outcome 5 - Comparison of I, M and C

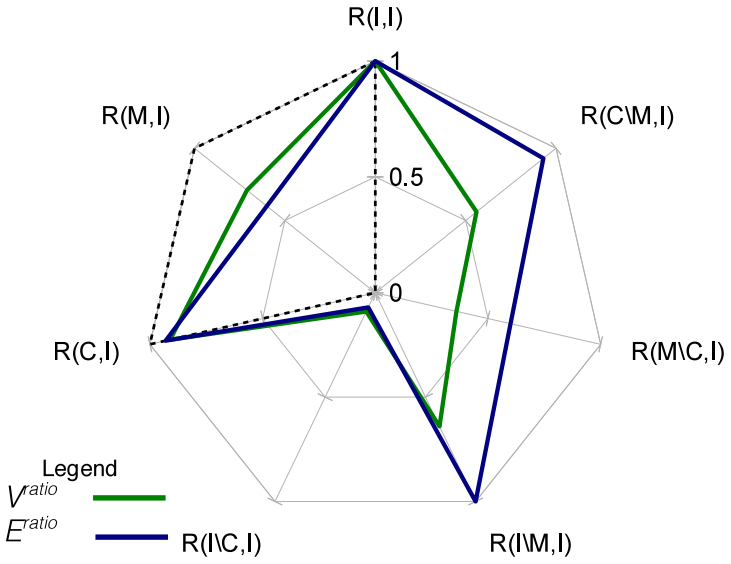


Figure 7.17: Outcome 5 - Radar chart

picture with 50% of the vertices in M not being in C in relation to M and 61% of the vertices in C are not in M in relation to C. This indicates that significant inference has taken place when building C.

In conclusion, as the ratios for vertices and edges for C tend to 1 and the ratio $R(I \setminus C, I)$ tends to 0, C moves closer to being an automorphism of I than a subgraph isomorphism.

7.7 Conclusion

The possible outcomes of algorithm $\mathcal{T}$ presented in Section 5 have been illustrated using toy application digraphs for the ideal and model as input to the algorithm. The complier for each of the outcomes has been analysed using the Graph Comparison Framework presented in Chapter 6.

From the discussions presented, it is clear that a complier that is comparable to the ideal is built using the information of the model and the structure of the ideal. Once the framework has been applied, it may be considered desirable to update the model, and perhaps even the ideal as well, especially when the complier results in outcomes as described by 2 and 3. Updating of the model should be guided by both the outcomes and the resultant vertex and edge sets for quantities $M \setminus C$ and $C \setminus M$ especially. Quantity $M \setminus C$ represents the information in M that is not in C. This information, for the purposes of the thesis has been referred to as extraneous. The quantity $C \setminus M$ presents information in C that is not in M. When building the complier from the information in M this information has been

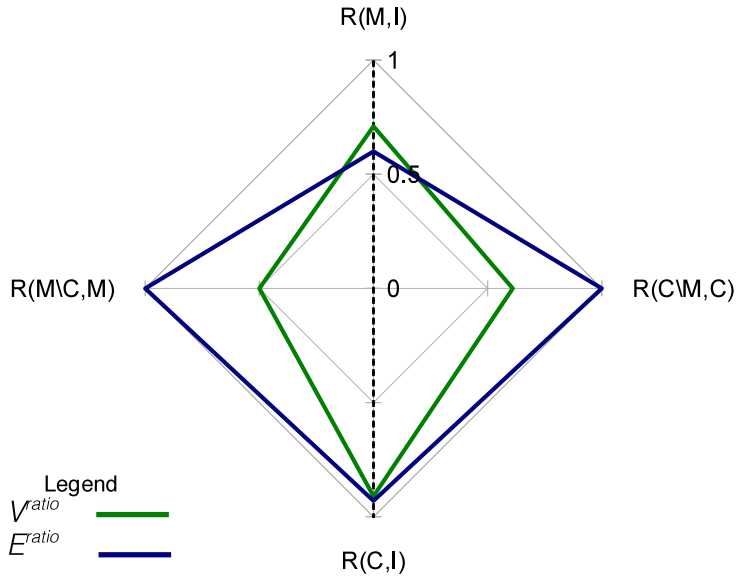


Figure 7.18: Outcome 5 - Radar chart for $R(M \setminus C, M)$ and $R(C \setminus M, C)$

inferred to be in M and may be used in a real-world application to improve M.

These toy applications, presented in this chapter and in Chapter 6, illustrate how the Graph Transformation Algorithm and the Graph Comparison Framework can be applied and how the results can be interpreted in order to determine to what extent the digraphs match. In Part II of this thesis, the framework will be applied to the real world application of curriculum design. In this application of the framework, the digraphs modelling the curricula do not have 10's of vertices and edges, but 100's. It is also not evident from a human perspective, what the similarities and differences are, as it was with the toy application.

Part II

Application

Chapter 8

Computing Curricula Specifications

8.1 Introduction

World over, societies and organisations exist that provide specifications for curricula within the discipline of computing. The most pervasive of these curricula is the ACM/IEEE Computing Curricula [Joint Task Force for Computing Curricula, 2005]. Many other curricula make use of the ACM/IEEE Curriculum volumes as a foundation for their own computing curricula or were involved in the process of specifying the computing curricula as specified by the volumes. Both the British Computer Society (BCS) and the Australian Computer Society (ACS) have been involved in aspects of the ACM/IEEE Curricula series and therefore it can be assumed that the fundamental aspects of their respective curricula are closely associated with those defined by the ACM/IEEE Computing Curricula series.

This chapter introduces the ACM/IEEE Computing Curricula and the volumes, which represent different disciplines. After presenting an overview of the volumes in the series, more details with regards to the Computer Science curriculum volume will be discussed before concluding.

8.2 ACM/IEEE Computing Curricula series

The ACM/IEEE Computing curriculum series [Joint Task Force for Computing Curricula, 2005] currently defines five disciplines within Computing. There is also scope to increase this number. The current disciplines[1] defined in the curriculum series are: Computer Science; Information Systems; Software Engineering; Computer Engineering; and Information Technology. Each discipline will be described in the sections that follow.

[1]English grammar rules require that the names of disciplines, such as computer science, should be written in lower case letters. In this thesis, the rule will not be applied when

8.2.1 Disciplines in the series

The disciplines defined in the ACM/IEEE Computing curriculum series will be briefly described in the sections that follow.

Computer Engineering

Computer Engineering is concerned with the design and construction of computers and computer-based systems. It focusses on the interaction between hardware and software components and the communications between these components.

Computer Engineering has its foundations in traditional electronic engineering and mathematics, but additionally includes the designing of computers and devices. Computer engineers also need to have a solid understanding of software development.

Computer Science

A Computer Science curriculum should cover a wide range of topics from the fundamentals to the more specialised. It also covers both theory and practice, specifically in terms of programming. A Computer Science curriculum should present graduates with a broad, comprehensive foundation in order for the graduate to easily be able to adapt to new technologies and ideas [Joint Task Force for Computing Curricula, 2005]. A Computer Science curriculum must enable the graduate to: design and implement software; identify new ways of using computers; and develop ways of solving computing problems.

Information Systems

The discipline of Information Systems focusses on the integration of information technology solutions and business processes to meet the needs of an organisation. Degree programmes presented in this discipline tend to fall under the business schools in the universities that present them.

The discipline focusses on information and technologies that are necessary to manage the information. It bridges the gap between the management of an organisation and the technical requirements for running the organisation. An Information Systems curriculum must enable the graduate to specify, design and implement the information system requirements of an organisation.

---

referring to the disciplines as defined in the ACM/IEEE Computing curriculum series. The Computing volume disciplines will be presented using a capitalised first letter for each word specifying the discipline, for example, Computer Science.

Information Technology

The Information Technology discipline focusses on all aspects of the technological requirements of an organisation. It includes the selection, creation, application, integration and administration of computing technologies [ACM SIGITE 2008 Task Force on IT Curriculum, 2008]. An Information Technology graduate understands software as well as how to solve technology problems, both related to hardware and software, within an organisation. This is in contrast to the Information Systems discipline which relates to the information of an organisation.

Software Engineering

Software engineers develop and maintain software systems that are reliable and that have been developed according to the requirements of the client. These systems also are maintained by the software engineer. There is an overlap between Computer Science and Software Engineering and many degree programmes present Software Engineering as part of Computer Science. Universities presenting Software Engineering degree programmes require students to take a large portion of the Computer Science curriculum.

From the descriptions above it is clear that the disciplines differ in terms of their focus. Some disciplines look at hardware, others software and some at the organisation itself. It is also clear that degrees conferred within computing resort under Engineering, the Sciences and Commerce. Table 8.1, taken from the Joint Task Force for Computing Curricula [2005, page 12], shows how the disciplines differ in terms of hardware, software and organisational needs. The discipline of Electronic Engineering is not part of the Computing Curricula series, but has been included to distinguish the focus areas of the disciplines that are within the series especially the discipline of Computer Engineering.

8.2.2 Series from 1991 to 2013

The Computer Science discipline was the first discipline to be defined in 1991. The discipline was referred to as the ACM/IEEE Computing Curriculum and abbreviated to CC1991. For a while the "Computing Curriculum" nomenclature was retained. However, with the growth of additional disciplines related to computing, by 2005 it was deemed necessary to rename CC2001 to CS2001 (Computer Science 2001). The term "Computing" was assigned a generic status that embraced these new disciplines. Figure 8.1 introduces each of the disciplines as they are defined in the 2005 ACM/IEEE Computing Curriculum Series document.

Discipline	**Issues**			**Comment and description**
	Hardware	*Software*	*Organisational needs*	
Electronic Engineering	✓			
Computer Engineering	✓	✓		Software development focusses on hardware devices
Computer Science		✓		Used to express ideas and explore problems and applications
Software Engineering		✓		Creating software that satisfies real-world requirements
Information Technology			✓	Makes use of software and hardware to meet the needs of an IT dependent organisation. It focusses on whether the infrastructure of an organisation is appropriate and reliable.
Information Systems			✓	Focusses on the generation and use of information

Table 8.1: Comparison of Computing Curricula [Joint Task Force for Computing Curricula, 2005, page 12]

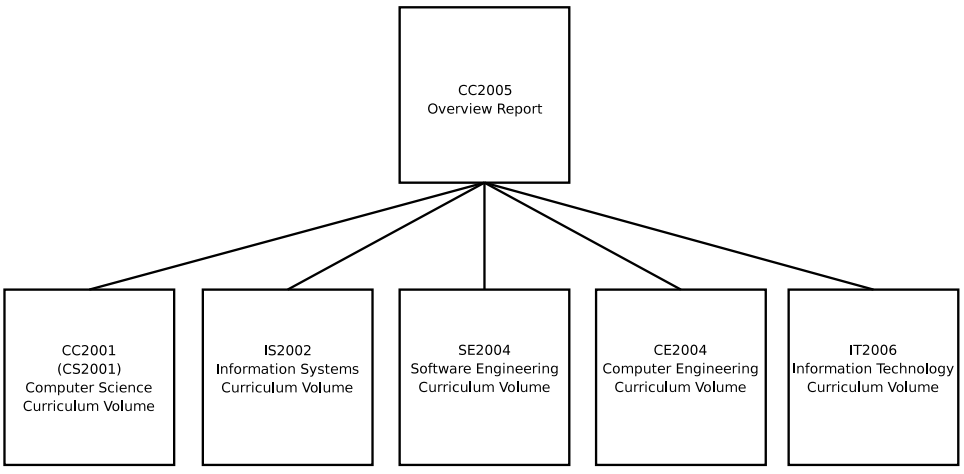


Figure 8.1: Computing Curricula Series [Joint Task Force for Computing Curricula, 2005, page 7]

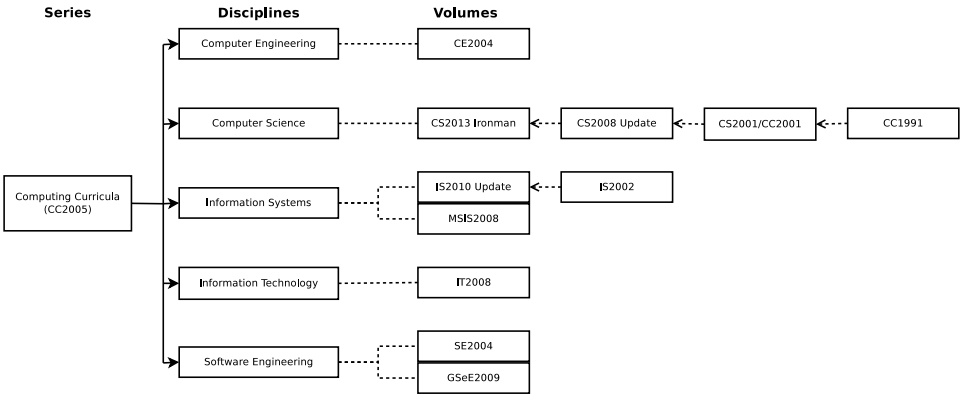


Figure 8.2: Computing Curricula Series 1991 to 2013

The Information Technology volume, referred to in CC2005 as IT2006, was completed and approved in 2008 and has subsequently been referred to as the Information Technology 2008 (IT2008) volume. The SE2004 volume describes the curriculum for undergraduate programmes in Software Engineering and a volume released in 2009 describes a curriculum for graduate programmes in Software Engineering, referred to as GSwE2009. In 2010 the Information Systems volume was updated for undergraduate programmes and a graduate programme curriculum, MSIS, was introduced in 2006. An update of the Computer Science volume is expected to be finalised by the end of 2013. Figure 8.2 provides an overview of the development of the Computing Curricula series from 1991 to 2013. The changes to the Computer Science volume will be further discussed in Section 8.4.

8.3 ACM/IEEE curriculum structure

For each discipline specified by the 2005 ACM/IEEE Computing Curricula Series [Joint Task Force for Computing Curricula, 2005], the contents of the discipline is characterised hierarchically. Various Knowledge Areas (KAs) are defined at the top level of the hierarchy. Each of these, in turn, comprise of Knowledge Units (KUs) and KUs comprise of topics [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2012]. In some cases sub-topics have been identified, but all these can still be seen as more in-depth topic specification. Although an entire KA can be shared between disciplines, it is more likely that a subset of KUs within a KA will be shared. A topic may also resort under more than one KU and therefore under more than one KA.

8.4 ACM/IEEE Computer Science Curriculum

Given the rate at which technology changes, it not surprising that the ACM/ IEEE Computer Science Curriculum volume needs to change too. Approximately every ten years, an updated version of the volume is released. Along with the new curriculum specification, the changes between the new volume and the previous volume are provided as part of the curriculum volume report.

In the sections that follow, a brief history of the ACM/IEEE Computer Science curriculum volume will be given. An overview of the KAs in the curriculum volumes will be presented, illustrating the changes that took place with regards to KAs between the volumes. Finally, the required hours of instruction for the core KAs of the volumes will be presented and compared

8.4.1 A brief history

Prior to the ACM/IEEE Computing Curricula of 1991, the ACM and the IEEE-CS worked independently on computing curricula recommendations. In the spring of 1988, the Joint Curriculum Task Force, comprising of the ACM and the IEEE-CS, was formed to develop curriculum recommendations for computing. The outcome of this partnership was the *Computing Curricula 1991* report [ACM/IEEE-CS Joint Curriculum Task Force, 1991]. Ten years later, the *Computing Curricula 2001* report was released. This was followed by the 2005 overview report which identified different disciplines within computing [Joint Task Force for Computing Curricula, 2005] and the *Computing Curricula 2001* was included in the overview report and renamed *Computer Science 2001* to better reflect the discipline it represented. In 2008, an update to the 2001 report was released and is referred to as the *Computer Science 2008 update* report. At the time of writing, the latest version of the Computer Science curricula volume was under review and was targeted for release towards the end of 2013. Two review volumes were released for public comment. The first was released in the beginning of 2012 and referred to as *Computer Science 2013 Strawman* version. Comment was open until June 2012. These comments were incorporated into the *Computer Science 2013 Ironman* version that was released in February of 2013.

The last complete Computer Science Curricula volume was therefore released in 2001. The 2008 update of the 2001 curriculum volume has a strong focus on Computer Security [ACM/IEEE-Curriculum CS2008 Joint Task Force, 2008]. In both the 2001 and 2008 curricula volumes, KUs were identified as being either *core* or *elective* [ACM/IEEE-Curriculum 2001 Task Force, 2001; ACM/IEEE-Curriculum CS2008 Joint Task Force, 2008]. Core KUs and their respective topics are regarded as base line requirements in any curriculum that is developed for the discipline. The minimum number

SE/Software Processes

[1 Core-Tier1 hours; 2 Core-Tier2 hours]

Topics:

[Core-Tier1]

- Systems level considerations, i.e., the interaction of software with its intended environment
- Phases of software life-cycles
- Programming in the large vs. individual programming

[Core-Tier2]

- Software process models (e.g., waterfall, incremental, agile)

[Elective]

- Software quality concepts
- Process improvement
- Software process capability maturity models
- Software process measurements

Figure 8.3: CS2013 Strawman Software Engineering KA, Software Processes KU [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2012, Page147]

of hours that should be spent on each KU is also specified. Elective KUs on the other hand are optional to the curriculum being developed.

The 2013 Strawman volume (hereafter CS2013S) [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2012] and updated Ironman volume (hereafter CS2013I) move the designation of *core* or *elective* from the KU level to the topic level and distinguish between two types of core topics. The first is referred to as *core tier 1* (CT1) and the second as *core tier 2* (CT2). Topics are now labeled as being core (either CT1 or CT2) or elective. A KU may therefore contain a mixture of core and elective topics. Refer to Figure 8.3 for an example containing CT1, CT2 and elective topics. Core topics, both CT1 and CT2, have a minimum number of hours allocated to them. In the figure, a minimum of one hour has been allocated to CT1 topics and two hours to the CT2 topics.

A further requirement in CS2013S, which has been incorporated into CS2013I as well, is that all topics in CT1 must be included in a curriculum, while at least 90% (with an 80% bare minimum) of CT2 is considered essential to be included in a Computer Science curriculum.

8.4.2 Changing Knowledge Areas (KAs)

CC1991 specified 11 subject areas (SAs). These were named with tags in brackets as [ACM/IEEE-CS Joint Curriculum Task Force, 1991]:

Algorithms and Data Structures (AL);
Architecture (AR);
Artificial Intelligence and Robotics (AI);
Database and Information Retrieval (DB);
Human-Computer Communication (HU);
Numerical and Symbolic Computation (NU);
Operating Systems (OS);
Programming Languages (PL);
Introduction to a Programming Language - optional (PR);
Software Methodology and Engineering (SE); and
Social, Ethical and Professional Issues (SP).

When CC2001 was released, the SAs were referred to as knowledge areas (KAs) and there was a total of 14 KAs identified. The number of KAs remained the same in CS2008, but increased to 18 in CS2013S [Sahami et al., 2012].

Table 8.2 provides a summary of the changes that took place from 1991 to the 2013 Strawman and Ironman versions of the ACM/IEEE Computer Curriculum volume. The six (6) KAs marked with an asterisk (*) map directly to SAs in Computing Curricula 1991. The other 5 SAs defined in CC1991 have been included in KAs marked with two asterisks (**).

The most significant change between CC1991 and CC2001 is the addition of KAs that focus on Discrete Structures, Computer Graphics and Net-centric Computing. Between CC2001 and CS2008, no marked changed took place with regards to the KAs.

From the table it is clear that there were no changes in KAs between CC2001 and CS2008. The changes came about in CS2013S. The KAs AL, AR, DS, HC, IM, IS, OS, PL and SE in CS2013S remain the same as they are in CC2001. KAs CN, GV, NC and SP changed focus, but not enough to drop them entirely. The PF knowledge area was dropped in CS2013S and IAS, PBD, PD, SDF and SF were introduced.

As with the KAs, the number of core KUs and topics also increased from 1991 to 2013. Figure 8.4 shows the number of core KAs in which core topics reside. In CC1991, all KUs related to the 10 non-optional KAs are core. The core topic count for CC1991 has not been shown. The related KAs and topics for the core KUs specified in the CC2001 and CS2008 volumes are also shown. Finally, the information shown for CS2013S reflects KUs that contain one or more CT1 or CT2 topics, as well as the KAs associated with these KUs. From the figure it is evident that there has been a marginal increase in topics between CC2001 and CS2008. However, between CS2008 and CS2013S, there has been a marked increase in all aspects.

KA	CC2001	CS2008	CS2013S
AL*	Algorithms and Complexity		
AR*	Architecture and Organisation		
CN**	Computational Science and numerical methods		Computational Science
DS	Discrete Structures		
GV	Graphics and Visual Computing		Graphics and Visualisation
HC**	Human-Computer Interaction		
IAS			Information Assurance and Security
IM**	Information Management		
IS**	Intelligent Systems		
NC	Net-centric Computing		Networking and Communication
OS*	Operating Systems		
PBD			Platform-Based Development
PD			Parallel and Distributed Computing
PF**	Programming Fundamentals		
PL*	Programming Languages		
SDF			Software Development Fundamentals
SE*	Software Engineering		
SF			Systems Fundamentals
SP*	Social and Professional Issues		Social and Professional Practice

* SA defined in CC1991
** CC1991 SA included in the KA

Table 8.2: Knowledge Areas

8.4.3 Core hour requirements

All curricula specifications, from 1991 to CS2013I, specify the core hours in terms of lecture hours. Lecture hours are the number of hours it takes to present the material in a lecture-based format. Lecture hours exclude activities such as preparation, practical sessions, assessment etc. related to the presentation of the material in the lecture.

Figure 8.5 gives the core hour requirements for each of the curriculum volumes. In the 1991, 2001 and 2008 volumes, KUs were specified as core

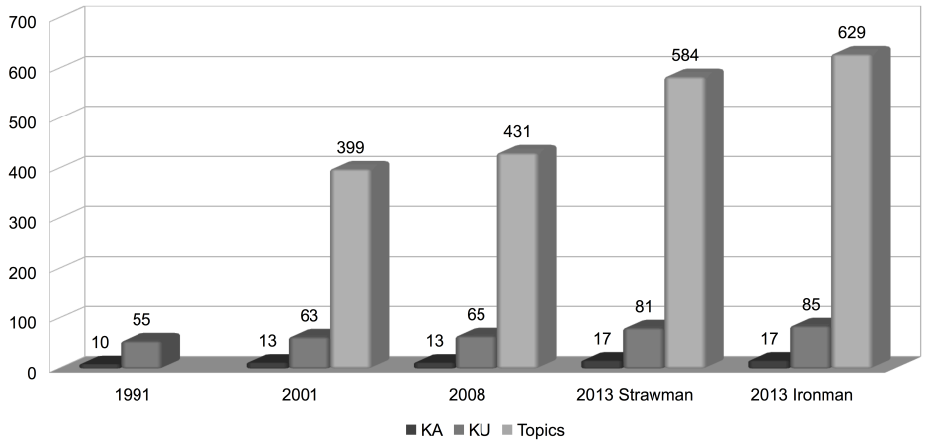


Figure 8.4: Core content of curricula

or elective. As previously noted, in the 2013 Strawman and Ironman volumes, the specification of core and elective was moved to the topic level, and a distinction was made between two tiers of core topics. CS2013I specifies 165 hours in total of CT1 topics and 142 CT2 hours [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2013]. Calculating the average core hours across the curriculum volumes suggests that a Computer Science degree programme requires approximately 284 $((271+280+280+(165+142))\div 4)$ lecture hours on core material. A general rule is that the out of lecture time a student should spend on the material presented in a lecture should be about 3 times the lecture hours. This gives a total of approximately $284 \times 4 = 1136$ hours required for Computer Science core material. In CC2001 it is suggested that a student should devote at least 160 hours per module [ACM/IEEE-Curriculum 2001 Task Force, 2001] in the Computer Science curriculum. This means that in order to cover the core Computer Science content, at least $1136 \div 160 = 7.1$ core modules are required in the curriculum of a Computer Science degree programme.

8.4.4 Excerpts from the curricula

Excerpts of the Computer Science curricula volumes are given in Appendix D. Both CC2001 and CS2008 present an overview of the curriculum volume KAs and KUs in tabular form. Refer to Figures D.1 and D.3 respectively. In CC2001 the core hours per KA is given when the name of the KA is specified. A breakdown of the core hours per KU is given in parenthesis are for the specific KU. The core KUs are also underlined. CS2008 follows a similar structure as CC2001, except the core KUs are not underlined. The details of the the KUs follow the overview tables for CC2001 and CS2008 in the

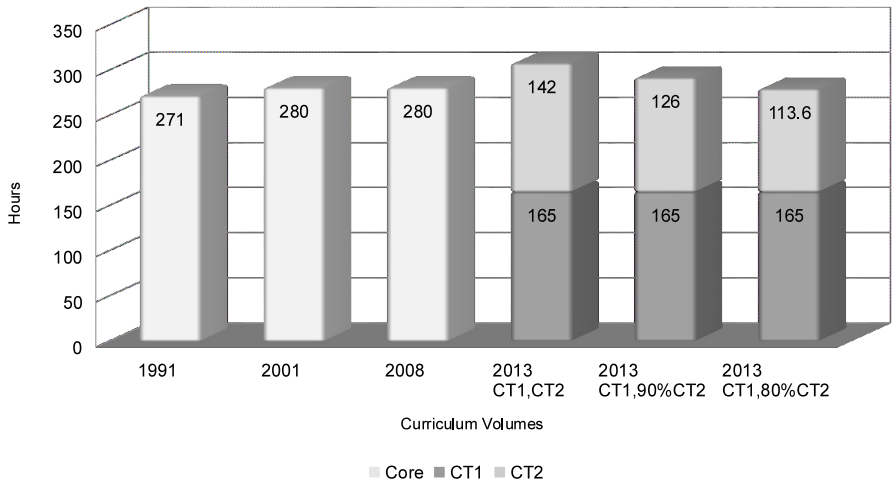


Figure 8.5: Core hour requirements

order in which they appear in the tables. In both the Strawman and Ironman volumes, each KA of the Computer Science Body of Knowledge (CS-BoK) is specified individually before the KUs are discussed in more detail. An example of the specification of the Software Development Fundamentals KA is given in Figures D.5 and D.7 for CS2013S and CS2013I respectively. Already here there is a difference between core hours for SDF/Development Method. The CT1 hours are given as 9 in CS2013S and 10 in CS2013I.

Examples of KU details for CC2001 and CS2008 are given in Figures D.2 and D.4 respectively. The KU presented is the Data Structures KU in the Programming Fundamentals KA. The Data Structures KU in CS2013S and CS2013I falls into the renamed Software Development KA. Figures D.6 and D.8 present the KU details for Strawman and Ironman respectively.

In the CS2013 Strawman and Ironman volumes, a distinction is made with regards to the learning outcomes as to where in the curriculum these topics need to be taught. Learning outcomes are categorised as Knowledge, Application or Evaluation. These relate to the level of mastery required for the particular topic with Knowledge representing a low level of mastery [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2012].

8.5 Conclusion

This chapter has introduced the ACM/IEEE Computing Curricula series. It gave a brief history of the origins and contituents of the series. The Computer Science discipline as identified within the series was presented in

more detail, showing the components that make up the volumes of 2001, 2008 and the proposed 2013 volume in the form of the Strawman and Ironman drafts. A comparison between the volumes was presented on the KA-level prior to looking how the different volumes specify core and elective aspects of the curriculum. This comparison, along with the presentation of the excerpts of the volumes, is presented on a high-level of specification. More detail regarding the similarities and difference between the modules will be presented in Chapter 11 when the Graph Comparison Framework that was discussed in Chapter 6 is applied to the curriculum volumes.

Chapter 9

University Degree Programme Requirements

9.1 Introduction

This chapter will discuss some of the hurdles that universities in a country like South Africa may encounter when developing a curriculum for Computer Science. In order to achieve this the qualifications structure, referred to as the National Qualifications Framework in South Africa, will be introduced and contrasted with a selection of countries against which universities in South Africa benchmark their Computer Science curricula. Accreditation will briefly be discussed as this is a mechanism by which degree programmes are compared on an international basis in order to determine equivalence, often referred to as *substantial equivalence*.

9.2 Qualification structures

In order to be able to compare a South African qualification with other degrees in the world, an overview of the qualification structures in North America, Australia, Europe and the United Kingdom is provided. A summary, showing the comparison between the qualification structures will conclude the section.

Countries, other than South Africa, that fall into the BRICS - an association of emerging national economies: Brazil, Russia, India, China and South Africa - grouping have very similar degree structures as Europe and the United States. The trend is to have a four years Bachelor's followed by a two years Master's and about three years of Doctorate study. There are some interesting additional requirements that students wanting to pursue postgraduate qualifications need to comply with. In India for example, in order to enter a Doctorate programme a student must have completed a Master's degree. They are then allowed to continue with an MPhil which

may take them into the Doctorate programme [International Education Exchange Center, 2009]. China requires prospective Doctorate students to be endorsed by two academics on the Associate Professor or Professor level before being allowed to enter the programme [Chinese Government's Official Web Portal, 2005]. Professional and accreditation bodies from these countries are also emerging as signatories of accreditation agreements in order to register degree programmes that are internationally recognised. As will be seen in Section 9.3.

9.2.1 United States of America, Canada and parts of Australia

Countries in Northern America and parts of Australia, refer to qualifications in terms of years after kindergarten. A high school diploma is on level K-12. A Bachelor's degree on K-16, 4 years after K-12. The first two years after K-12 are college years followed by two university years. The college years are referred to as Freshman and Sophmore years, while the two university years are called the Junior and Senior years [ABET Review, 2010]. A Bachelor's degree is conferred when a student completes approximately 120 semester hours over the 4 years of study in accordance with the regulations of the institution. A semester hour is used as a universal quantification of contact time. Semester hours are calculated by multiplying the number of contact, also referred to as lecture, minutes in a week with the number of weeks in a semester and dividing by 60. A practical session of less than 4 hours per week will add 1 semester hour to the total. A typical semester is from 14 to 17 weeks. The time allocated to a lecture is institution dependent. An average module will equate to approximately 3 semester hours.

Master's and Doctoral qualifications follow on from the Bachelor's and typically take at least 2 years each.

9.2.2 Europe

The European Higher Education Area (EHEA) was established as a direct result of the Bologna process [Fuller et al., 2006]. The Bologna process culminated in an agreement between different educational ministries in Europe regarding the transfer of credits between countries and the equivalences of qualifications in higher education. The Bologna process was proposed at the University of Bologna and the declaration was signed in 1999. The Bologna declaration came about due to the differences in education systems in many countries in Europe and the incompatibility between them [Career Space Consortium, 2001]. The aim of the EHEA is to facilitate the development of higher education in Europe. There were two dominant education systems in Europe, the continental system, mainly based on the German system, and the Anglo-American system.

As a result of the Bologna process, a framework comprising 3 cycles has been developed [EHEA Framework, 2005]. An undergraduate degree (represented by cycle 1) of at least three years is called a Bachelor's degree (or licence) and requires between 180 and 240 ECTS (European Credit Transfer and Accumulation System) credits. The Bachelor's degree may include a Short Cycle of 120 ECTS credits. There are 60 ECTS credits in a full time year of study, representing about 1500-1800 hours of study. Cycle two represents a one or two year diploma called a Masters of 90 to 120 ECTS credits with at least 60 credits on cycle 2 level. Cycle 3 represents a Doctorate degree and is meant to be obtained in at least 3 years of study. Cycle 3 does not have a credit requirement.

9.2.3 United Kingdom

The United Kingdom has two defined frameworks: one for England, Wales and Northern Ireland; and a separate one for Scotland. Each of these frameworks are briefly explained in the paragraphs that follow.

The Qualifications and Credit Framework (QCF) of 2010 applies to England, Wales and Northern Ireland. The framework comprises of 9 levels, beginning at the Entry level and then from 1 to 8 [Office of Qualifications and Examinations Regulations, 2012]. Levels 6, 7 and 8 map onto the degree designations for Bachelor's with Honours, Masters and Doctorate as specified in the Framework for Higher Education Qualifications (FHEQ) which is regulated by the Quality Assurance Agency (QAA) [Qualifications Assurance Authority, 2008]. The FHEQ was developed so that credit transfer can take place between the United Kingdom and Europe with levels 6, 7 and 8 mapping directly onto the European cycles. Levels 6 and 7 in the framework have a credit count. Level 6 is 360, level 7 180. Level 8 typically has no credit value [Diagram of higher education qualification levels in England, Wales and Northern Ireland]. One credit equates to 10 hours of learning.

The Scottish Credit and Qualifications Framework (SCQF) works on a 12 point level scale with Bachelor degree programmes beginning at level 9. Levels 10, 11 and 12 represent Honours, Master's and Doctorate respectively. An Honours refers to a degree that is designed to be completed in 4 years or more [Qualifications Assurance Authority, 2001].

9.2.4 South Africa

All qualifications in South Africa must comply the specifications of the National Qualifications Framework (NQF) as set out by the national Department of Education. This is similar to the structures already presented in the previous sections.

The NQF defines three categories of certificates in education and training, beginning in primary school and ending with Doctoral studies. These

categories are referred to as: General Education and Training Certificate (GETC), Further Education and Training Certificate (FETC) and Higher Education and Training Certificate (HETC) [NQF Bands, 2010]. Table 9.1 shows the categories with their corresponding NQF Exit levels. In order to enter a Bachelor's degree programme as defined in the HETC category, a relevant National Senior Certificate (NSC) is required with an exit level of 4.

Category	NQF exit level	Description
GETC	1	First 9 years of schooling
FETC	2	Grade 10
	3	Grade 11
	4	Grade 12
HETC	5 to 10	Specified by the HEQF

Table 9.1: NQF categories

The HETC category is detailed in the Higher Education Qualifications Framework (HEQF). The HEQF identifies two qualification types, undergraduate and postgraduate. NQF exit levels 5, 6 and 7 are considered undergraduate study. A Bachelor's degree taking 3 years of study is considered to have an NQF exit level of 7. Postgraduate study occupies levels 8, 9 and 10. Table 9.2 summerises the NQF exit levels of the HEQF. For each degree designation the total minimum credits for the degree is given in the second last column of the table. The credits give a direct indication of the hours required to complete the degree. One credit equates to 10 hours of study, referred to as notional hours. A Bachelor's with an NQF exit level of 8 takes 4 years of study and is referred to as a professional degree. It leads directly to a Master's degree programme [Pandor, 2007].

NQF exit level	Degree designation	Min total credits	Year(s)
7	Bachelor's	360	3
8	Bachelor's	480	4
8	Honours	120	1
9	Master's	180	1 to 2
10	Doctorate	360	2

Table 9.2: HEQF exit levels

9.2.5 Summary

Table 9.3 provides a summary of the qualifications presented in South Africa, the United Kingdom, Europe and the United States of America. United

Kingdom(1) in the table refers to the framework for England, Wales and Northern Ireland, and United Kingdom(2), the framework for Scotland. As can be seen, the naming conventions of the degree programmes are standard with regards to Bachelor's, Master's and Doctorate. The concept of an Honours degree that follows a 3 year Bachelor's is only defined in the qualifications frameworks of South Africa and Scotland.

The minimum total hours of study for each of the qualification frameworks is presented in brackets in Table 9.3. These hours include both class, also referred to as contact, hours and preparation hours. For the calculation of the hours for the United States it is assumed that a semester is 14 weeks and that for every contact hour there are three hours required for preparation. Hours for postgraduate studies are dependent on the institution but tend to be around 5400 hours for Master's and Doctorate combined.

9.3 Accreditation structures

Accreditation of programmes takes place at different levels and for a variety of reasons. Accreditation of degree programmes for credit transfer or furthering qualifications may take place at either the institutional level or at a governmental level. The Bologna process helps with these types of transfers in Europe. In South Africa, transfer between South African universities is done on the institutional level, while transfer from outside South Africa to a South African institution is on the governmental level. Accreditation of specific degree programmes to ensure that the discipline content is complied with is done by accreditation bodies that focus on the particular discipline.

9.3.1 Accreditation for transfer reasons in South Africa

In South Africa, the South African Qualifications Authority (SAQA) determines the equivalences of degree programmes and certifies degree programmes within South Africa. International students who wish to study in South Africa are required to present their existing qualifications to SAQA for approval. SAQA does not stipulate curricula, nor does it provide an accreditation service.

The South African Department of Higher Education classifies educational institutions of higher education and training as either public or private. Public South African Universities are categorised into three categories: Universities, Comprehensive Universities and Universities of Technology [International Education Association of South Africa, 2009; Jooste, 2009]. Universities provide theoretically-oriented degree programmes while Universities of Technology focus on vocational programmes. Comprehensive Universities deliver programmes that may be either theoretical or vocational or both theoretical and vocational. Private institutions have programmes that are approved by SAQA for presentation in South Africa.

Degree	**South Africa** NQF levels	**United Kingdom (1)** QCF levels	**United Kingdom(2)** SCQF Levels	**Europe** Cycles	**United States**
Bachelor's 3-year	7 (3600)		9		
Bachelor's 4-year	8 (4800)	6 (3600)		1 (4500)	Bachelor's (6720)
Honours	8 (1200)		10		
Master's	9 (1800)	7 (1800)	11	2 (2250)	Master's
Doctorate	10 (3600)	8 (no credit value)	12	3 (no credit requirement)	Doctorate

Table 9.3: Higher education qualification summary

9.3.2 Accreditation of disciplines

Accreditation is a mechanism used to assure quality of institutions and their degree programs [Ford, 1991]. It is especially necessary for disciplines in which graduates are to enter into professional practice. Accreditation can be acquired on either an institutional level, school or department level, or on a programme level. Many accreditation bodies follow the programme level model [Impagliazzo et al., 1997].

The world over, societies and organisations exist that provide specifications for curricula and the accreditation thereof. In many cases there is synergy between these groupings, specifically accreditation agreements in which there is mutual understanding of accreditation status. These agreements are in the form of being signatories, or members, of an accord.

The accords

There are two predominant accreditation agreements, the Washington Accord and the Sydney Accord. The Washington Accord focuses on engineering programs, while the Sydney Accord is focused on technology programs.

Washington Accord: The Washington Accord was established in 1989 to recognise substantial equivalence of engineering programs accredited by the organisations holding the member signatory status and their affiliates within the particular countries. The accord enables programmes that have been accredited to be recognised by other signatories. Graduates of accredited programmes have therefore met the requirements to practice engineering in any of the jurisdictions of the signatory [Accord, Washington].

The organisations that originally signed the Washington Accord in 1989, along with their respective countries in brackets, are: Engineers Australia - IEAust (Australia), Canadian Engineering Accreditation Board of Engineers Canada - CEAB-EC (Canada), Engineers Ireland - EI (Ireland), Institution of Professional Engineers New Zealand - IPENZ (New Zealand), Engineering Council UK - ECUK (United Kingdom) and the Accreditation Board of Engineering and Technology - ABET (United States of America). In 1999 the Engineering Council of South Africa - ECSA joined as a signatory of the the accord. Figure 9.1 presents the signatories of the Washington Accord according to when they signed and their status.

Sydney Accord: The Sydney Accord came into being in 2001 as a result of the efforts of the Ottawa Intent (1999) working group. It provides for recognition of technology programs, particularly in engineering. The accord was signed by IEAust (Australia), CEAB-EC (Canada), HKIE (Hong Kong China) , EI (Ireland), IPENZ (New Zealand), ECSA (South Africa) and ECUK (United Kingdom). The arrows that emanate from box on the right

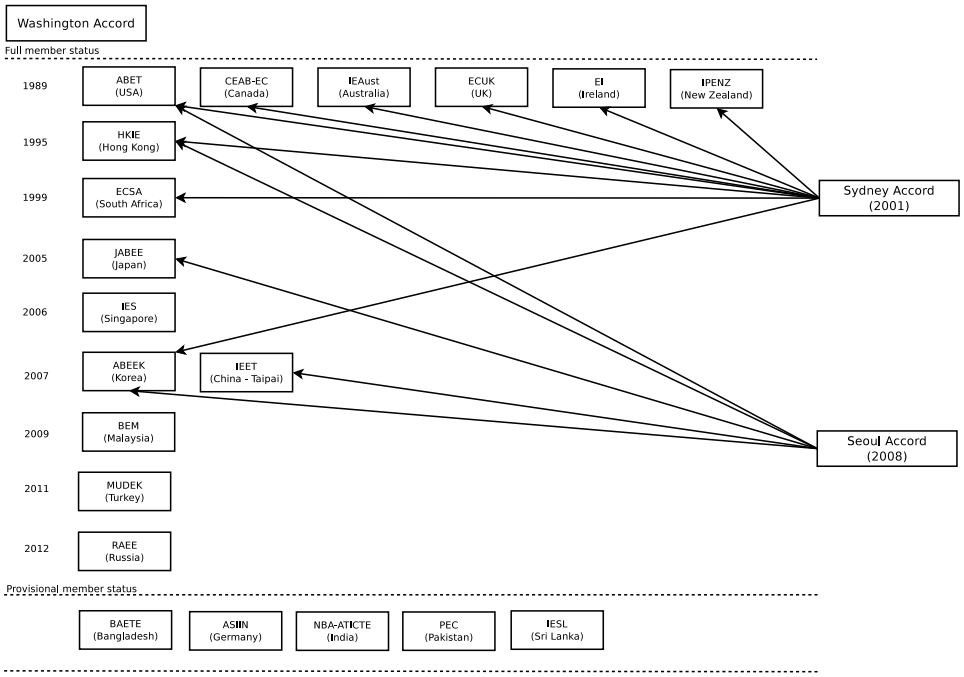


Figure 9.1: Accreditation Accords [Signatories in September 2013]

labelled Sydney Accord of Figure 9.1 show the signatories of the Sydney Accord [Accord, Sydney]. Figure 9.2 presents the signatories of the Sydney Accord according to the years in which they signed the accord.

Accrediting computing

The Joint Task Force for Computing Curricula [2005, pages 45 to 48] briefly discusses accreditation of computing programs in the USA by ABET and the UK by the British Computer Society - BCS. Accreditation of computer science programs began in the 1980's [Ford, 1991]. As with engineering, certain parties perceived a need to bring into being an accord specifically for Computer Science accreditation which is referred to as the Seoul Accord. After a brief discussion of the Seoul Accord, a brief discussion of the ABET accreditation criteria will be presented.

Note, in passing, that in South Africa no official national organisation to represent the interests of the educational aspects of the Computer Science community. South African universities therefore tend to be guided by international trends for both the curriculum specification as well as accreditation criteria.

Seoul Accord: In 2008 the Seoul Accord, specifically for Computer Science accreditation between different accreditation bodies in the world, was

Figure 9.2: Sydney Accord [Signatories in September 2013]

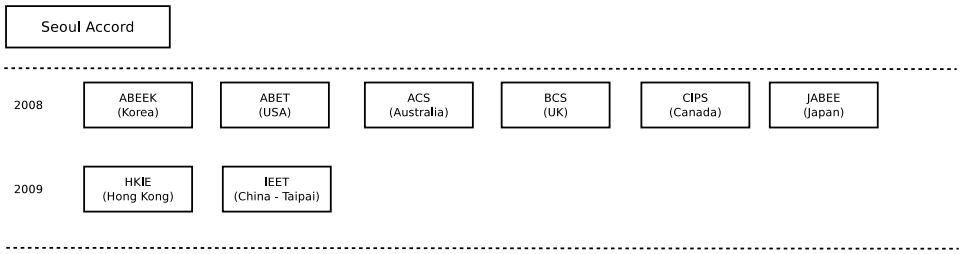


Figure 9.3: Seoul Accord [Signatories in August 2013]

signed. It aimed at establishing an agreement of mutual recognition between the signatories of the accord to recognise graduates from accredited programmes between the accreditation organisations in the discipline of Computer Science. The founding 6 signatories of the accord were the Accreditation Board for Engineering Education of Korea - ABEEK (Republic of Korea), ABET (USA), Australian Computer Society - ACS (Australia), BCS (United Kingdom), Canadian Information Processing Society - CIPS (Canada) and JABEE (Japan) [Calitz, 2010]. In June 2009 two more signatories joined the Seoul Accord, the Hong Kong Institution of Engineers - HKIE (Hong Kong China) and Institution of Engineering Education Taiwan - IEET (Chinese Taipei) [Accord, Seoul]. Figure 9.3 presents the signatories of the Seoul Accord. The signatories of the Seoul Accord that are also signatories of the Washington Accord have also been included in Figure 9.1.

ABET: In order for a Computer Science degree to be accredited by ABET, a Computer Science degree programme must comply with specified minimum requirements. A summary of these requirements is given in Table 9.4 [ABET Criteria, 2010; ABET Review, 2010]. From the table it can be seen that the requirements cover 4 categories, each having a minimum number of semester hours linked to them to which the degree programme

must comply. A typical Computer Science Bachelor's degree programme requires at least 120 semester hours.

Category A relates to the Computer Science component of the degree programme and is divided into two sub-categories, fundamental and advanced. Fundamental Computer Science focusses on the areas of algorithms, data structures, software design, programming languages concepts, computer organisation and architecture. The advanced category builds on the fundamentals and provides depth with regards to Computer Science knowledge.

The Mathematics and Science category (B) is also divided into two subcategories each requiring a specific number of semester hours to be complied with. The mathematics subcategory (B1) must include discrete structures as a field of study and may be augmented with modules in calculus, linear algebra, numerical methods, probability, statistics, number theory, geometry, or symbolic logic. The science subcategory (B2) provides exposure to lab work and subsequently scientific reasoning.

Category C provides breadth to the degree programme by requiring modules to be taken from other disciplines such as Humanities, Arts and Economic Sciences. Category D on the other hand requires the degree programme to include modules in order for students to be able to engage in the profession of Computer Science. These modules include oral and written skills as well as ethical aspects of the discipline.

Necessary skills that are not linked to semester hours, but still form an integral part of the accreditation requirements, are skills such as problem solving and having a good knowledge of at least one programming language. However, it is also required that the degree program should expose the student to more than one language.

9.4 Institutional requirements

In the previous sections, governmental and accreditation body requirements for degree programmes have been discussed. These are not the only requirements that degree programmes at a university need to adhere to. The university itself may also place institutional requirements onto degree programmes it offers. Institutional requirements tend to include required modules for degree programmes. They may also dictate the structure of degree programmes. Institutional requirements are also in many cases more specific and on a lower level of granularity that governmental or accreditation body requirements.

Category	Semester Hours	Sub-category	Semester Hours
Computer Science (A)	$\geq$ 40	Fundamental (A1)	$\geq$16
		Advanced (A2)	$\geq$ 16
Mathematics and Science (B)	$\geq$ 30	Mathematics (B1)	$\geq$ 15
		Science (B2)	$\geq$ 12
Education General (C)	$\geq$ 30		
Other (D)	difference		
TOTAL	$\geq$ 120		

Table 9.4: Minimum ABET requirements

9.5 Challenges

From a South African perspective, the two main challenges faced in preparing a curriculum for ABET accreditation based on the ACM/IEEE Computer Science curriculum are:

1. the compatibility between credits and semester hours; and
2. that Bachelor's degree programmes in South Africa are typically 3 years instead of 4.

Minor challenges include complying with government and institutional requirements as set out for degree programmes and competing internationally with institutions in more prosperous countries, some of which might have attained institutional ranking on the world ranking lists. These challenges will briefly be discussed in the sections that follow.

9.5.1 Economic

The United Nations classifies Africa as a developing continent, but treats the countries belonging to the Southern African Customs Union (SACU) as falling into a developed region [United Nations, 2010]. Botswana, Lesotho, Namibia, South Africa and Swaziland are members of SACU [Southern African Customs Union, 2011]. The World Bank classifies South Africa as an "upper middle income" economy with a so-called IBRD lending category [World Bank, 2010]. IBRD (International Bank for Reconstruction and Development) is a division of The World Bank and provides loans to governments with an extended repayment option. The United States, Europe and Australia are classified as "High income: OECD". This means that they are members of the Organisation for Economic Co-operation and Development (OECD) which is an organisation for the stimulation of economies and world trade. South Africa is therefore not in the same league economically as the United States, Europe or Australia against whom we compete.

9.5.2 Ranking

With regards to the academic ranking, none of the universities in South Africa which present Computer Science degree programmes rank on the 2013 list of the Academic Ranking of World Universities for Computer Science [ARWU CS 2013, 2013]. The top rankings are all North American. These rankings are determined by using the criteria published by Academic Ranking of World Universities (ARWU) at `http://www.shanghairanking.com/ARWU-SUBJECT-Methodology-2013.html`.

9.5.3 Semester hours versus notional hours

Credits as defined by the South African NQF and semester hours as defined by the ACM/IEEE and in the ABET specification are not directly comparable units of measure. It is therefore necessary to determine the equivalent semester hours per module presented in South Africa in order to calculate compliance to the international requirements.

Semester hours represent contact hours [ABET Criteria, 2010; ABET Review, 2010], while the credits per module in the South African system reflect so-called notional hours [Pandor, 2007]. Notional hours represent the total time expected to complete the module, which includes contact hours, preparation time and non-class related educational activities such as completing of assignments. An indication of the total time required for a degree programme has already been presented in Table 9.3.

In order to be able to compare the contact time per module it is necessary to calculate the contact time, in actual minutes, for the module. For the calculations, it will be assumed that a semester is 14 weeks and that a contact session is 50 minutes in duration. This means that one semester hour for this scenario approximates to $14 * 50 = 700$ contact minutes over a semester in terms of lectures. To calculate the practical session component contact minutes for a three hour practical session would result in an additional $14 * 180 = 2520$ contact minutes per semester.

With the South African credit system giving an indication of total time required, it is necessary to look at the breakdown of each module in terms contact hours. At the University of Pretoria, for example, a typical module comprises of a certain number of lectures, tutorials and practical sessions per week (lpw, tpw and ppw respectively). From this, the total contact time for the semester can be calculated.

At the University of Pretoria, a 16 credit module typically has 3 lpw, 1 tpw and 1 ppw. Lectures and tutorials are 50min each while a practical is usually 3 hours. The total contact minutes per week, excluding practical sessions, is therefore given by $(3 * 50) + (1 * 50)$, giving a total of 200 minutes per week. Over 14 weeks, the total contact minutes for such a module would be $200 * 14 = 2800$ minutes. To calculate the semester hours for the lectures, the total minutes needs to be divided by the total contact minutes for a semester. The semester hours for such a module is therefore $2800/700 = 4$. The practical session, being less than 4 hours per week adds 1 semester hour to the total resulting in a module of 5 semester hours.

Knowing the total contact hours for a module helps with module planning and knowing how many hours the student should have at their disposal to attend to assignments and preparation. For the 16 credit module given above, the student should have: notional hours - contact hours, which is $(16 * 10) - ((2800 + (180 * 14))/60) = 160 - 89 = 71$ hours at their disposal.

9.5.4 Four years into three

A Bachelor of Science (BSc) degree in South Africa requires 3 years of full time study giving an NQF exit level of 7 with a minimum credit value of 360 (refer to Table 9.2). For each year in the degree programme, a student is required to complete at least 120 credits or 1200 hours of work. The ABET requirement of at least 120 semester hours (refer to Table 9.4) results in a total of at least $(120 * 700)/60 = 1400$ contact hours as per the rationale given in Section 9.5.3 for a degree programme. Using the general rule, refer to Section 8.4.3, regarding a student working 3 times the classroom time out of the classroom, requires the student to study for at least $1400 * 4 = 5600$ hours to complete the degree.

With Bachelor's degrees in the United States of America being 4 years in length, students are expected to work at least 5600 hours over the 4 years. The HEQF minimum expects 3600 hours of work for a degree programme resulting in a difference of 2000 hours between the ABET and HEQF requirements. The net result is that in order for a South African curriculum to comply with curriculum and accreditation requirements in the United States, the additional year's hours needs to be included over the 3 years of the degree. This translates into a total of approximately 1667 hours per year devoted to study, or 12 hours per day for a 5 day working week. Many students in their first year are unable to cope with the pace and quantity of work expected of them and elect to extend their degree programme by a year. This is in line with the statistics produced by the Council of Higher education from a study which looked at the gradation patterns of the 2005 cohort of students. From this study, 27% of students completed the three year degree programme in the minimum of 3 years. A further 24% had completed their degree programme by 2010 [Council of Higher Education, South Africa]. Of the remaining 49%, 12% dropped out from 2007 to 2010.

9.6 Conclusion

It is clear that there is a need to be able to compare degree programmes on an international basis. The Bologne process was successful in unifying degree structures in Europe and the United Kingdom. This makes it easier to move between institutions in Europe and the UK either during undergraduate studies (3 or 4 year Bachelor's degree programme) or for postgraduate studies, that is for Master's and Doctorate. Accreditation is another mechanism by which degree programmes can be compared by having the degree programme accessed and then as a result of the Accords being seen as *substantially equivalent* to other degree programmes that have been accredited by accreditation bodies who have signed the Accord.

South Africa still follows the 3+1, Bachelor's plus Honours, approach to enter into a Master's programme. This structure for the qualifications

presents South African universities with a challenge to compete internationally and apply for international accreditation. As the qualification structure is not directly comparable with international trends, the ability to accredit degree programmes results in students needing to complete 4 years of work in 3. This decidedly will have an effect on the accreditation of programmes in particularly Computer Science. A further impact is the standing of South Africa in the world economically, which will have a direct impact on rankings of institutions of higher learning, especially in Computer Science.

Chapter 10

Modelling Curricula using Digraphs

10.1 Introduction

This chapter represents the bridge between the theory relating to digraphs and the application of the framework to real-world curriculum specifications. The modelling of the curricula (introduced in Chapters 8 and 9 of Part II) as digraphs (as defined in Part I) will be briefly discussed. This discussion serves as a precursor to the discussion in Chapter 11 on how the framework can be applied to curriculum models.

In the sections that follow, modelling the curriculum volumes as well as a real-world curriculum as digraphs will be discussed. When modelling real-world curricula, capturing the information to be represented by the digraph can be difficult. Suggestions as to how to go about it are discussed. When modelling, it may happen that two different entities represent the same concept. In this case it will be necessary to build equivalence relationships between these entities such as KAs, KUs, topics, etc. Finally a short discussion on improving representations, particularly the model digraph representations, will be discussed.

10.2 Curricula volumes as digraphs

To be able to effectively compare the curricula volumes it is necessary to model them in a uniform manner. The volumes have already defined a linear path between KAs and the topics. Taking this as a basis and including all the KA to topic paths in a single structure will result in a tree-like structure.

Consequently, with this approach a curriculum volume is modelled by using an onion as the metaphor. Refer to Figure 10.1. In the center is the discipline, in this case Computer Science (CS). The next ring represents the KA's, followed by the KU's. The final ring represents topics.

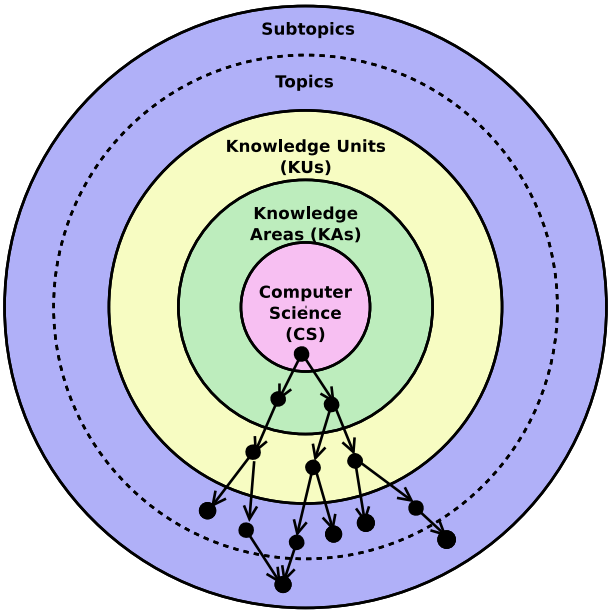


Figure 10.1: Modelling a curriculum volume

As stated in Section 8.3, some topics are further broken down into subtopics, but for modelling purposes, these are simply modelled as topics that are related to other topics. There are instances where topics resort under more than one KU. This fact can be accommodated in a directed graph model in which the KAs, KUs and topics are represented by vertices and the edges represent the relationships between these vertices.

Once the digraph has been defined, the structure and the content of one curriculum volume can be compared with one another. Comparing digraph structures visually often reveals differences and similarities between the graphs. This inspection can be used to expose interesting areas where investigation based on the framework may be warranted and may provide further details.

10.3 Real-word curricula as digraphs

The structure of real-world curricula is dependent on the qualifications framework and the institutional requirements placed on the degree programmes being presented. The most common models for curricula are modules in terms of year-levels; and, modules in terms of their prerequisites beginning with introductory modules and culminating in advanced modules. Using the onion metaphor again, these two structures will result in degree programme structures as represented by Figure 10.2 and 10.3 respectively.

Contrasting Figure 10.2 with the curriculum volume in Figure 10.1 in-

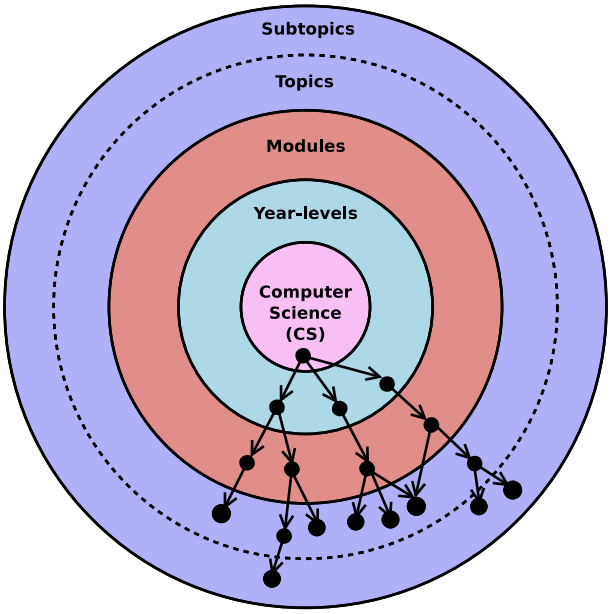


Figure 10.2: Modelling a real-world curriculum in terms of year-levels

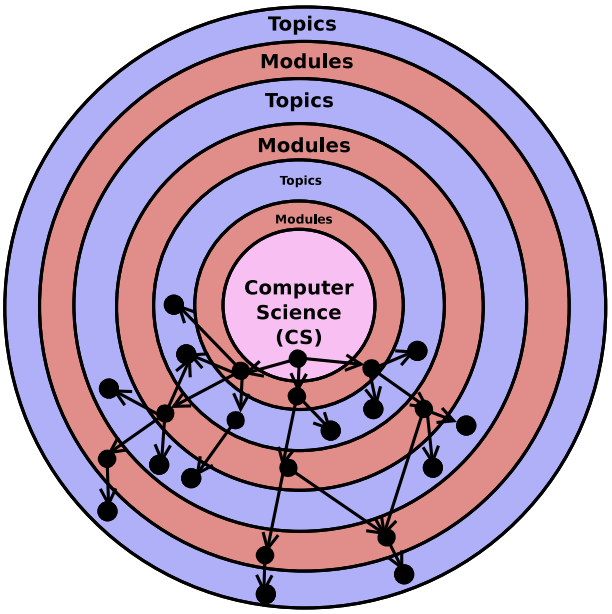


Figure 10.3: Modelling a real-world curriculum in terms of prerequisites

dicates that what is common between the two structures is only that both begin in CS and both have longest paths ending in topics and subtopics. Other than this, what lies between these two extremities is completely different. Modelling a real-world curriculum in terms of prerequisites results in layers of modules that are related and their respective prerequisites. The structure in Figure 10.3 is completely different from the ones in the other two figures, yet the layers representing modules and topics are also in Figure 10.2 which gives the representation of the real-world curriculum in terms of year-levels.

Setting up a the real-world curriculum as a digraph can be tricky, not in so much as creating the basic structure, but in identifying the topics presented in the modules. Capturing the topic information will the presented in more detail in the next section, Section 10.4.

10.4 Capturing topic data

Capturing topic data requires that inputs from a curriculum expert and a module leader. The curriculum expert has a thorough knowledge of the ACM/IEEE curriculum volume that has been modelled and is being used to determine topics in the real-world curriculum. The module leader has in-depth knowledge of the content of the real-world module being presented. The curriculum expert will be modelling the real-word curriculum in terms of the curriculum volume with the input of the module leader who assumes the role of the domain expert. Capturing accurate information in the real-world curriculum requires communication between the curriculum expert and the module leader.

There are two basic approaches for capturing from the particular ACM/IEEE Curriculum Volume topics that are related to the modules in the real-world curriculum. The first approach assumes the volume topics are the only topics that can be used as input to modelling the real-world curriculum. The second approach takes existing modules and their respective topics and tries to match these to the ACM/IEEE curriculum volume topics by matching keywords. These two approached will be discussed in the paragraphs that follow.

Capturing topic data using a spreadsheet. A possible way of implementing this approach is to make use of a spreadsheet similar to the one that accompanies CC2013I, but extended to the topic level. An excerpt of such a spreadsheet is given in Figure 10.4. When using a spreadsheet, each topic in turn needs to be considered in the context of its KA and KU whether it is taught in a particular module or not. This spreadsheet is then transformed into a digraph representing the particular real-world curriculum. This method is highly human inten-

sive and requires the input of both the curriculum expert as well as the module leaders at all stages.

Capturing topic data using keywords. In the alternative method in which keyword matching is used, keywords are automatically extracted from the real-world curriculum descriptions. A list of words that may appear in the real-world curriculum descriptions, such as conjunctions, are ignored. The keywords are then matched to the topics of the curriculum volumes. This method allows for electronic parsing of the module description to automatically do the linking of keywords in the module description to topics in the particular volume. This method partly automates the process of identifying topics in the curriculum volume and can be used to suggest topics that can be used as the initial input for a discussion with the module leader. This method does not remove the curriculum expert, but does reduce the topics that needs to be taken into consideration.

10.5 Modelling equivalences

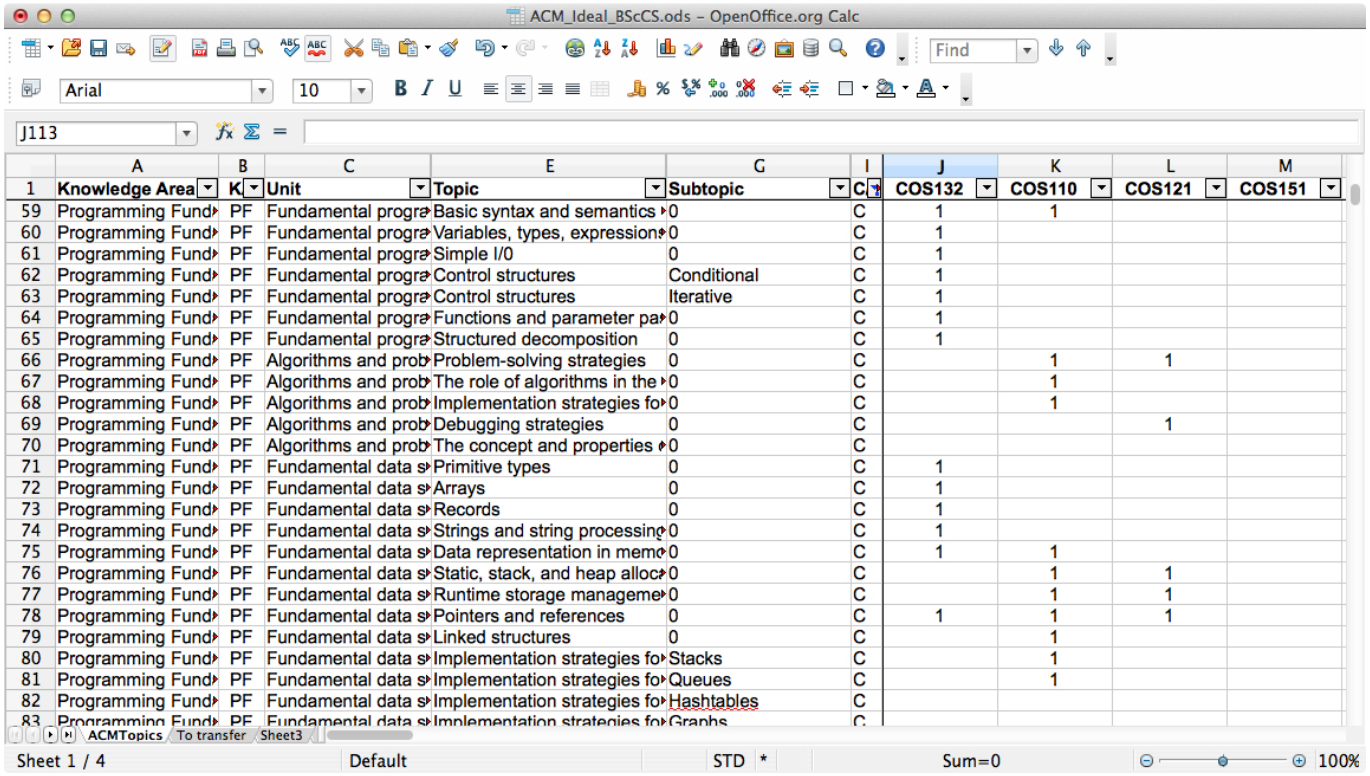
Across the curricula volumes the terminology of some KAs, KUs and topics have subtly altered through the years, yet the aspect being referred to has remained the same. The curricula volumes do provide a narrative regarding changes, but these are mostly on a high, overview level. Exact changes to specifically KUs and topics are not highlighted in detail. This is a problem when the curricula are modelled as digraphs and exact matching techniques, and finding the differences and similarities between the graphs, are performed. In order to ensure that across volumes and in curricula being modelled a standard terminology is used, it is of the utmost importance that equivalences are modelled. The modelling of the equivalences must not change the meaning of the digraph.

To illustrate the modelling of equivalences, consider the KU linked to the SE KA that represents *requirements*. Over the curricula volumes subtle changes have been applied to it. For the purposes of the discussion, the unique identifier assigned to the KU will be used.

***Software requirements and specifications*:** The unique identifier assigned to it for modelling purposes is U0119 for CC2001.

***Requirements specifications*:** In CS2008, the description of the KU changed slightly. The unique identifier given for modelling purposes is U0119_1.

***Requirements engineering*:** In both CS2013S and CS2013I, the text of the KU changed to Requirements engineering. The unique identifier assigned to this KU is U0206.



	A	B	C	E	G	I	J	K	L	M
1	Knowledge Area	K	Unit	Topic	Subtopic	C	COS132	COS110	COS121	COS151
59	Programming Fund	PF	Fundamental progra	Basic syntax and semantics	0	C	1	1		
60	Programming Fund	PF	Fundamental progra	Variables, types, expressions	0	C	1			
61	Programming Fund	PF	Fundamental progra	Simple I/0	0	C	1			
62	Programming Fund	PF	Fundamental progra	Control structures	Conditional	C	1			
63	Programming Fund	PF	Fundamental progra	Control structures	Iterative	C	1			
64	Programming Fund	PF	Fundamental progra	Functions and parameter pa	0	C	1			
65	Programming Fund	PF	Fundamental progra	Structured decomposition	0	C	1			
66	Programming Fund	PF	Algorithms and prob	Problem-solving strategies	0	C		1	1	
67	Programming Fund	PF	Algorithms and prob	The role of algorithms in the	0	C		1		
68	Programming Fund	PF	Algorithms and prob	Implementation strategies fo	0	C		1		
69	Programming Fund	PF	Algorithms and prob	Debugging strategies	0	C			1	
70	Programming Fund	PF	Algorithms and prob	The concept and properties	0	C				
71	Programming Fund	PF	Fundamental data s	Primitive types	0	C	1			
72	Programming Fund	PF	Fundamental data s	Arrays	0	C	1			
73	Programming Fund	PF	Fundamental data s	Records	0	C	1			
74	Programming Fund	PF	Fundamental data s	Strings and string processing	0	C	1			
75	Programming Fund	PF	Fundamental data s	Data representation in memo	0	C	1	1		
76	Programming Fund	PF	Fundamental data s	Static, stack, and heap alloc	0	C		1	1	
77	Programming Fund	PF	Fundamental data s	Runtime storage manageme	0	C		1	1	
78	Programming Fund	PF	Fundamental data s	Pointers and references	0	C	1	1	1	
79	Programming Fund	PF	Fundamental data s	Linked structures	0	C		1		
80	Programming Fund	PF	Fundamental data s	Implementation strategies fo	Stacks	C		1		
81	Programming Fund	PF	Fundamental data s	Implementation strategies fo	Queues	C		1		
82	Programming Fund	PF	Fundamental data s	Implementation strategies fo	Hashtables	C				
83	Programming Fund	PF	Fundamental data s	Implementation strategies fo	Graphs	C				

Figure 10.4: Example of a spreadsheet used to model the BSc CS degree programme

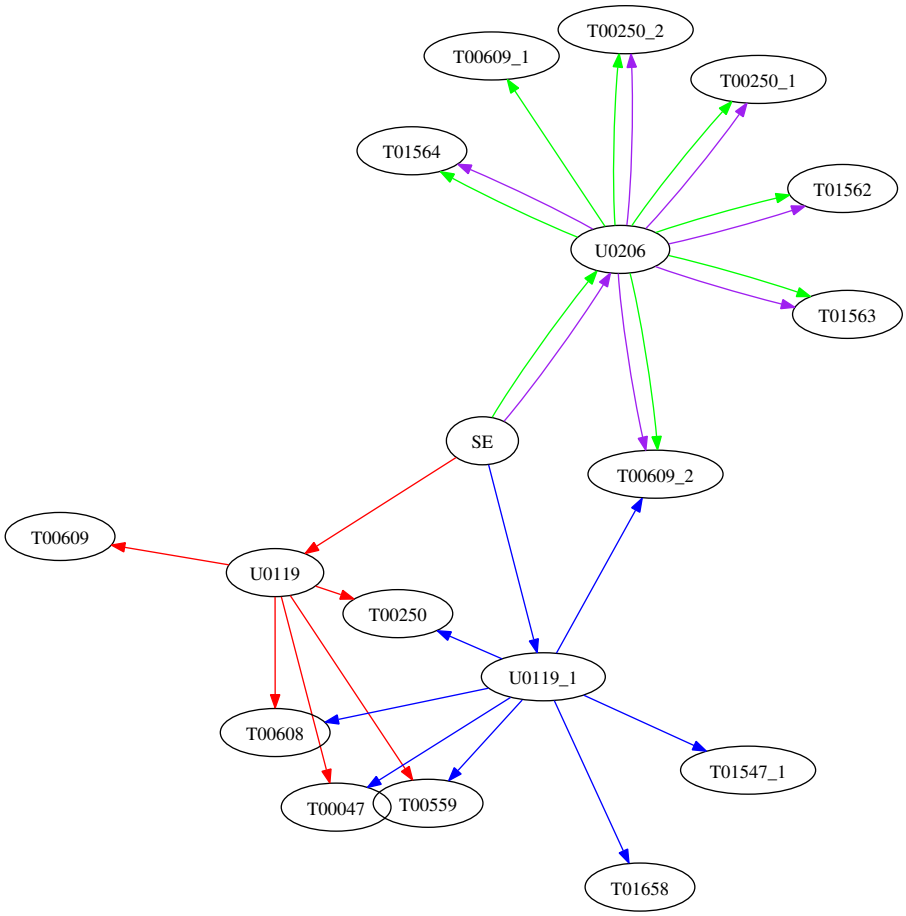


Figure 10.5: Representation of the KU representing requirements

A visual representation of these KUs across the curriculum volumes, related topics that link to the KU descriptions and the KA, are represented by the sub-digraph in Figure 10.5. From the figure it can be seen that U0119, U0119_1 and U0206 share topics. U0119 and U0119_1 share four topics between them. U0206 and U0119_1 shares one topic. There is one topic unique to U0119 and therefore CC2001. CS2008 contains two topics unique to it via U0119_1. Six topics are common to CS2013S and CS2013I of which five are not shared with one of the other two curriculum volumes.

Using the shared topics as an indication and some domain knowledge, it can be said that U0119, U0119_1 and U0206 are equivalent KUs. For this reason they need to be modelled as such. For traceability and matching purposes, it is necessary that the KUs do not loose their unique identity. They also however need to remain unified. In order to achieve this, the vertices are arranged in such a way that they are all reachable from the SE KA and that they all continue to the relevant topics. This can be achieved by

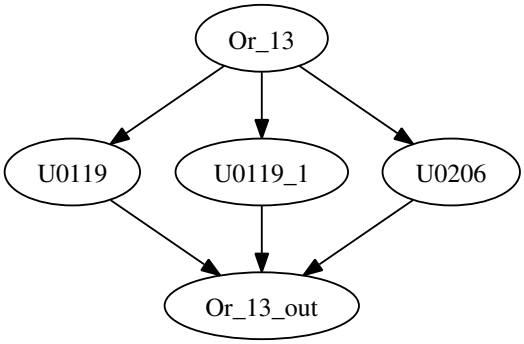


Figure 10.6: *Or-subgraph* for the KU representing requirements

relying on a so-called *or-subgraph*. The rule to apply to build an *or-subgraph* for these vertices is given by Rule 10.1.

Rule 10.1 (*Vertex equivalence - or-subgraph*)

For vertices v_1, v_2 to v_n which are equivalent, the following construction provides a so-called or-subgraph.

$(v_1, v_{1_N}, L_{1_N}), (v_2, v_{2_N}, L_{2_N}), ..., (v_n, v_{n_N}, L_{n_N}) \longrightarrow$
$(or_{name}\_in, v_1, L_1), (v_1, or_{name}\_out, L_{name_1}),$
$(or_{name}\_in, v_2, L_2), (v_2, or_{name}\_out, L_{name_2}), ...,$
$(or_{name}\_in, v_n, L_n), (v_n, or_{name}\_out, L_{name_n})$

where N represents any number of out-bound edges from 1 to N.

The resulting structure after the application of Rule 10.1 to Figure 10.5 is given in Figure 10.6. This structure will be referred to as Or-subgraph_13.

The *or-subgraph* seen in Figure 10.6 must be incorporated into the subgraph given in Figure 10.5. This is achieved by applying the rule given by Rule 10.2. Figure 10.7 shows the result of linking Or-subgraph_13 into the sub-digraph in Figure 10.5.

Rule 10.2 (*Linking the or-subgraph*)

$(s, v_1, L_1), (v_1, v_{1_N}, L_{1_N}), (s, v_2, L_2), (v_2, v_{2_N}, L_{2_N}), ...,$
$(s, v_n, L_n), (v_n, v_{n_N}, L_{n_N})$
$\longrightarrow$
$(s, or_{name}\_in, L_{or_{name}}),$
$or\text{-}subgraph_{name},$
$(or_{name}\_out, v_{1_N}, L_{1_N}), (or_{name}\_out, v_{2_N}, L_{2_N}), ..., (or_{name}\_out, v_{n_N}, L_{n_N})$

where N represents any number of outbound edges from 1 to N.

The subgraph connected by edges of a given colour in Figure 10.7, for

example the red edges for CC2001, represents the same information as was originally represented by the vertices connected by the red edges in Figure 10.5. In the former case, however, there is a path from SE through either U0119, U0119_1 or U0206 instead of through U0119 only.

For every application of Rules 10.1 and 10.2, two additional vertices are included in the original digraph. A digraph, G, with $\mid V_G \mid$ vertices and X sets of equivalent vertices will result in a digraph with $\mid V_G \mid +2X$ vertices. Similarly, the number of edges in the digraph after the equivalences have been included will be at least $\mid E'_G \mid +3X$. This will account for the increase in the cardinality in terms of vertices and edges when equivalences are applied to the real-world examples in the application to curriculum development discussed in Chapter 11.

10.6 Improving representations

By considering the quantities $M \setminus C$ and $C \setminus M$, the model and in some instances the ideal as well, can be improved. The improvements for each of the quantities will briefly be discussed. The use of the quantities for improvement has already been demonstrated in Section 6.5 of Chapter 6 when the toy application was discussed, and in Chapter 7 when the outcomes of the algorithm $\mathcal{T}$ were presented. It will also be used in Chapter 11 when comparing curricula volumes and real-world curricula.

Improving the representation by considering quantity $M \setminus C$.

The quantity $M \setminus C$ represents all the vertices and edges in M that are not in C. In most cases these are structural, such as year-levels or module codes. It may be that these also include topics that are not represented in C. In this case these are extraneous[1] topics and are not necessary for the comparison under review. Removing these topics will typically result in a better match. This quantity, $M \setminus C$, should be evaluated as a possible means of exposing or suggesting equivalences.

Quantity $M \setminus C$ is also useful when used in conjunction with $I \setminus M$ to determine the integrity of the digraphs representing the ideal and model. Assume the algorithm $\mathcal{T}$ is run twice for two digraphs A and B. The first run is with A as the ideal and B as the model for which the results are captured. The second time the algorithm is run, B represents the ideal and A the model and the results are once again captured. The cardinalities of the vertices and edges for the quantity $I \setminus M$ for the first run must equal the cardinalities of the vertices and edges for $M \setminus C$ for the second. If the reverse is also true, that is the cardinalities of the vertices and edges for the quantity $I \setminus M$ for the second run equals the cardinalities of the vertices and edges for $M \setminus C$ of the second run, then the digraphs A and B can be said

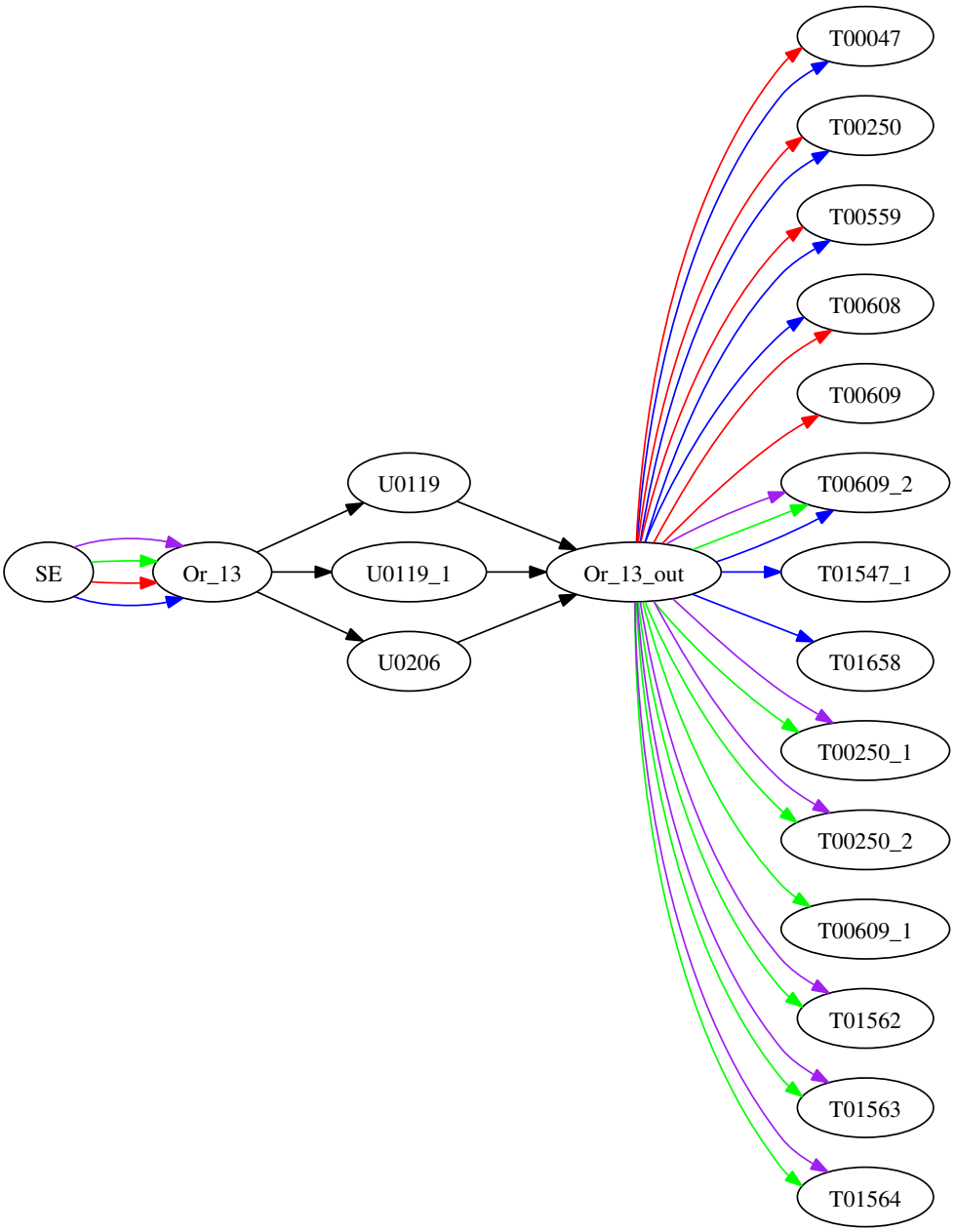


Figure 10.7: Linking Or-subgraph_13 into the digraph

to be consistent. Table 10.1, illustrates this for curriculum volumes CS2013S and CS2013I. From the table it can clearly be seen that the cardinalities for $R(I \setminus M, I)$ and $R(M \setminus C, I)$ have flipped when the digraphs representing the ideal and model have been swapped around after the algorithm is rerun. If the cardinalities are not the same between the forward and reverse runs, further investigations into the sets represented by the cardinalities may reveal the inconsistencies within either the model or the ideal. In many instances these inconsistencies can be attributed to errors made during the modelling of I and M. These differences need to be resolved before an accurate comparison can be made.

Improving the representation by considering quantity $C \setminus M$. The quantity $C \setminus M$ represents the sets of edges and vertices that are in C but not in M. These edges and vertices have been inferred[1] from I using the information from M. By considering the inclusion of these edges and vertices into M, the compliance of M with regards to I should improve.

Ratio $(R(\mathcal{X}, \mathcal{Y}))$	CS2013S as I, CS2013I as M		CS2013I as I, CS2013S as M	
	$\mid V_{\mathcal{X}} \mid$	$\mid E'_{\mathcal{X}} \mid$	$\mid V_{\mathcal{X}} \mid$	$\mid E'_{\mathcal{X}} \mid$
$R(I, I)$	683	683	732	732
$R(M, I)$	732	732	683	683
$R(C, I)$	593	593	597	597
$R(I \setminus C, I)$	90	90	135	135
$R(I \setminus M, I)$	**93**	**208**	**142**	**257**
$R(M \setminus C, I)$	**142**	**257**	**93**	**208**
$R(C \setminus M, I)$	3	118	7	122

Table 10.1: Set quantity cardinalities for CS2013S and CS2013I

10.7 Conclusion

In Part I, when the theory regarding the Graph Comparison Framework was presented, no mention was made of how the information in the real-world that is represented by the digraphs would be captured nor what problems may arise in doing so. This chapter has bridged the gap between the application of the framework on a theoretical toy application and how it will be applied in the real-world to the ACM/IEEE curriculum volumes and a real-world degree programme specification in Chapter 11.

[1]The concepts of extraneous and inferred were defined in Section 6.3.1.

Chapter 11

Application of the Framework to Computing Curricula

11.1 Introduction

This chapter will illustrate how the framework presented in Chapter 6 can be applied to curricula relating to Computer Science. Five areas in which the Graph Comparison Framework can be applied to Computer Science curricula have been identified. The broad categories for these areas are:

Area A: Comparison of ACM/IEEE Computer Science Curriculum volumes

Area B: Comparison of a real world curriculum against one of the ACM/IEEE Computer Science Curriculum volumes to determine how it compares.

Area C: Comparison of one real-world curriculum with another real-world curriculum to determine similarities and differences.

Area D: Comparison of the ACM/IEEE Computer Science Curriculum volumes with accreditations structures such as ABET.

Area E: Comparison of a real-world curriculum with an accreditation structure.

This list of areas is by no means complete. Many combinations of curricula specifications, accreditation requirements and real-world curricula can be applied where one needs to be compared to the other. It is also conceivable that the framework could be used to determine progress during a curriculum development and improvement process.

To illustrate how the Graph Comparison Framework can be applied to curricula comparison, it will be applied to the first two areas specified above.

Three scenarios have been identified. The first two scenarios relate to Area A. The third scenario considers a real-world curriculum and compares it to the curriculum volumes. The application of the framework to the other areas not discussed in this chapter will follow a similar rationale.

The discussion related to each scenario will present aspects of how the framework can be applied to curriculum comparison and development. When applying the framework in a real-world context, all results presented by the framework must be considered to form a holistic picture of the comparison.

For purposes of uniformity, edges between vertices on the graph visualisation diagrams are presented in different colours according to the curriculum volume. CC2001 will be presented in red, CS2008 in blue, CS2013S in green and CS2013I in purple. The real-world curriculum is presented in orange and the complier in yellow. This colour coding will be used where possible when representing information on radar charts and other diagrams as well.

As stated before, the ACM/IEEE curricula volumes discuss changes that have taken place in the CS-BoK. For example, [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2013, Page 200] discusses the changes that have taken place between CS2001 and CS2013I at a macro-level. When modelling the curricula volumes as digraphs and comparing them with each other, the comparison takes place on a micro-level.

11.2 Scenario 1: Comparing the core aspects of the curricula volumes

11.2.1 Scenario overview

In this scenario, the Graph Comparison Framework will be applied to CC2001, CS2008, CS2013S and CS2013I, illustrating and identifying the subtle changes that have taken place between the curricula volumes. The changes at KA and KU level will be considered. Equivalences will be incorporated into the comparison. The changes with regards to topics will be highlighted and discussed by applying the Difference Comparison component to the digraphs and further investigating the difference sets. This scenario will only be applied to the core aspects of the volumes. Only the core aspects are considered because representations taking electives into account as well, will detract from the explanation of the application of the framework. The core aspects already account for a vertex count in some representations of approximately 700. Adding the elective topics will in many instances double the current vertex cardinality of the digraph.

As CC2001 is the last official curriculum volume, CS2013S and CS2013I will primarily be compared with it. CS2013S and CS2013I will also be compared with one another to highlight the changes made after the contributions

from the public have been taken into account.

11.2.2 Graph visualisation

Table 11.1 visualises the core aspects of the four curricula volumes as digraphs. (The `neato` command of the GraphViz software package was used to provide the so-called "spring model" layouts. These were briefly discussed in Section 6.4.1.) Although much of the detail is obscured, the visualisation nevertheless suggests that the volumes have changed over time. To determine what has remained the same, a more fine-grained comparison between volumes is necessary.

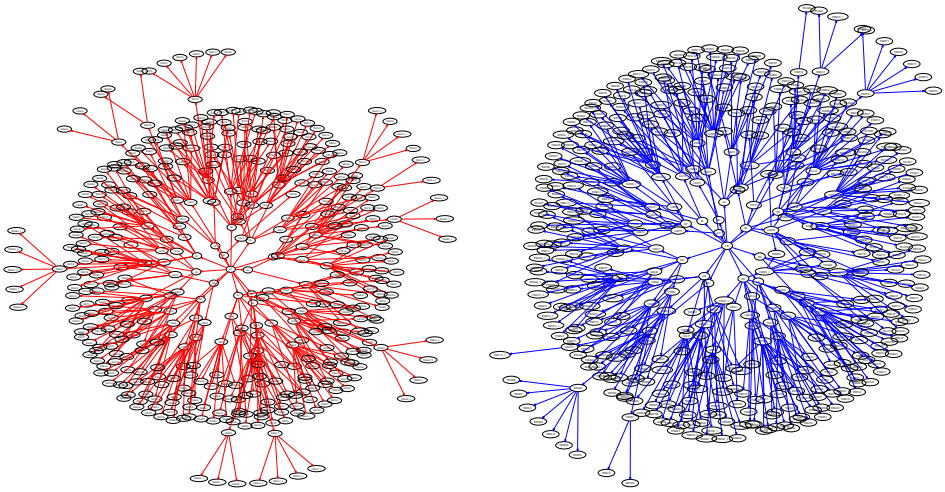
From the `neato` spring model results in Table 11.1 it can be inferred that CS2013S does not have explicit subtopics. An in-depth study of the curriculum volume confirms this. CC2001, CS2008 and CS2013I all seem to reflect that there are subtopics defined. CS2013I also has what can be referred to as subsubtopics defined. When closely studying the respective curriculum volumes, what is highlighted by the spring models can be seen in the volumes. A closer investigation of the content of the spring models will provide more detail into the similarities and differences between the curriculum volumes.

In order to derive more information from the digraphs it is necessary zoom into specific areas within the graph. Table 11.2 illustrates what can be seen when aspects of the spring models are enlarged. The Human Computer Interaction KA and KUs highlighted in the table will be discussed in more detail in Scenario 2, presented in Section 11.3.

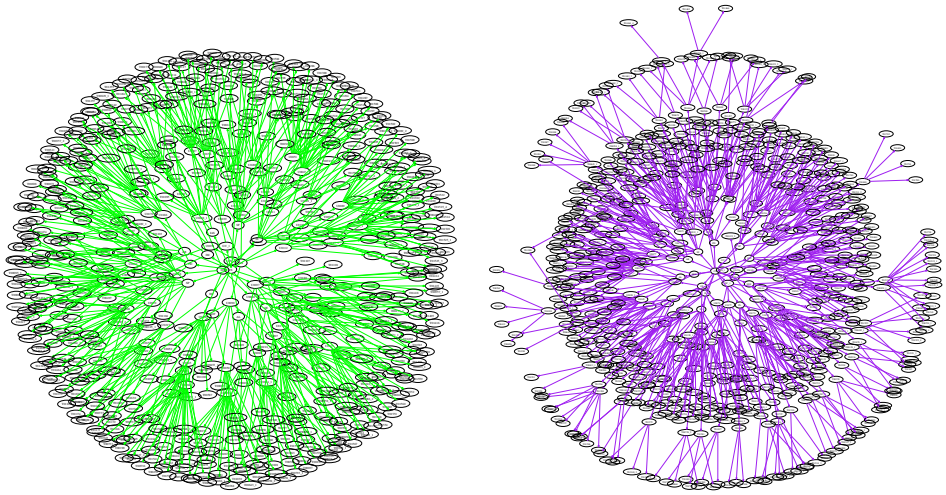
11.2.3 Difference comparison

The digraphs representing the core aspects of the curricula volumes will be compared. The first comparison will consider how each of the review curriculum volumes, CS2008, CS2013S and CS2013I, differ from the last complete Computer Science curriculum volume, namely CC2001. In this comparison CC2001 will be regarded as the ideal. CS2008, CS2013S and CS2013I will each in turn represent the model and be compared with CC2001. The second comparison is between CS2013S and CS2013I. CS2013S will be considered the ideal and CS2013I the model. This comparison should indicate the impact public comment of CS2013S had in the development of CS2013I.

For each comparison, that is for: CC2001 as ideal (I) with CS2008 as model (M); CC2001(I) with CS2013S(M); CC2001(I) with CS2013I(M); and CS2013S(I) with CS2013I(M), three representations for each of the digraphs will be considered. The first digraph representation will be exactly as published in the respective curriculum volume. The second digraph representation to be considered will take equivalences in KAs into account. The third and final digraph representation will consider equivalences for both KAs and

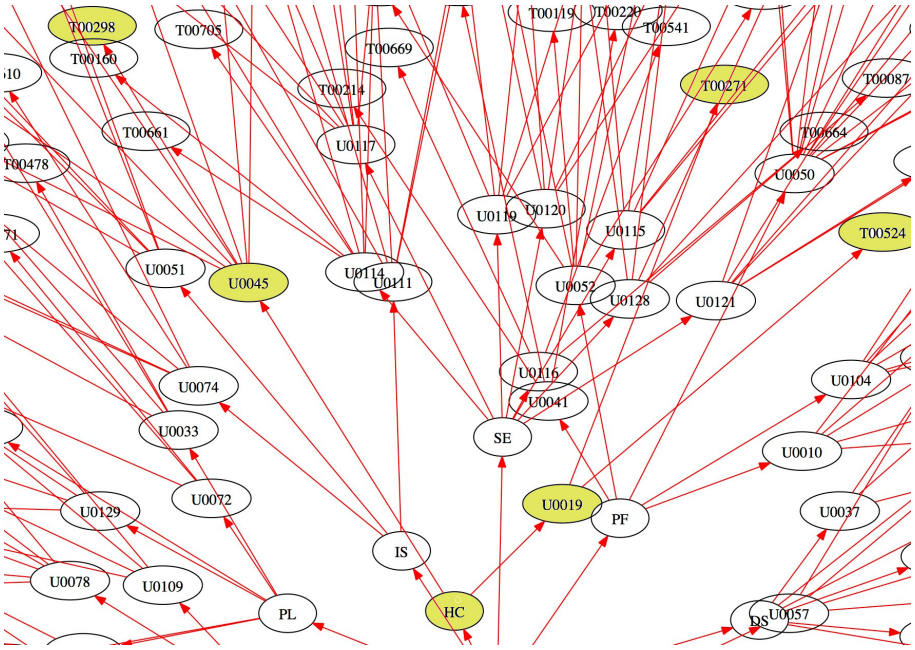


CC2001 CS2008

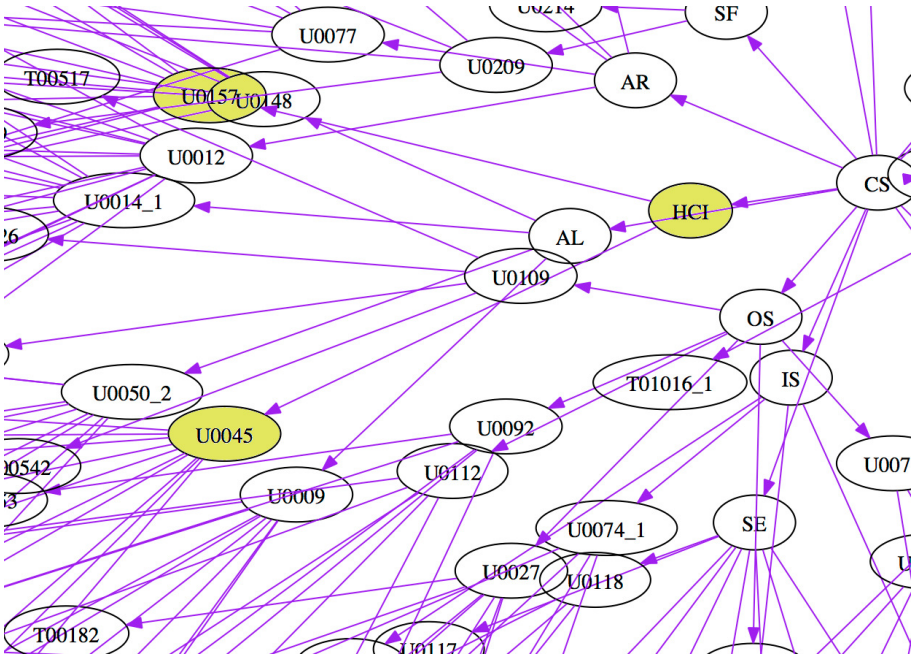


CS2013S CS2013I

Table 11.1: Core aspects of the curriculum volumes



CC2001



CS2013I

Table 11.2: Zoomed into CC2001 and CS2013I HC/HCI in Table 11.1

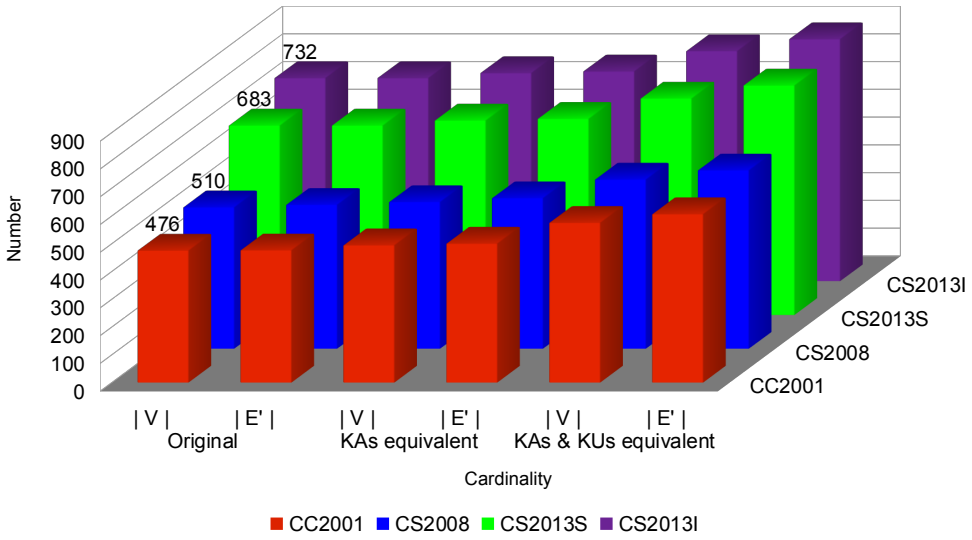


Figure 11.1: Respective sizes of the curricula volume digraphs

KUs.

Respective sizes of the digraphs across curricula volumes.

The sum of the KAs, KUs and topics specified in the core of a curriculum volume is one less than the size of the associated digraph. This is because an additional "entry point" vertex, labelled as CS, appears in each diagraph, linking to each of the KAs. (An example of such an "entry point" was previously encountered in Figure 10.1.) The respective digraph sizes, $\mid V \mid$, and the associated number of edges in each case, $\mid E' \mid$, are shown in the "Original" major row of Table 11.3. (The data to obtain $\mid V \mid$ is available in Figure 8.4.) The information is visually displayed in Figure 11.1.

Table 11.3 and Figure 11.1 further show how the cardinality of the vertices and edges increase per volume when Rule 10.2 is applied to equivalent KAs. If the rule is further applied to include equivalent KUs the result is shown by the final major row in the table and the final two rows of bars in the figure.

Figure 11.1 and Table 11.3 show that the growth in the curriculum size from CS2008 to CS2013S is significant. A total of 173 (34%) vertices have been added and 164 (32%) edges from CS2008 to CS2013S. A further growth of 49 (7%) vertices and edges between CS2013S and CS2013I indicates that changes took place with regards to content in the curriculum volumes. An increase of 44% in vertices and 42% in edges between CS2008 and CS2013I has taken place.

		CC2001	CS2008	CS2013S	CS2013I
Original	$\mid V \mid$	476	510	683	732
	$\mid E' \mid$	477	519	683	732
KAs equivalent	$\mid V \mid$	495	529	701	750
	$\mid E' \mid$	501	543	707	756
KAs & KUs equivalent	$\mid V \mid$	573	607	777	826
	$\mid E' \mid$	607	643	827	872

Table 11.3: Values showing the respective cardinalities of the curriculum volume digraphs

Respective sizes of the difference sets across curricula volumes.

The resulting cardinalities and ratios of the Graph Trans-morphism algorithm for the comparison of the core aspects of the curricula volumes as described in the previous section are given in Section E.1 of Appendix E. The section in the appendix is structured as follows:

- There are three tables in which CC2001 serves as I and CS2008, CS213S and CS2013I serve as M, respectively.
- A fourth table in which CC2013S serves as I and CS2013I serves as M.

Each table has the following structure:

- Nine rows giving $R(X, Y)$, where X and Y appear in various combinations of I, M and C, as discussed in Section 6.3.1.
- Three major columns labeled "Original", "KA equivalences" and "KA and KU equivalences" as used in Table 11.3.
- Each major column in each of these tables has sub-columns labeled $\mid V_{\mathcal{X}} \mid$, $\mid E'_{\mathcal{X}} \mid$, $V_{\mathcal{X}}^{ratio}$ and $E_{\mathcal{X}}^{ratio}$. The meaning of these sub-column labels was explained in Section 6.3.1.

It should be noted that the equivalences bloat the vertex and edge cardinalities for the respective digraphs. The bloating of vertices and edges was discussed in Section 10.5. To ensure that the comparisons between the different graph combinations, when taking equivalences into account, remain constant, exactly the same equivalence vertices and edges have been included in all digraph representations being compared. A set difference between the digraphs will therefore not result in side effects from the addition of the equivalence vertices and edges as all the additional vertices and edges will not be taken into account in the results of the operation.

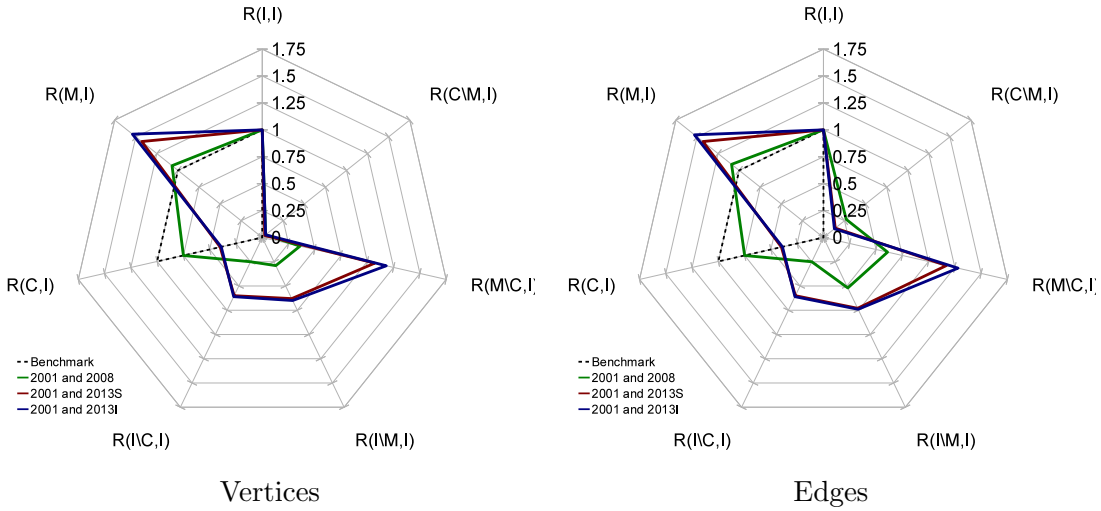


Figure 11.2: Scenario 1: Original - Radar charts

The values given in the tables in the appendix, confirm that there was marginal change in core content of the curricula volumes CC2001 and CS2008. However, some rearranging of relationships took place. This rearranging is confirmed by the relatively high edge ratios for $R(I \setminus M, I)$ and $R(M \setminus C, I)$ (0.52 and 0.61 respectively) in the column representing the original curricula volume.

The values presented in Tables E.2 and E.3 (which compare CC2001 with CS2013S and CS2013I, respectively) indicate that the differences between these volumes is more significant than in the case of CC2001 and CS2008. The comparisons will be highlighted in the section that follows where the differences are visualised on radar charts.

11.2.4 Difference visualisation

The difference visualisations presented are all derived from the information given in Tables E.1 to E.4 in Appendix E. The first visualisation will consider the curricula volumes as originally specified, while the second discussion will show whether including equivalences makes a difference in the matching of the volumes.

Visualisation of differences between the curricula volumes.

The visualisation of the information representing the curricula volumes as originally defined from the tables in Appendix E, Section E.1 are given by the radar charts in Figure 11.2 for the comparisons with CC2001 as I. From these visualisations, the differences between the curricula volumes as discussed in the previous difference comparison section can be seen.

From the figures it is clear that CS2008 looks very much like CC2001 in terms of vertices. The ratio $R(C, I)$ indicates that not all vertices in CS2008 are in CC2001. C has roughly 25% less vertices than CS2008 even though the respective cardinality of CS2008 is marginally more than that of CC2001, as shown by $R(M, I)$. The representation of the edges follows a similar trend as for vertices. However the magnitude of the differences is more pronounced when viewing $R(I \setminus M, I)$ and $R(M \setminus C, I)$.

Inspection of the difference sets, particularly for $M \setminus C$ and $C \setminus M$ indicates that there are indeed differences between CC2001 and CS2008. Table 11.4 summarises the extent of these differences. This data indicates that there are a total of 27 extraneous KUs in CS2008 and 18 inferred KUs in C. These KUs naturally have a knock-on effect on the topics for which there are no matches in terms of edges. With regards to topic counts themselves, there are 145 extraneous topics in CS2008. No topics have been inferred in C.

The curricula volumes CS2013S and CS2013I have significantly more vertices and edges than CC2001 as shown by $R(M, I)$ in Figure 11.2. Indeed, the differences with respect to CS2013I is somewhat larger than with respect to CS2013S. From $R(M \setminus C, I)$ it can be seen that structurally the curricula volumes have also changed. The complier, however, does match well to the model as can be seen by ratio $R(C \setminus M, I)$, this means that very little was inferred.

The visual comparison of the original of CS2013S with CS2013I will be discussed in the next paragraph. This paragraph will discuss equivalences between elements in the digraphs representing the curriculum volumes.

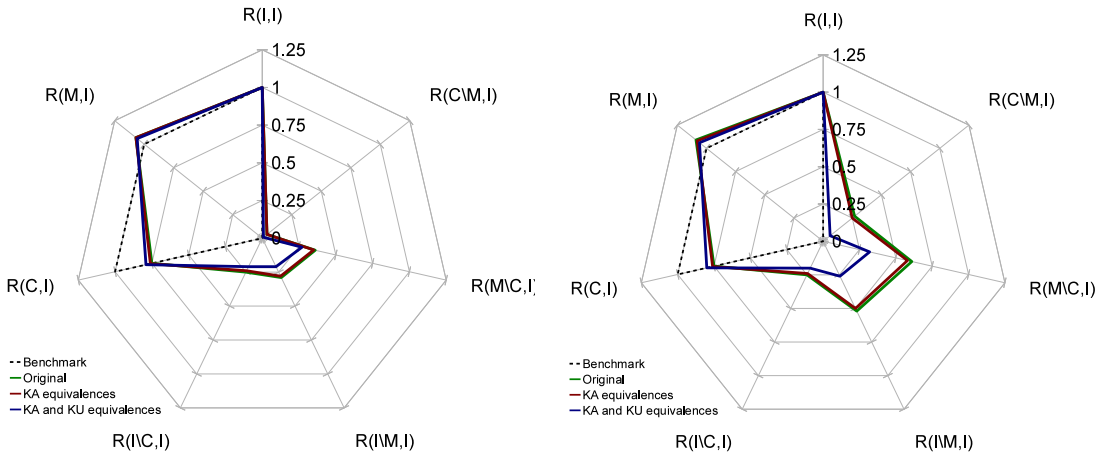
Visualisation of the impact the equivalences have of the matching of curricula volumes.

Figure 11.3 presents the vertex and edge radar charts with CC2001 representing the ideal and CS2008 the model. On each radar chart the curricula as specified in the curricula volumes is given along with the application of the equivalences for KA and KA with KU also applied. From the figures it is clear that the equivalences do increase matching, be it marginally. Comparison of CC2001 and CS2008 with the equivalences for both KAs and KUs taken into account result in the additional values presented in the last columns of Table 11.4 for vertices. These values correspond to the values for the cardinality of vertices given in Table E.1 by calculating the sum of the respective KU and topic entries. The comparisons for CC2001 with CS2013S and CS2013I follow similar trends.

The radar charts in Figure 11.4 compare CS2013S modelled as the ideal and CS2013I as the model. The ratio $R(M, I)$ in the radar charts indicates that CS2013I when modelled as a digraph has more vertices and edges than CS2013S. This is in line with the results in Table 11.3 which show an in-

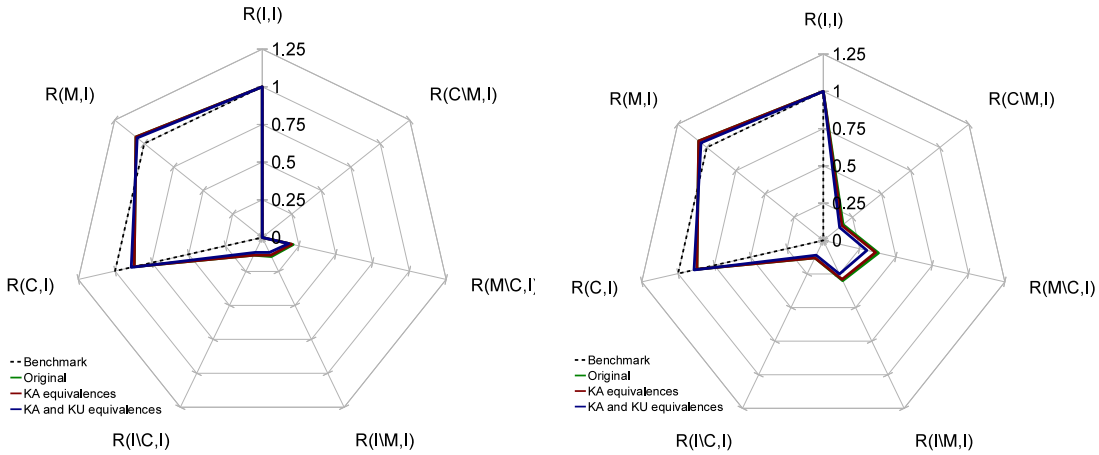
	Original		**KA equivalences**		**KA and KU equivalences**	
	KUs	Topics	KUs	Topics	KUs	Topics
$M \setminus C$	27	145	17	155	11	145
$C \setminus M$	18	0	18	0	2	0

Table 11.4: Cardinality of vertices in the difference sets $M \setminus C$ and $C \setminus M$ for CC2001(I) and CS2008 (M)



Vertices Edges

Figure 11.3: Scenario 1: Equivalences CC2001 and CS2008



Vertices Edges

Figure 11.4: Scenario 1: Equivalences CC2013S(I) and CS2013I(M)

crease in vertices and edges from 683 to 732 from CS2013S to CS2013I. The ratio $R(C \setminus M, I)$, specifically for vertices indicates that the complier is a good representation of the model. $R(M \setminus C, I)$, indicates that CS2013I contains vertices and edges that are not in CS2013S. Further investigation of specifically the vertices will highlight where to look for possible equivalences and changes that took place. With the equivalences of KAs and KAs with KUs following a similar trend, it would mean that the topics need to be investigated in more detail.

11.2.5 Discussion

Two observations can be made from the preceding discussions. The first is that the curricula have changed rather significantly from CC2001 to CS2013, both Strawman and Ironman. This change is both in terms of content and structure. The shift in content can be seen in Figure 11.2. Closer inspection of the sets $C \setminus M$ and $M \setminus C$ will give an indication of where the differences lie between the curricula volumes.

The second observation relates to the addition of the equivalences. It is clear from Figures 11.3 and 11.4 that the addition of the equivalences does improve matching. It is therefore necessary to also include equivalences on the topic level. Scenario 2 will illustrate topic level equivalences for a subgraph of each curriculum volume representing the Human Computer Interaction KA. Finding topic equivalences for all the KAs in each curriculum volume will be left for future work.

11.3 Scenario 2: Details regarding the Human Computer Interaction KA

11.3.1 Scenario overview

This scenario compares the core aspects of the Human Computer Interaction KA as defined in the curricula volumes. This comparison was previously done and reported on in Marshall [2012] with respect to CC2001, CS2008 and CS2013S. The comparison with CS2013I has been included in the discussion of this scenario due to the fact that CS2013I had not been released when the paper was submitted and accepted. The inclusion of equivalences has also been added—something that had not previously been done for this KA.

11.3.2 Graph visualisation

Figure 11.5 presents the core aspects of the Human Computer Interaction KA. The colour of the edges is in keeping with the conventions as set out in the introduction: CC2001 is red, CS2008 is blue, CS2013S is green, and CS2013I is purple. From the figure it can be seen that the KA underwent a name change in CS2013I . Also, two KUs, identified as U0019 and U0221, have been dropped in the CS2013 versions of the curricula volumes. U0019 was defined in CC2001 only and U0221 was defined in CS2008 only. The KU, U0045, is present in all four volumes. U0157 was included for the CS2013 volumes and underwent a change between CS2013S and CS2013I.

11.3.3 Difference comparison

From the visualisation of the digraphs given in Figure 11.5, there are areas of interest in the Human Computer Interaction KA. The first is between CC2001 and CS2008, the second is between the 2013 Strawman and Ironman volumes. The Graph Comparison Framework, beginning with the Trans-morphism Algorithm, will be applied to each of these areas to determine if there are any equivalences that may need to be taken into consideration when comparing the volumes.

Graph Trans-morphism Algorithm results for CC2001 and CS2008.

Application of the Graph Trans-morphism Algorithm to CC2001 and CS2008, with CC2001 representing the ideal and CS2008 representing the model and *vice versa* results in the compliers given in Figure 11.6. Topic T00524 is linked to U0019 in the complier for CC2001 and to U0221 in the complier for CS2008. On closer inspection it can be seen that U0019, *Building a simple graphical user interface*, and U0221, *Building GUI interfaces*, are equivalent.

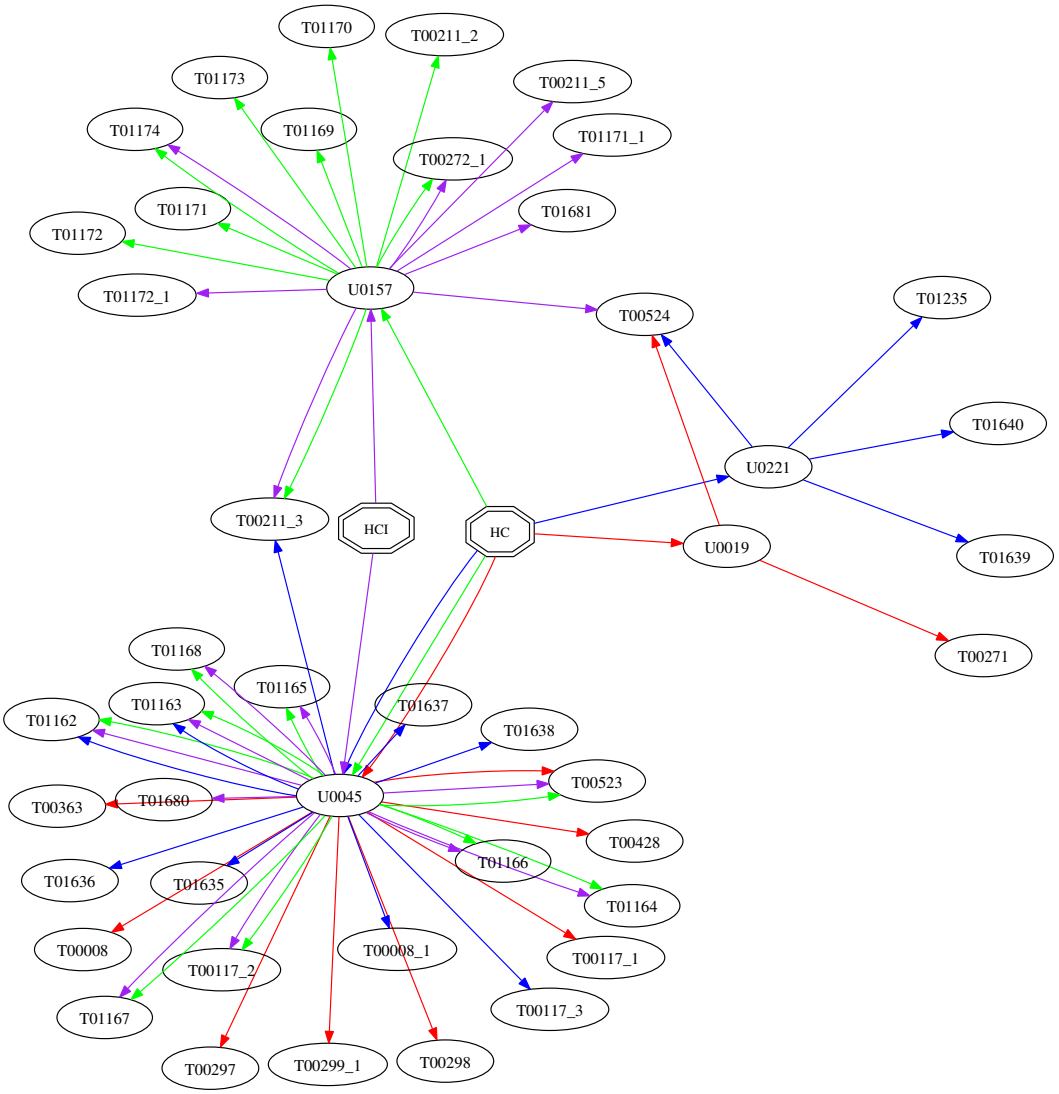


Figure 11.5: Representation of the Human Computer Interaction (HC and HCI in 2013I) KA

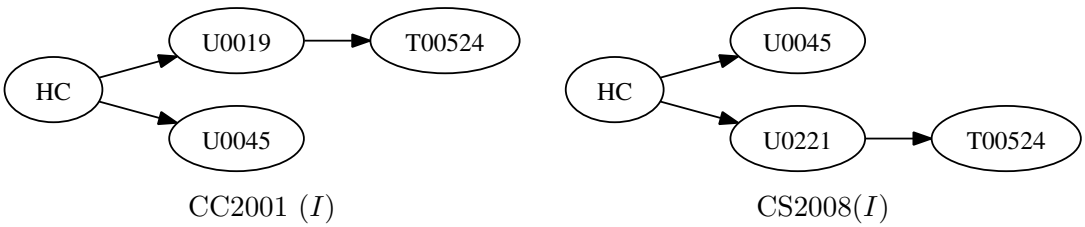


Figure 11.6: Scenario 2: Compliers for CC2001 as I and CS2008 as M and vice versa

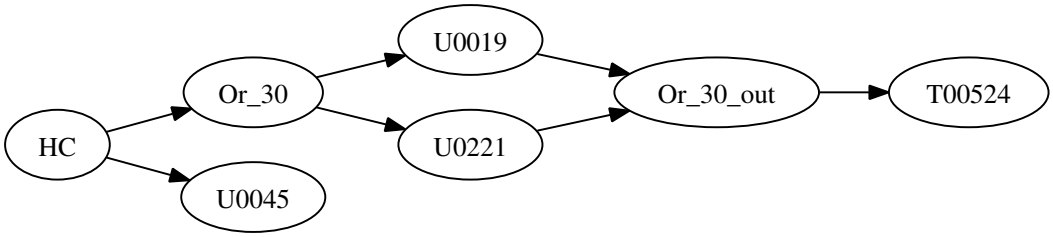


Figure 11.7: Scenario2: Complier for CC2001 and CS2008 with *or-subgraph*

By including an *or-subgraph*, the resultant complier in Figure 11.7 is the same for CC2001 as ideal and CS2008 as model as it is for CS2008 as ideal and CC2001 as model. From this figure it can be confirmed that there are no topics in common between CC2001 and CS2008 that are linked to KU U0045 and that there is still only one topic, as expected, in common between U0019 and U0221.

Investigation of the difference sets in terms of vertices, given in Table 11.5, indicates that all vertices in the complier are also in the model, $C \setminus M = \emptyset$. The reverse is not true; there are vertices in the model which are not in the complier, $M \setminus C$. It is also interesting to note that the sets for $I \setminus C$, $I \setminus M$ for CC2001 as the ideal and CS2008 as the model, and $M \setminus C$ for CS2008 as the ideal and CC2001 as the model are the same. Closer investigation of both the sets for $M \setminus C$ may reveal similarities in topics. Topics are distinguishable by there vertex names beginning with "T". This investigation is left for the Observation section of this discussion.

Graph Trans-morphism Algorithm results for CS2013S and CS2013I.

Applying the Graph Trans-morphism Algorithm to the Human Computer Interaction related core defined in CS2013S and CS2013I results in identical complier graphs as given in Figure 11.9 on the left. The complier comprises of two disjoint trees representing the KUs U0045, *Foundations of human-computer interaction*, and U0157, *Designing Interaction*. Outcome 2 in Section 7.3 discussed disjoint digraphs and the need to include an "entry point" in the digraphs. When considering the original digraphs that represent CS2013S and CS2013I, there is no common KA between the two graphs. In CS2013S the KA is referred to as HC and in CS2013I as HCI. (Refer back to Figure 11.5.) By including the equivalence, described by the *or-subgraph* in Figure 11.8, the resultant complier is as given in Figure 11.9 on the right.

Table 11.6 presents the cardinality of the vertex and edge sets for each of the difference quantities when the *or-subgraph* has been included. This table is an excerpt from the resultant table, Table E.8, that can be found in Section E.2 of the Appendix. The tables presented in the section are

Quantity	I	M	Set representation
$I \setminus C$	CC2001	CS2008	{T00008, T00117_1, T00271, T00297, T00298, T00299_1; T00363, T00428, T00523 }
	CS2008	CC2001	{T00008_1, T00117_3, T00211_3, T01162, T01163, T01235, T01635, T01636, T01637, T01638, T01639, T01640}
$I \setminus M$	CC2001	CS2008	{T00008, T00117_1, T00271, T00297, T00298, T00299_1; T00363, T00428, T00523 }
	CS2008	CC2001	{T00008_1, T00117_3, T00211_3, T01162, T01163, T01235, T01635, T01636, T01637, T01638, T01639, T01640}
$M \setminus C$	CC2001	CS2008	{T00008_1, T00117_3, T00211_3, T01162, T01163, T01235, T01635, T01636, T01637, T01638, T01639, T01640}
	CS2008	CC2001	{T00008, T00117_1, T00271, T00297, T00298, T00299_1; T00363, T00428, T00523 }
$C \setminus M$	CC2001	CS2008	$\emptyset$
	CS2008	CC2001	$\emptyset$

Table 11.5: Scenario 2: Vertex sets for selected quantities

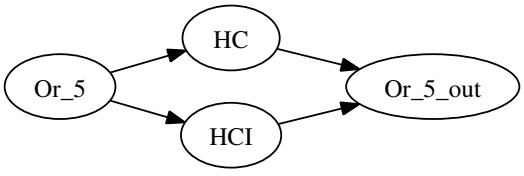
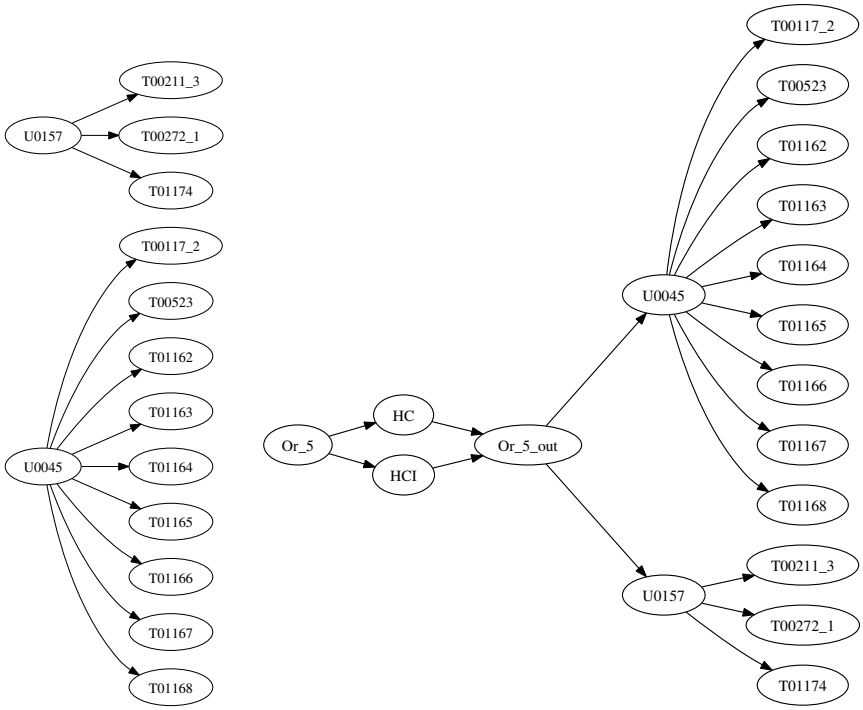


Figure 11.8: Or-subgraph for HC/HCI KA



Disjoint complier Complier after inclusion of *or-subgraph*

Figure 11.9: Scenario 2: Compliers for CS2013S and CS2013I

similar in structure to the tables in Section E.1, but relate to Scenario 2. As in the previous section, the comparison of the difference sets indicates that there are still vertices that need to be investigated for equivalences, particularly on the topic level. From the table it can be seen that $M \setminus C$ results in a set of cardinality 6 for vertices for CS2013S as ideal and CS2013I as model, and *vice versa*. These results mean that there are 12 vertices in total that are not common in the two curricula specifications for the Human Computer Interaction KA. As these are the vertices that differ, it is necessary to consider similarities between them.

Quantity	**I**	**M**	$\mid V \mid$	$\mid E' \mid$
$I \setminus C$	CS2013S CS2013I	CS2013I CS2013S	6	6
$I \setminus M$	CS2013S CS2013I	CS2013I CS2013S	6	6
$M \setminus C$	CS2013S CS2013I	CS2013I CS2013S	6	6
$C \setminus M$	CS2013S CS2013I	CS2013I CS2013S	0	0

Table 11.6: Scenario 2: Set cardinalities for selected quantities

Observations

The difference sets give a good indication of which vertices in the model were not matched. This is particularly seen in respect of the vertices of $M \setminus C$. A detailed investigation into the vertex sets reveals that there are indeed equivalences. A total of 9 equivalences on the topic level were found and each modelled with an *or-subgraph*. These *or-subgraph*s are labelled Or_N, where N is a letter from A to I. Taking topic equivalences into account and including them into the graph representation, results in a graphical representation of the curriculum volumes presented by Figure 11.10.

From Figure 11.10 it can be seen that U0045 is defined in all four curriculum volumes. Of the topic equivalences in U0045, three (Or_C, Or_D and Or_A) are between CC2001 and CS2008 only, showing a change in how topics are described already in the 2008 volume. The topics in equivalence Or_E, remained the same for CS2005 and the two CS2013 volumes, that is T01162. The change in the topic description took place between CC2001 and CS2008. Or_F represents a topic that returned in CS2013I after last being specified in CC2001. After the topic equivalences has been applied, the general structure of the KU seems more ordered when compared with Figure 11.5.

Topic equivalences Or_B, Or_G, Or_H and Or_I in Figure 11.10 reflect

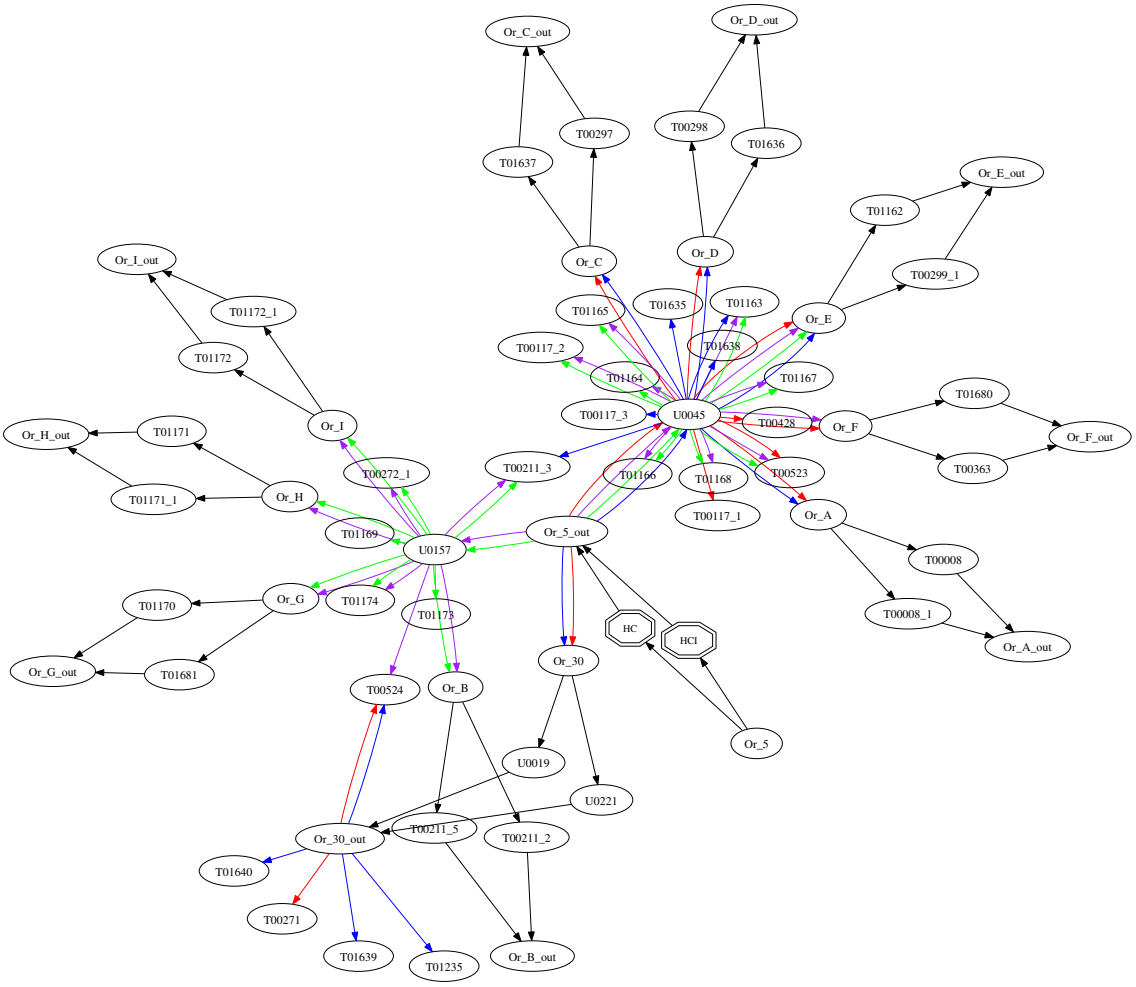


Figure 11.10: Representation of the Human Computer Interaction (HC and HCI in 2013I) with topic equivalences

subtle changes to the topics made between CS2013S and CS2013I. This accounts for the limited commonality between topics of the KU U0157 in Figure 11.5.

Topic T00524, which was shown in Figure 11.5 and aligned with either U0019 or U0221 in Figure 11.6, is not present in CS2013S.

Closer investigation of the tables in Appendix E.2 for Scenario 2 reveals that the ratios for the original digraph specifications are exactly the same for the ideal and model as they are for when the ideal and model were reversed. (Specifically, it will be seen that the first set of four values under the Original column heading in Table E.5, with CC2001 representing the ideal and CS2008 the model, are the same as the first set of four values in Table E.6.) The same is true for Tables E.7 and E.8 representing CS2013S and CS2013I.

Comparison of equivalence data in the tables in Appendix E.2 are more interesting. The comparison is presented in the radar charts to be discussed in the section regarding difference visualisation that follows.

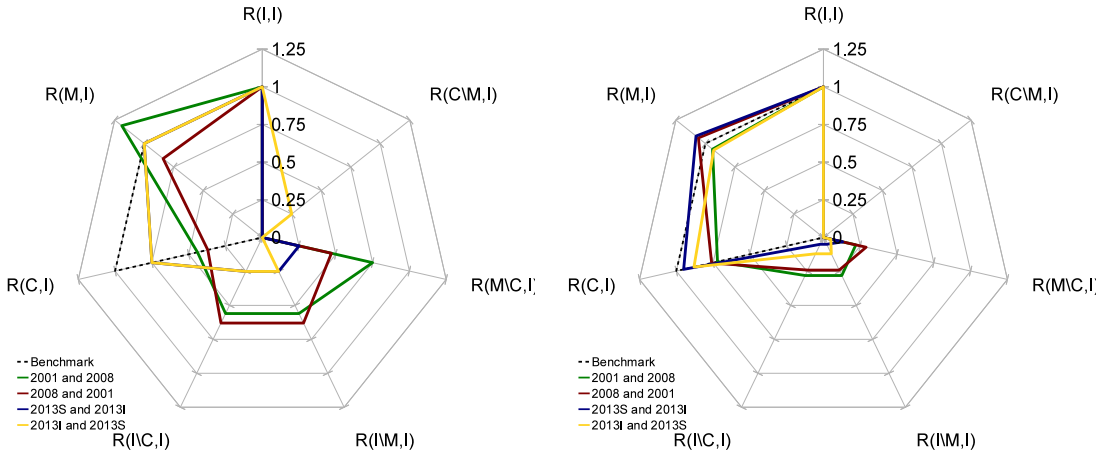
11.3.4 Difference visualisation

The ratios for vertices and edges in the tables in Appendix E.2 for Scenario 2 are exactly the same when regarding equivalences of the KAs and KUs. When adding topic equivalences to the digraphs being compared, the difference between the ratios is less than 0.05 from each other. This means that there is minimal differentiation between vertices and edges.

The radar charts in Figure 11.11 show the comparison of the curriculum volumes with equivalences. The equivalences in terms of KUs are on the left-hand-side of the figure. The right-hand-side of the figure includes the nine topic equivalences for HC/HCI across all curriculum volumes. The KA equivalence of HC in CC2001, CS2008 and CS2013S is equivalent to HCI in CS2013I is applied to both the left-hand-side and the right-hand-side. From the radar charts it can be seen that when taking the KA and KU equivalences into account, as on the left-hand-side, the differences between the curriculum volumes is greater. The addition of the topic equivalences results in a minimal difference between the curriculum volumes.

Analysis of radar chart with KU equivalence

When analysing Figure 11.11 using the equivalent KA and KUs, that is the radar chart on the left, it is apparent that CS2013S and CS2013I are similar. The only differentiation between them on the radar chart is for the ratios $R(M \setminus C, I)$ and $R(C \setminus M, I)$. Where the one ratio is 0 the other is 0.25 and *vice versa* when the model and ideal are swapped—refer to the yellow line *versus* blue line on the radar chart. This change means that there has been a 25% increase in representation of digraph CS2013I from CS2013S. Ratio



Knowledge Units (KUs) Topics

Figure 11.11: Scenario 2: Equivalences in the HC/HCI KA

$R(C, I)$ confirms that the similarity between the two volumes is 75%.

The ratio $R(M, I)$ when equivalence of KUs is taken into account highlights the fact that the representation of the HC KA has increased by 19% in going from CC2001 to CS2008. This was not immediately apparent in Figure 11.5. For both CC2001 and CS2008 as the ideal, the resulting complier—represented by the ratio $R(C, I)$—is dissimilar to the ideal. The match between the complier and the ideal is less than 50% in both instances. This ratio, along with $R(I \setminus C, I)$, $R(I \setminus M, I)$ and $R(M \setminus C, I)$, indicates that other than just an increase in representation, there was also a shift from CC2001 to CS2008 in terms of content. Referring back to Figure 11.5, this shift can be seen when considering the the conjunction between the edges with red lines and the edges with blue lines.

Analysis of radar chart with KU and topic equivalences

Analysis of the radar chart on the right of Figure 11.11 shows that when topic equivalences are included, the matching between the digraph pairs tend towards the benchmark. This was what was hoped for when viewing the visual representation of the digraphs in Figure 11.10 and comparing it to the digraphs in Figure 11.5.

The difference between CC2001 and CS2008 of approximately 25% is still visible in ratios $R(I \setminus C, I)$, $R(I \setminus M, I)$ and $R(M \setminus C, I)$. This confirms that adding the equivalences does not change the meaning as depicted by the relationships between vertices in the digraph, but rather enables improved matching. The matching between CS2013S and CS2013I has improved with the inclusion of topic equivalences.

11.3.5 Discussion

From the analysis it can be seen that there was a shift between CC2001 to CS2008 with regards to the Human Computer Interaction KA. This shift was not as pronounced as the one that took place when CS2013S was specified. During the shift to CS2013S the topics represented by the equivalent KUs U0019 (*Building a simple graphical user interface*) and U0221 (*Building GUI interfaces*) were dropped and a new KU, U0157 (*Designing Interaction*) was introduced. With one topic shared between the three KUs it cannot be concluded that the three KUs are equivalent. A further shift took place between CS2013S and CS2013I. This is most pronounced where the topics related to U0157 are concerned, as can be seen in Figure 11.5. When considering the equivalences in topics represented in Figure 11.10, it was seen that the shift was as a result of changes in topic descriptions.

This scenario has shown that finding equivalences for KAs, KUs and—more specifically—for topics, does improve matching. It is also important that, in order to remove bias, equivalences are applied uniformly to all digraphs that are to be compared. When equivalences are uniformly applied, their net effect on the ratios is limited to the ratios $R(M, I)$ and $R(C, I)$. The equivalences have no effect of the difference ratios. This is because uniform application of equivalences means that the same equivalences occur in the two digraph sets which serve as operands of the set difference operator. As a result, the resulting set will therefore not contain any equivalence information.

Scenario 2 shows that it is desirable to extend the mapping of topic equivalences to the other KAs. An interesting KA to begin with would be the KA defined as PF in CC2001 and CS2008 and defined as SDF in CS2013 Strawman and Ironman. Section 8.4.2 and Table 8.2 referred briefly to these changes.

11.4 Scenario 3: Application to a real-world curriculum

11.4.1 Scenario overview

In this scenario, the core aspects of a real-world curriculum presented in Appendix F will be compared with the ACM/IEEE curriculum volumes. The real-world curriculum is that of the BSc degree programme in Computer Science (BSc CS) presented at the University of Pretoria.

In this scenario, knowledge provided by the curriculum expert and the module expert, will be needed to make decisions regarding curriculum choices.

11.4.2 Overview of the BSc CS degree programme

The BSc CS curriculum was developed in 2007 using CC2001 as a basis. The methodology followed to develop the curriculum was to complete a spreadsheet based on what was being presented in the modules that existed at the time. An example of the spreadsheet has already been presented in Chapter 10, Figure 10.4. The information was used to determine the gap between what was being presented and what is required in a Computer Science curriculum as defined in CC2001 [Marshall, 2011]. In the absence of any software tools (other than a spreadsheet), the data collection and the gap determination process had to be done manually—all in all a rather demanding task.

The BSc CS curriculum that resulted from this process is presented in Appendix F. The curriculum follows the curriculum structure as required by the University of Pretoria. Modules are classified as fundamental, core and elective. Fundamental modules are prescribed by the University. Core modules are prescribed modules which all students following the degree programme must complete. Electives add variety to the degree and, within parameters, students may use them to customise their programmes. In the BSc CS degree programme, Computer Science students are required to pass the specified COS modules presented by the Computer Science department. Other departments in the University are responsible for the modules whose codes do not begin with COS. Mathematics presents the modules on Discrete Structures (WTW115 and WTW285), Information Systems presents the database module (INF214) and Information Science the Social and ethical issues surrounding Information Technology in INL240. The Computer Architecture module, ERA284, was presented by the Electrical, Electronic and Computer Engineering department.

The first intake of students on the curriculum was in 2009 with the first cohort completing at the end of 2011. Since the implementation of the degree programme, a few changes have been made to the programme. These changes are mainly due to institutional changes, changes to modules presented by other departments that impact the core modules of the degree programme, and changes to staff and/or textbooks used in the modules.

11.4.3 Changes require re-evaluation

With changes taking place both in the ACM/IEEE curriculum volumes as well as the BSc CS degree programme, it would be sensible to test what is being presented against what is expected. In 2011 this was done [Marshall, 2011]. From this study it was seen that the curriculum inadequately reflected the changed ACM/IEEE curriculum volumes, namely CS2008 and CS2013S.

The original spreadsheet, along with the official university module description and the study guides for the respective modules, was used as a basis

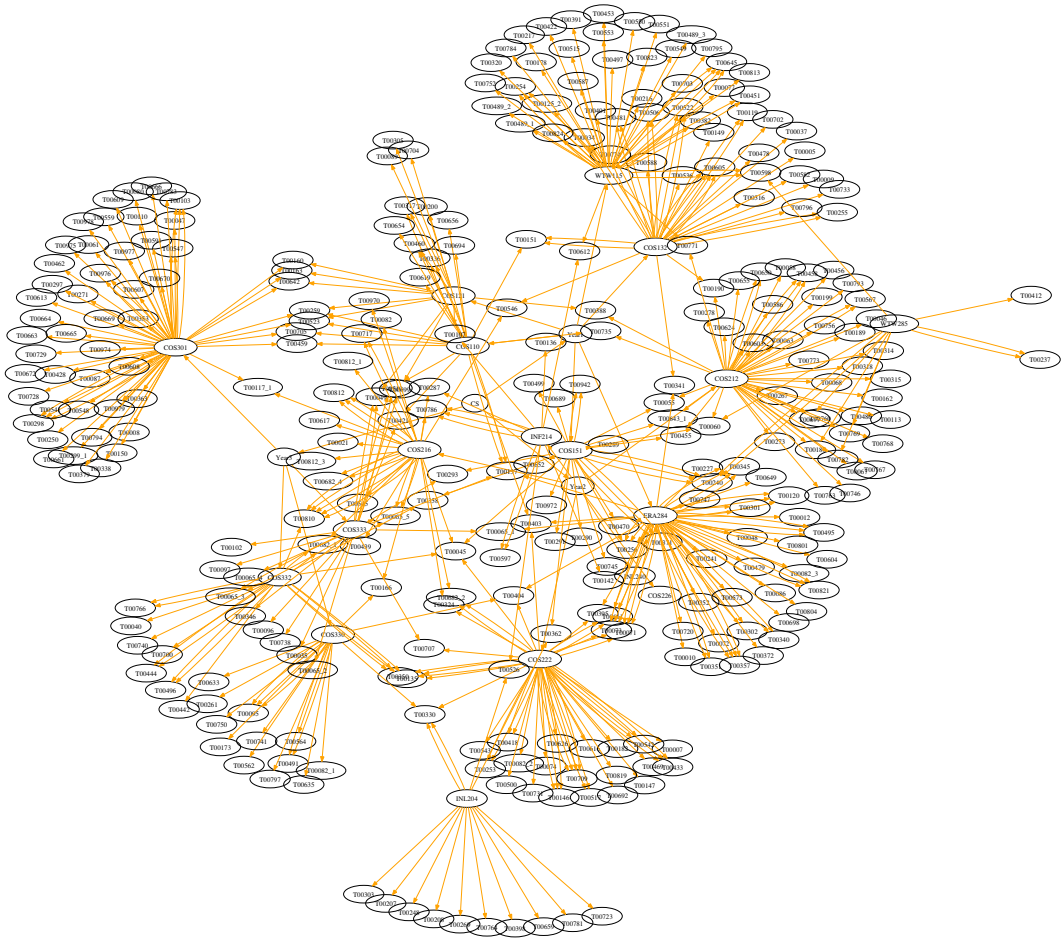


Figure 11.12: Real-world BSc CS

for re-evaluating the BSc CS curriculum. From this information, a keyword match (introduced in Section 10.4) was applied to the BSc CS specification. The CC2001 topics in the then-existing curriculum were derived from these keywords.

11.4.4 Graph visualisation

In this section the BSc CS degree programme will be compared visually with the ACM/IEEE Curriculum volumes. Figure 11.12 represents the core modules of the BSc CS degree programme. When this figure is compared with the core in CC2001 given in Table 11.1, the two figures do not appear to be structurally similar. Neither is the extent to which the BSc CS curriculum complies with CC2001 immediately apparent.

Figure 11.13 presents the complier graph of the BSc CS degree as model, shown in yellow, with the ideal of CC2001 in red. Common edges will have a

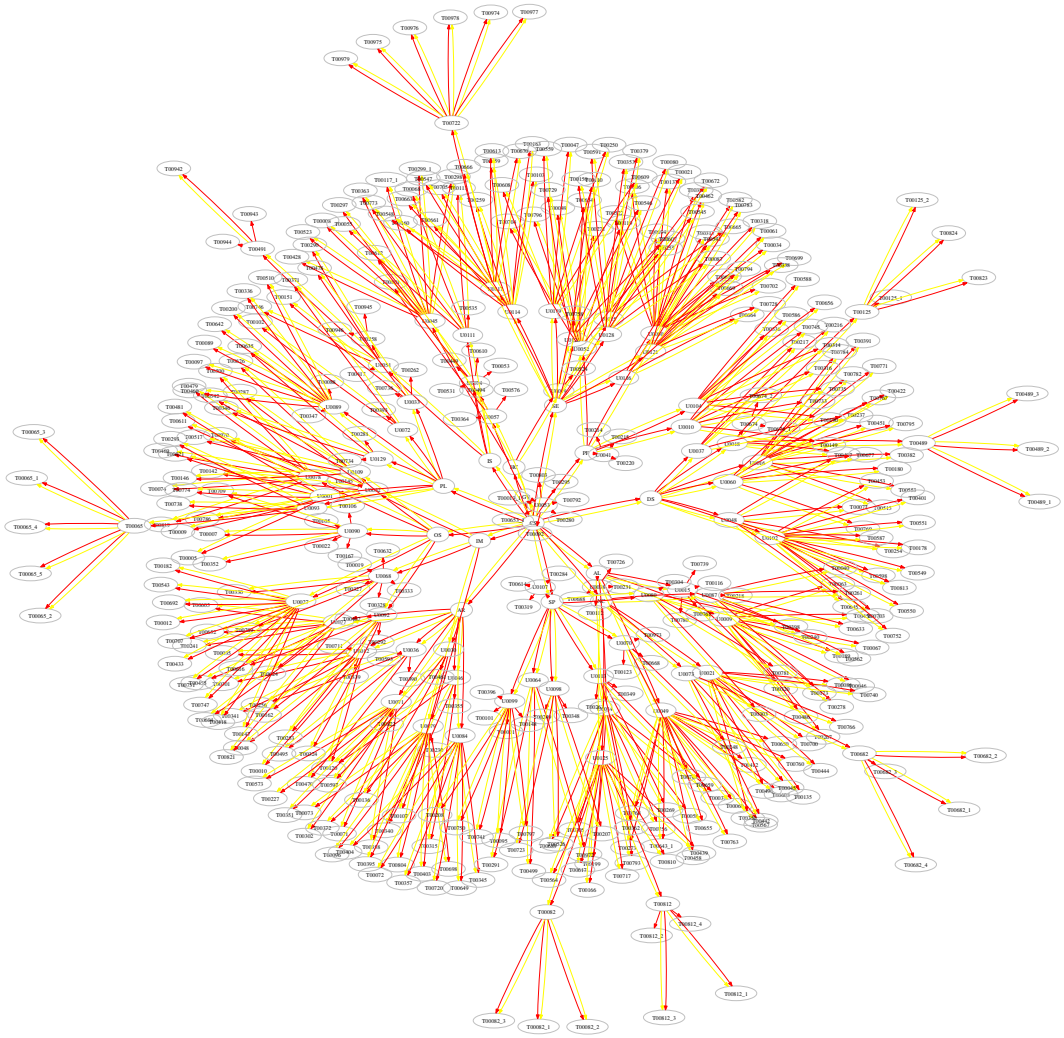


Figure 11.13: Complier of the BSc CS as model and CC2001 as ideal

double line, one in yellow representing the edge of the complier and one in red representing the edge of CC2001. From the figure one has the impression that the complier is similar to CC2001. The Difference Comparison and Difference Visualisation components presented in Sections 11.4.5 and 11.4.6 respectively will provide more detailed insight into the similarity.

The immediate question to arise is: "How well does the BSc CS curriculum compare with CS2013I?". To answer this question, the visual representations of the complier of the BSc CS curriculum as model and CS2013I as ideal are overlayed and presented in Figure 11.14. From the representation in Figure 11.14, it can be seen that the matching of the complier of the BSc CS degree programme no longer matches as well as it did with CC2001 (compare with Figure 11.13). This can be seen when inspecting the edges

that are represented in both yellow and purple. In this comparison, these edges are few. On closer investigation between CC2001 and CS2013I, this result can be attributed to the subtle changes in topic descriptions that have taken place between the two volumes.

In order to update the topics in the BSc CS degree programme from CC2001 topics descriptions to CS2013I topics descriptions, a keyword match—as described in Section 10.4, Capturing topic data—is applied. Keywords or phrases for the modules in the BSc CS degree programme are identified by using the official university module descriptions and the module study guides. The list of topics associated with CS2013I are used to represent the curriculum volume topics. For each keyword or phrase representing the module, the corresponding CS2013I topic(s) for the related KA and/or KU are identified. The identification follows a two step process. The first step makes use of the program to suggest candidate CS2013I topics that relate to the module description keywords or phrases. The second step requires the curriculum and/or module expert to accept or reject the topics that have been proposed as matching. The results of the topic matching process and inclusion in the BSc CS degree programme are given in Figure 11.15. This figure is a representation of the complier of the BSc CS degree programme with topics in terms of the topics defined in CS2013I and compared with CS2013I. The edges that are both yellow and purple in Figure 11.15, indicates that modelling the BSc Cs degree programme with topics in CS2013I when comparing with CS2013I, results in a better match between the complier and the ideal. A more in-depth analysis of the results is presented in Sections 11.4.5 and 11.4.6.

11.4.5 Difference comparison

	CC2001(I), BSc CS(M)		**CS2013I(I), BSc CS with CC2001 topics (M)**		**CS2013I(I), BSc CS with CS2013I topics(M)**	
Quantity	$\mid V \mid$	$\mid E' \mid$	$\mid V \mid$	$\mid E' \mid$	$\mid V \mid$	$\mid E' \mid$
I	476	477	732	732	732	732
M	332	397	332	397	370	470
C	338	389	157	156	461	460
$I \setminus C$	88	88	575	576	271	272
$I \setminus M$	165	477	625	732	383	732
$M \setminus C$	21	397	225	397	21	470
$C \setminus M$	77	389	50	156	112	460

Table 11.7: Scenario 3: Set cardinalities for the comparison of the BSc CS with CC2001 and CS2013I

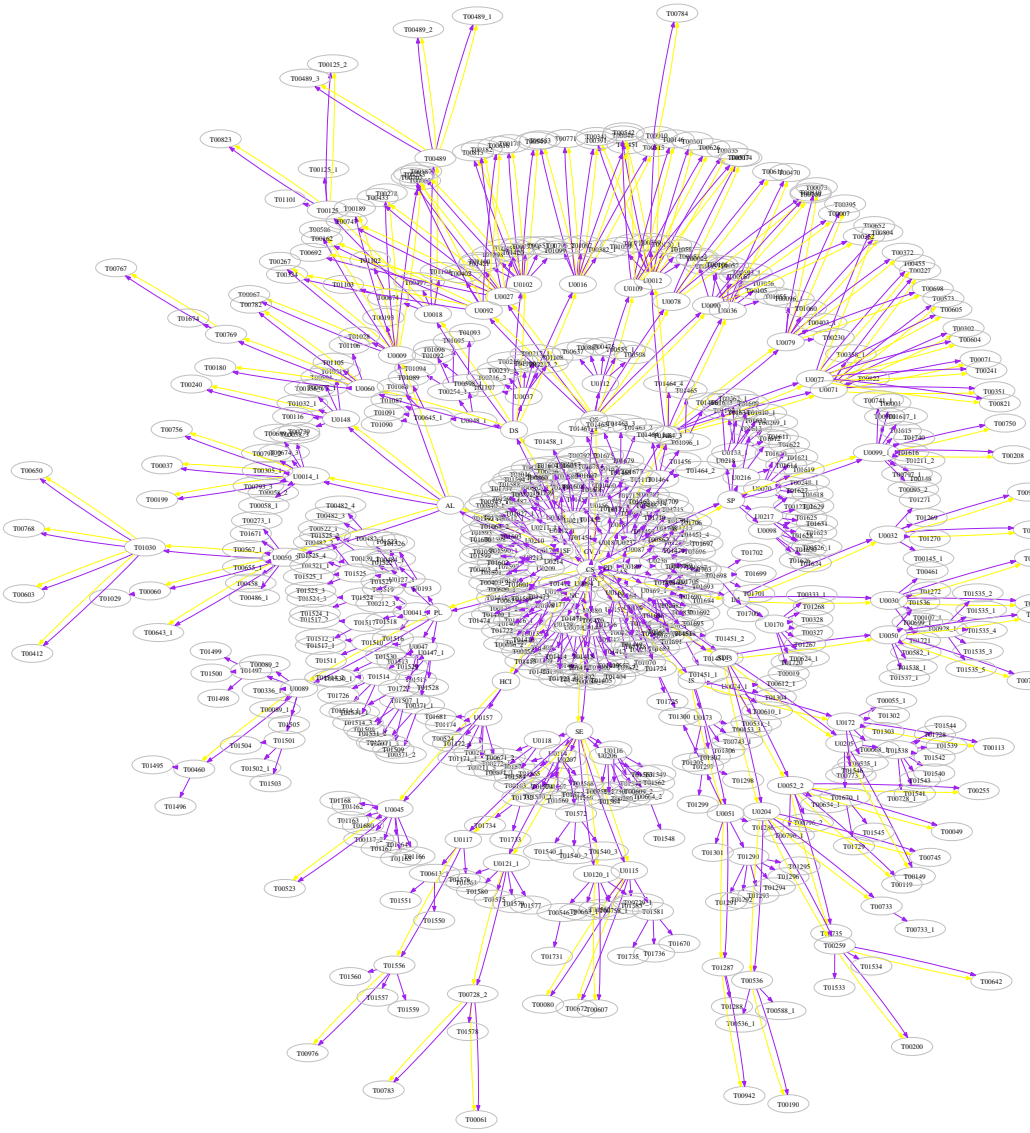


Figure 11.14: Complier of the BSc CS as model and CS2013I as ideal - BSc CS topics defined in CC2001

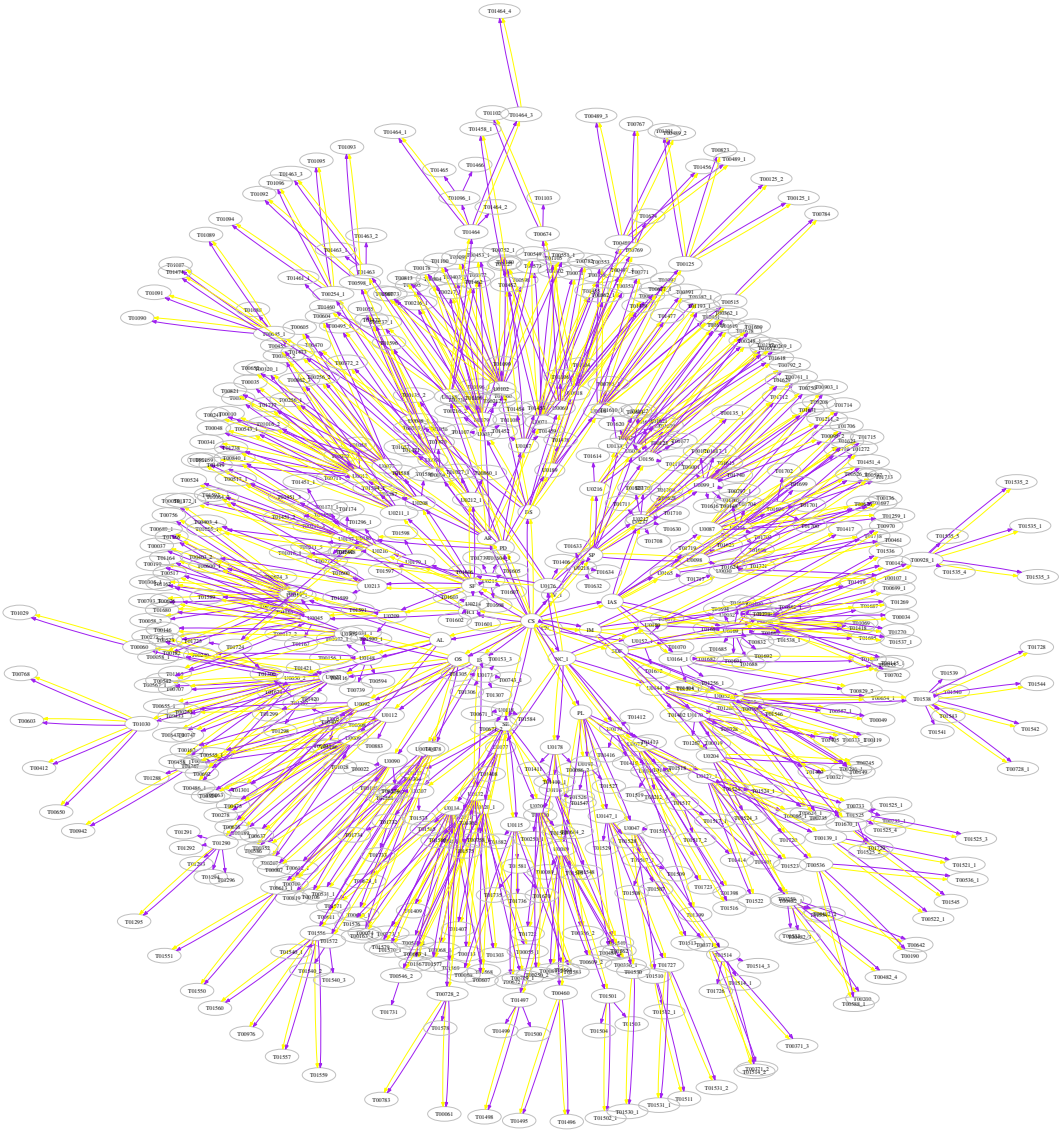


Figure 11.15: Complier of the BSc CS as model and CS2013I as ideal - BSc CS topics defined in CS2013I

Table 11.7 presents the set difference quantities where the real-world BSc CS curriculum is taken as model and where first CC2001 and then CS2013I are used as the ideal. As with the visualisation, there are two versions of the comparison with CS2013I. The first makes use of the topics as defined in CC2001. The second makes use of topics in the BSc CS as defined in CS2013I. The cardinalities of the ideal, model and complier sets are included in the first three rows of the table. The data in the table is used to compute the ratios that are visualised in radar charts to be discussed later in Section 11.4.6.

Inspection of the vertices for each of the difference sets reveals some interesting insights. These will briefly be discussed for each of the ideal and model digraph combinations presented in Table 11.7. A listing of the vertices for each of the vertex difference sets as produced by the Graph Trans-morphism Algorithm has been provided in Appendix G. A synopsis and brief discussion of these results follows.

(a) CC2001(Ideal) and BSc CS(Model)

$I \setminus C$**:** Of the 88 vertices, 4 are KUs. The difference of 84 therefore represents the topics in the ideal that are not in the complier.

$I \setminus M$**:** The vertices represent KAs, KUs and topics. It is understandable that the KAs and KUs will not be in M as M does not make use of these concepts when modelling the real-world curriculum. The total number of topics in the vertex set is 89, more than half the total number of vertices in the set (namely 165).

$M \setminus C$**:** The vertex count of 21 reflects differences with respect to structural differences in the BSc CS only. Thus, all topics in M are also in C

$C \setminus M$**:** Of the total of 77 vertices, only 5 refer to topics. The other 72 vertices highlight the difference in structure between the complier and the model. The rationale for these vertices not being in M is the same as for $I \setminus M$.

Discussion

Closer examination of the 84 topics in $I \setminus C$ shows that all of them are also in $I \setminus M$. The 5 topics in $I \setminus M$ that are not in $I \setminus C$ are: T00065, T00125, T00489, T00682 and T00722. These are the exact same topics that are in $C \setminus M$, meaning that only these 5 topics have been inferred. The 84 topics in I which are not in C are of more concern and need to be investigated by the curriculum specialist as well as the real-world curriculum module presenters to determine whether they were not overlooked when defining the model.

(b) CS2013I (Ideal) and BSc CS with CC2001 topics (Model)

$I \setminus C$**:** A total of 575 vertices in the vertex difference set are in I and not in C. Of these vertices, 6 represented KAs and 52 represented KUs. This means that there are 517 vertices representing core topics of CS2013I that are not in the complier.

$I \setminus M$**:** This set difference in terms of vertices results in a total of 625 vertices with 17 representing KAs and 85 KUs. The vertices representing topics is therefore 523.

$M \setminus C$**:** Of the 225 vertices, 204 represent topics. These 204 topics in the real-world curriculum cannot be matched with topics in CS2013I. The topics however in the real-world curriculum were based in the topics in defined in CC2001 and therefore it is evident that there has been a shift with regards to how the topics are specified between CC2001 and CS2013I. The topics in this set of vertices should be investigated to determine whether there are equivalences for them in CS2013I.

$C \setminus M$**:** There are 6 vertices representing topics. These topics are: T00125, T00728_2 T01030, T01287 and T01556. These topics have been included in C after they have been inferred from the information in M.

Discussion

The number of topic vertices for the difference quantities $I \setminus C$, $I \setminus M$ and $M \setminus C$ reflect that changes in the specification of topics has taken place between CC2001 and CS2013I. It is necessary to determine the the equivalences between the topics in the curricula volumes in order to preserve backward compatibility of the real-world curricula with the information presented in the latest volumes. By considering the topics represented in the difference set $M \setminus C$ and looking for equivalent topics in CC2001, it may be possible to improve the comparison after application of Rules 10.1 and 10.2.

(c) CS2013I (Ideal) and BSc CS with CS2013I topics (Model)

$I \setminus C$**:** A total of 271 are in the vertex set. Of these 266 represent topics and 5 KU's.

$I \setminus M$**:** Of the 383 total vertices, 17 represent KAs and 85 KUs. A total of 281 topics are in I which are not in M.

$M \setminus C$**:** The 21 vertices in this set are the structural elements of M such as the year-levels and module codes that are not in C. These differences reflect structural aspects of the BSc CS degree programme in the following way: 18 vertices represent the modules

of the degree program; and the remaining 3 vertices represent the years into which these modules are classified.

$C \setminus M$: The majority of the 112 vertices in the vertex set of the quantity $C \setminus M$ represent structural aspects of C. There are 17 KAs, which corresponds to the KA value for $I \setminus M$, and there are 80 KUs. Therefore, $112 - (17 + 80) = 15$ topics have been inferred.

Discussion

The number of topics for the difference quantities $I \setminus C$ and $I \setminus M$, namely 266 and 281 respectively, gives an indication to what extent it is necessary to keep abreast with the curriculum changes when presenting a curriculum such as a BSc CS degree programme. From the results it can be concluded that the BSc CS degree program (having been developed using CC2001 as a guide) does not adequately comply with the specification in CS2013I even when using un updated CS2013I related topic list when representing the degree programme.

Summary

From Table 11.7 it can be seen that the model increases in size when the topics from CS2013I are used. It is important to remember that a topic equivalence exercise between CC2001 and CS2013I was not done in order to select the topics. The topics from CS2013I were selected by extracting keywords from the module descriptions and study guides of the BSc CS degree programme and then looking for these keywords in the topic list for CS2013I. From the list of modules and related keyword and the corresponding list of topics and related KUs and KAs from CS2013I, relevant topics for the modules were chosen by a semi-automated process. The CC2001 topics in the BSc CS degree programme were therefore not used as a basis to select the topics for the updated BSc programme.

The above comparison did not consider topic equivalences. The notion of topic equivalences was illustrated for the Human Computer Interaction KA in Section 11.3. To determine the topic equivalences between CC2001 and CS2013I, the topics selected for the BSc CS programme using CC2001 and CS2013I topics respectively can be used as a starting point. These topics are structured in terms of modules. This clusters relevant topics together and reduces the complexity of the search space. Determining topic equivalences between the different curriculum volumes is included in the future work chapter.

From the table it can also be seen that comparing a BSc CS curriculum that was developed using CC2001 with CS2013I results in a drop in the size of the complier. The cardinality of the vertices of the complier drops from 338 when compared with CC2001 to 157 when compared with CS2013I.

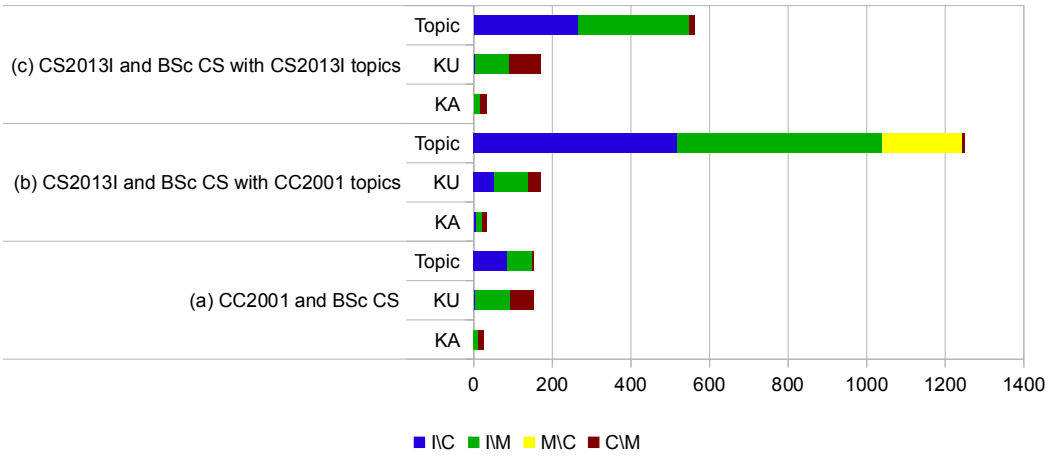


Figure 11.16: Scenario 3: Vertex set cardinalities in terms of KAs, KUs and Topics

This indicates a change either in topics descriptions between CC2001 and CS2013I, or a shift in focus between the curriculum volumes. In order to determine the cause, a curriculum expert will need to consider the difference sets given in Appendix G.

Figure 11.16 breaks the total cardinalities for the vertices presented in Table 11.7 into KA, KU and topic cardinalities. These cardinalities are presented per difference quantity. The cardinalities are presented as accumulative values on the bar graph per comparison presented by the scenario.

An interesting result, highlighted in Figure 11.16, is that for the topics represented by the quantity $M \setminus C$ when the BSc CS using topics from CC2001 was compared with CS2013I. These topics (shown in yellow and 204 in total) are topics defined in CC2001 that are no longer in CS2013I. Each of these topics needs to be investigated and equivalences in CS2013I found for them. Topics with no equivalences indicate a shift in focus from CC2001 to CS2013I. This focus shift could mean either that CS2013I has moved totally away from the topic or it could mean that the topic is no longer defined as core. Finding equivalences for the majority of these topics will reduce the differences shown by quantities $I \setminus C$ and $I \setminus M$.

The topics not in the BSc CS that are in CC2001, highlighted by the quantity $I \setminus C$ represented by (a) in Figure 11.16, should also be resolved. Not resolving the topics difference in (a) may have a knock-on effect for the comparisons if the BSc CS with CS2013I. Both for the BSc CS with CC2001 topics (topic bar for (b)) and then for the BSc CS with CS2013I topics (refer to the topic bar in the figure for (c)).

Similarly the topics should also be resolved for (c) for the same quantity. Resolution of (b)for the quantity serves no purpose due to the already

reported on disparity between the topics of CC2001 and CS2013I.

11.4.6 Difference visualisation

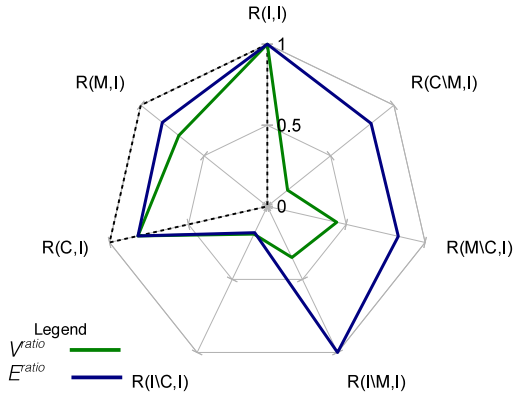
From the results presented in the previous section, Section 11.4.5, it would seem that the BSc CS curriculum that was developed using CC2001 as a basis poorly complies with CS2013I. By comparing the radar charts in Figure 11.17, this supposition is confirmed.

Each of the ratios in each of these radar charts will now be discussed. For purposes of brevity and clarity, the comparisons will be referred to as (a), (b) and (c) as indicated in Section 11.4.5. Cardinalities for the vertices and edges for each of the quantities can be found in Table 11.7.

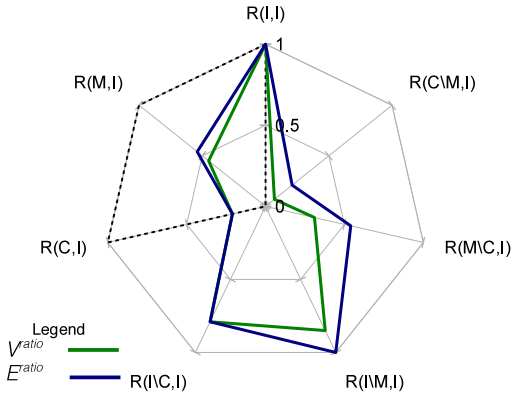
$R(M, I)$**:** This ratio represents the relative size of M in terms I. Because of the increase in the size of I in going from CC2001 to CS2013I while M does not change significantly, it is to be expected that the ratio in (a) will be greater than in (b) and (c). The values in (b) and (c) are similar, suggesting that the two ways in which topics are determined (i.e. based on CC2001 or based on CS2013I) does no influence the comparison of M against I for CS2013I.

$R(C, I)$**:** The complier provides an indication of how well M matches I in terms of content. The ratio for (a) is 0.82 indicating a relatively good match between the model and the ideal. The ratio for (b) is the worst of the three at 0.21 indicating that there is a mismatch between the information in M, particularly the topics, and the requirement as specified by I - in this case CS2013I. The value for C is 0.63 which shows that there was an improvement in matching when the topics of the BSc CS degree programme were defined in terms of topic descriptions given in CS2013I. Nevertheless, the match in (c) is not as good as it was in (a), suggesting that the curriculum of the BSc CS needs to be reconsidered and adjustments made in line with the changes specified by CS2013I. Investigation of the sets relating to $M \setminus C$ and $C \setminus M$ will give insights into what needs to be included in the revamped BSc CS curriculum and what needs to be removed.

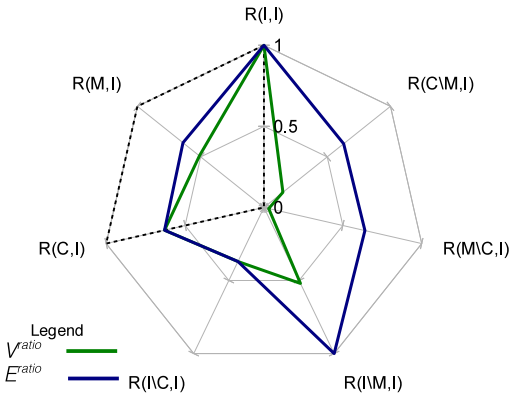
$R(I \setminus C, I)$**:** This ratio is closely related to $R(C, I)$ in that in both ratios, only I and C, (which is derived as a subgraph isomorphism of I) are being compared. There are 88 vertices and 88 edges, from Table 11.7, in (a) which need to be investigated. Of the 88 vertices, 84 (18%) represent topics and 4 represent KUs. The ratio in (b) fares badly at 0.79 for both vertices and edges. Of the 575 vertices in I that are not in C, 517 refer to topics. There is an improvement in (c) to a ratio of 0.37.



(a)
CC2001 and BSc CS with CC2001 topics



(b)
CS2013I and BSc CS with CC2001 topics



(c)
CS2013I and BSc CS with CS2013I topics

Figure 11.17: Scenario 3: Radar Charts - BSc CS (taken as M) compared to CC2001 and CS2013I Curriculum volumes (taken as I in each respective case).

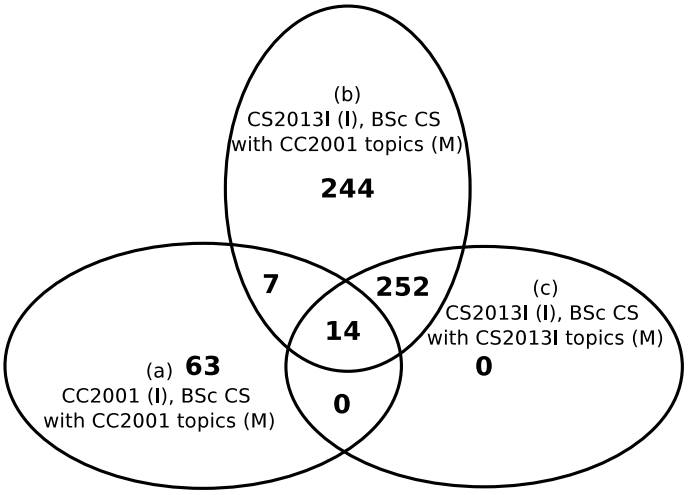


Figure 11.18: Scenario 3: Venn diagram of common topic vertices between comparisons of curricula volumes with a real-world curriculum for quantity $I \setminus C$

A comparison of the topics in (a), (b) and (c) shows that 14 topics are common across the three comparisons. These topics should be the first to be evaluated by the curriculum expert and considered for possible inclusion in the BSc CS degree programme by the relevant module expert. Figure 11.18 presents an overall view of the number of topics that are common between the different comparisons. The 7 topics that are common between only (a) and (b) also need to be considered for inclusion in the BSc CS degree programme.

Once a decision regarding the topics that are common to the comparisons has been made, equivalences between topics can be investigated. There are 63 topics in (a) which are not in (b). These are topics defined in CC2001. The topics in (b), a total of $244 + 252 = 496$, which are not in (a) are topics defined in CS2013I. The curriculum expert should consider each of the 63 topics in (a) and see whether there are not equivalences in (b) for these topics.

$R(I \setminus M, I)$**:** In general, the ratio performs better for vertices than it does for edges for all three comparisons. This can be attributed to the structural differences between I and M. A similar comparison as was done with $R(I \setminus C, I)$ can be applied to this ratio to determine where the similarities and differences lie between I and M.

An interesting comparison that can be made is between the difference sets of $I \setminus C$ and $I \setminus M$. This has already been discussed in Section 11.4.5 for (a), where it was pointed out that all 84 topics in $I \setminus M$ are also in $I \setminus C$ but that the latter contains an additional 5 topics. A similar

comparison for (b) indicates that there are 4 additional topics in $I \setminus M$ that are not in $I \setminus C$. This means that if the curriculum expert uses $I \setminus C$ when considering differences for resolution and reruns the algorithm before looking at $I \setminus M$, the differences that have already been resolved will not reflect in $I \setminus M$.

$R(M \setminus C, I)$: It is to be expected that $R(M \setminus C, I)$ for both vertices and edges is not 0. This is due to the structural components in M that are not in I. It is expected that all the structuring information, such as year level and module (this information was reflected in Figure 10.2) will be included in the calculation of the ratio.

In (b), $M \setminus C$ contains both structuring information as well as topic information that are in M and not in I. The topics in M are topics are defined in CC2001, while the topics in C are defined in CS2013I. This result is highlighted in Figure 11.16. These topics in M that are not in C can be seen as being extraneous to M in terms of C. By analysing these 204 topics in M, the curriculum expert may be able to find equivalences in CS2013I.

$R(C \setminus M, I)$: This ratio is an indication of what has been inferred to be "implicit" in M and has been included in C. The ratio for edges for (a) and (c) indicates that 82% and 63% respectively of edges have been inferred. The percentages in terms of vertices are 16% and 15% respectively for (a) and (c). The percentage represented by the ratio for vertices in (b) is 7%. Closer inspection of the vertex sets, lists of which are also in the appendix, shows that most of the information that has been inferred are the KAs and KUs. The inference of topics is limited to 5, 6 and 15 topics for (a), (b) and (c) respectively. The venn diagram in Figure 11.19 presents the number of topics and their relation to each other. From the diagram it can be seen that 2 topics are in all 3 sets for $M \setminus C$. This means that these two topics (namely T00125 and T00489) are inferred by all comparisons. This is a clear indication that these topics should be considered for inclusion in the BSc CS degree programme curriculum.

A closer investigation of the other topics by a curriculum expert may result in equivalences being identified which will in turn result in better matches for the comparisons. For example, the topic T00722 *Team management* and topic T01556 *Team participation* need to be investigated in terms of their definitions in their respective curriculum volumes. T00722 is only defined in CC2001 as a topic in the KU U0117 *Software project management*. T00722 has 6 subtopics in CC2001 and it is likely that one or more of these resulted in the topic being inferred. T01556 from CS2013I also falls under the KU U0117. It also has subtopics, four in total, and has also been inferred. It is therefore

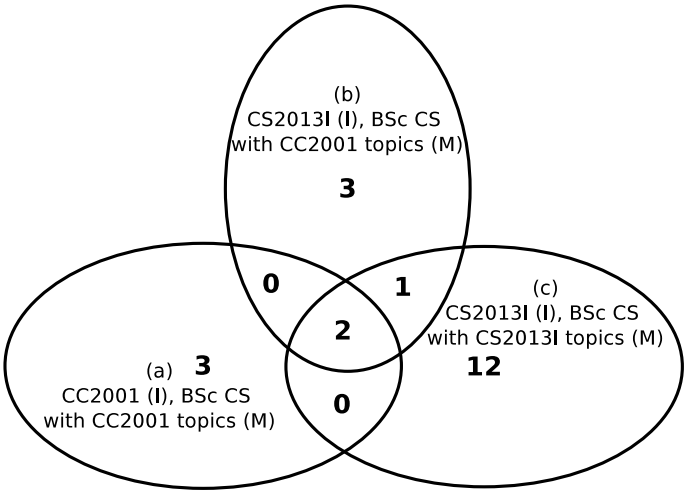


Figure 11.19: Scenario 3: Venn diagram of common topic vertices between comparisons of curricula volumes with a real-world curriculum for quantity $M \setminus C$

necessary to further investigate U0117.

Figure 11.20 presents the topics of KU U0117 for CC2001 and CS2013I. From the figure it can be seen that one topic, T00976, is shared between CC2001 and CS2013I. This topic is a subtopic in both curriculum volumes. In CC2001 it is a subtopic of T00722 and in CS2013I it is a subtopic of T01556, the two topics from the example. Topics T00722 and T01556 can be considered as equivalent. This equivalence will be referred to as **or_se_1**. Considering the descriptions for the other topics in the figure, the following additional equivalences can be determined:

or_se_2: T00613 is equivalent to T00613_1

or_se_3: T00979 is equivalent to T01599

or_se_4: T00974, T00975 and T00977 are equivalent to T01557

Applying Rules 10.1 and 10.2 to **or_se_1**, **or_se_2** and **or_se_3** will result or-subgraphs as described previously. The application of these rules to **or_se_4** will not result in the desired or-subgraph as given in Figure 11.21. The rules will need to be refined to make provision for situations such as these. This refinement will be left to form part of the future work presented in Chapter 12.

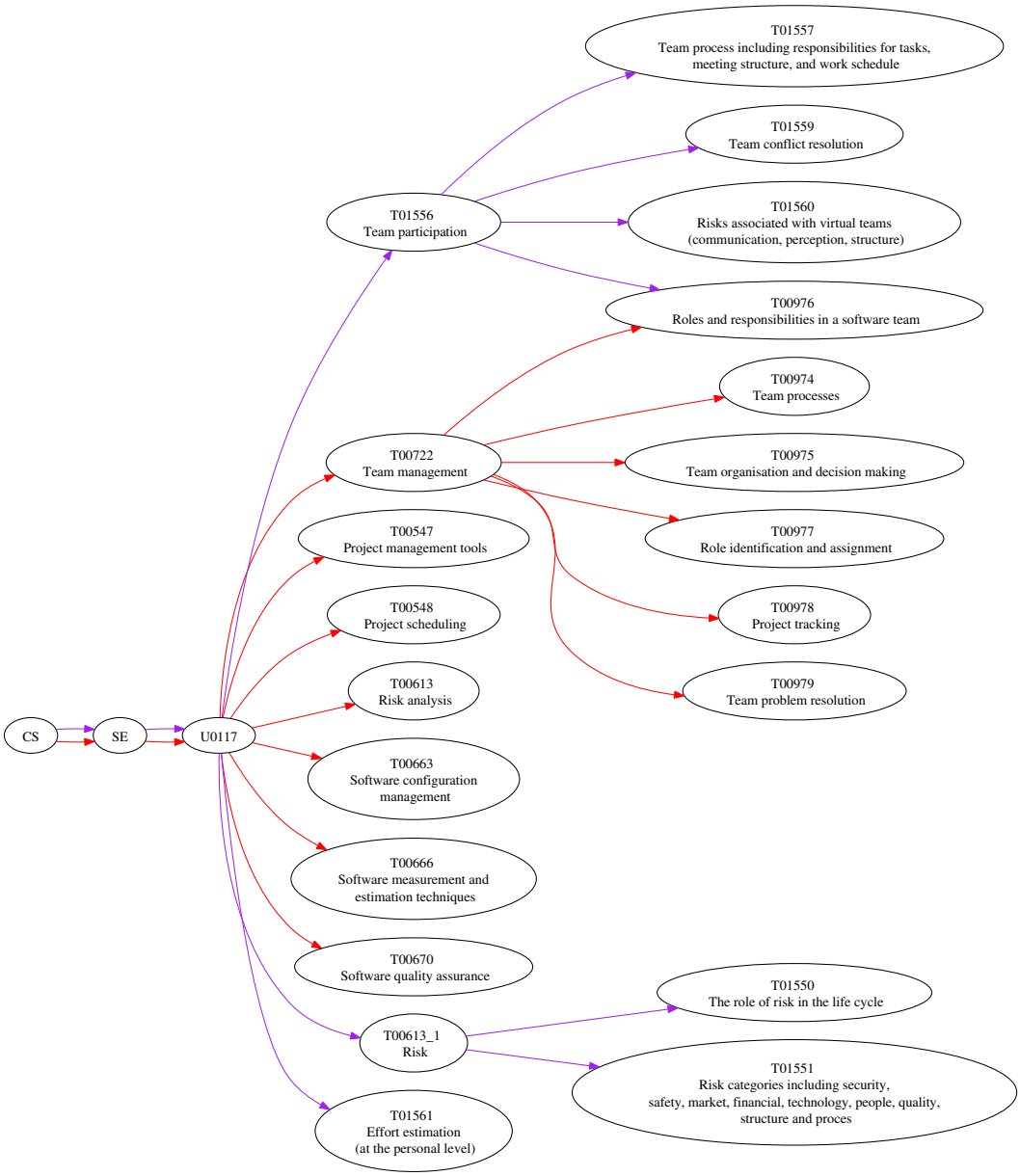


Figure 11.20: Scenario 3: U0117 topics for CC2001 and CS2013I

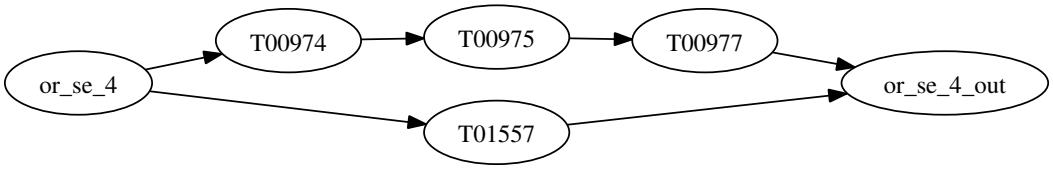


Figure 11.21: Scenario 3: or-subgraph for **or_se_4**

11.4.7 Discussion

The topics associated with the BSc CS degree programme for both (a) and (c) were determined using the same set of keywords derived from the module descriptions and study guides as documented for the BSc CS degree programme. For both, the keywords were searched for in the list of topic descriptors for the respective curriculum volumes. The resulting list was checked by a curriculum expert (the author) to make sure that the match, either exact or partial, is relevant to the topic in terms of the structure of the volume and the degree programme. Only relevant topics were kept for comparison. The results indicate that there has been a shift of focus between CC2001 and CS2013I and that a curriculum that has been developed in terms of one curriculum volume may not comply as well with an updated volume.

From a curriculum development point of view, it is desirable to keep the real-world curriculum in line with the latest curriculum volume specification in order to remain relevant with regards to the discipline. In the narrative for this scenario, some advice has been given regarding where to look for and how to approach the differences that may exist in the curriculum. Currently there is no process defined that will guide the curriculum expert into finding similarities and differences. Neither is there a processes indicating what to do when these have been found. The work presented here highlights these similarities and differences of a specific real-world curriculum in relation to CS2001 and CS2013I, using different notions of the topics in these curricula. In some places an indication of action has been given in regard to redefining topics, defining equivalences, etc. It is left to future work to determine processes that a curriculum expert needs to follow for some predetermined scenarios that may arise when comparing a real-world curriculum with a specification.

11.5 Conclusion

This chapter illustrated how the Graph Comparison Framework can be applied to real-world scenarios. Of the five areas mentioned in the introduction, two have been illustrated using only the core aspects of the curricula being modelled. The discussions around each of the areas illustrated how the results from the framework can be interpreted by a curriculum expert.

From the scenarios presented, it is clear that the modelling of equivalences is of utmost importance. In order to successfully model equivalences between the curriculum volumes a systematic approach needs to be followed. This approach could be to consider each KA independently, such as was presented in Scenario 2 in Section 11.3. Once these equivalences have been modelled, the matching between curriculum volumes and the matching of a real-world curriculum should improve specifically with regards to topics.

The results of each of the ratios should not be seen in isolation from each other. A holistic approach needs to be followed when evaluating the results and considering the difference sets. The difference sets for quantity $I \setminus C$ presents all information in I that is not in C. Invariably it is the case that if the information is not in C then it is also not in M.

Usually the difference set $M \setminus C$ for vertices includes structural information M that is not in C If this is the case then a decision can be made to ignore the quantity. If, however, the vertex sets contain topic information for the curriculum application area, then the information is significant. It means that information exists in the model that is not in the complier and the curriculum expert should attend to this.

The difference sets for $C \setminus M$ presents all information that was derived from I and included in C, but is not in M. The curriculum expert needs to determine whether the information in C but not in M is significant and whether this information needs to be explicitly included in the model.

The curriculum expert, after having reviewed the results of a number of curricula comparisons using the framework, will get a feel for which ratios are significant in which situations. It is a matter for future research to determine whether this expert evaluation knowledge can be captured and in parts automated. The framework may also be extended to include some partial analysis of the results.

Part III

Future Work and Conclusion

Chapter 12

Future Work

12.1 Introduction

The work presented in this thesis is but the beginning of a larger project with the vision of developing a tool for curriculum comparison and development in the next three or so years as one of the key projects in the CSEDAR (Computer Science Education Didactics and Applications Research) at the University of Pretoria. The work effectively forms the foundation towards the development of such a tool. Before this tool can be developed, a number of smaller projects need to be completed. Some of these projects have already been referred to in previous chapters. This chapter will provide a summary of them and include others that are mentioned for the first time.

The projects identified fall into two broad categories. The first is projects which involve further research into specific aspects of the work presented in the thesis. This includes further investigating the theories on which the framework is grounded and extending the framework itself. The second is more practical in nature. It involves the application of the framework and requires the ground work to be laid in terms of data capture with regards to curriculum volumes. It also involves the application of the framework to comprehensively assess existing curricula wherever such a need exists.

The sections that follow will involve aspects of both theory and practical projects. They have however been classified in terms of their presentation in the thesis rather than in terms of their theoretical or practical nature.

12.2 Digraph related projects

These projects focus on the implementation and transformation of digraphs. Enhancements to the Graph Trans-morphism algorithm are also discussed.

12.2.1 Set theory

In many instances when analysing the results of the Graph Trans-morphism algorithm, relationships between the set differences can be seen. An in-depth study of these relationships needs to be undertaken. This study should focus on finding trends in the results and try to determine under which circumstances these results occur.

Examples of interesting relationships in the results are presented below. These are observations and need to be investigated further to determine under which circumstances they hold. Once this has been done, their role in the analysis process of the digraphs can be determined.

Outcome 2, Section 7.5 of Chapter 7: For the specific digraphs used to illustrate Outcome 2, the following relationship between the difference sets was observed: $(I \setminus M) \setminus (I \setminus C) = C \setminus M$ and $C \cong I$. Further investigation needs to conducted to determine when $(I \setminus M) \setminus (I \setminus C) = C \setminus M$ and whether C will always be a subgraph isomorphism of I in these instances.

Scenario 2 in Chapter 11: In Table 11.5 an interesting result is seen. This result is summarised in Table 12.1 in which A and B represent the two digraphs being compared. When the algorithm is run for the first time, A is the ideal and B the model. The algorithm is run for a second time with B the ideal and A as the model. From the table it can be seen that, in each respective run, the resultant set representations for $I \setminus C$ and $I \setminus M$ are the same. The resultant set representations are reversed for $M \setminus C$. Furthermore $C \setminus M = \emptyset$ in both runs, indicating that in each respective case, M's vertices and edges are supersets of their respective counterparts in C.

Quantity	I	M	Set representation comment
$I \setminus C$	A	B	Result 1
	B	A	Result 2
$I \setminus M$	A	B	The same as Result 1
	B	A	The same as Result 2
$M \setminus C$	A	B	The same as Result 2
	B	A	The same as Result 1
$C \setminus M$	A	B	$\emptyset$
	B	A	$\emptyset$

Table 12.1: Summary of set representations of Table 11.5

This result has been used to test the integrity of digraphs A and B. If the resultant set representations do not form the pattern as seen in Table 12.1, then there is an error in the specification of either A or

B. This error in specification can be identified by comparing all the sets which should look as Result 1 does with each other. A similar comparison is also done for the Result 2 sets. Any deviations need to be checked in A or B and action needs to be taken. When the resultant sets of A and B as the ideal and model and the reverse give the pattern as shown in the table, then it can be said that there are no structural anomalies in the digraphs.

A further result provided by the sets represented by Result 1 and Result 2 in the table, is that these are most likely the vertices for which equivalences between Result 1 and Result 2 can be found.

It needs to be determined whether this phenomenon is general for all digraphs A and B. If it is, then the test for digraph integrity can be automated.

12.2.2 Algorithm improvements

Improvements to the algorithms relate to before and after execution of the algorithm and to the implementation of the algorithm itself. The discussion will begin by looking at the algorithm and how it is currently implemented.

The algorithm as presented by Algorithm 1 requires that a path be searched for in I for every *source* and *destination* vertex combination in M. Translating this implementation to the *set of triples* representation resulted in some paths being searched for multiple times due to the inclusion of labels in the path. To make the algorithm more efficient it was necessary to only consider paths between unique combinations of the *source* and *destination* vertices in M. This definition of the algorithm excludes labels. This improvement was already included in Algorithm 2. Algorithm 2 can still be made more efficient by storing a list of paths that have already been discovered and only searching for paths in I if the path being searched for in I is not already in the list or a subpath of the the path is already in the list.

Currently the algorithm does not make use of the label element of the triple when searching for paths. This means that there is no possibility of stating that only edges with specific label attributes may be included in the complier. An example would be if a distinction were to be made between core and elective topics. It would be handy if one of the elements in the label tuple would represent this information and then the selection is made at algorithm level instead of when the digraphs are created, as is currently the case. This idea can be extended to include any meta-knowledge that may be stored in the label tuple, making it possible to run the algorithm using partial definitions of the the ideal and/or model.

In some circumstances it may not be necessary to execute the algorithm in order to be able to say something about the compliance of the model

to the ideal. If, for example, as discussed in Outcome 3 in Section 5.3.2, $V(I) \cap V(M) = \emptyset$, then there are no common vertices between I and M. The resultant complier after $\mathcal{T}$ has executed will be the empty set; that is, $C = \emptyset$. On the other end of the scale as discussed in Section 6.3.1, if for example $I \setminus M = \emptyset$, then I and M are automorphic and therefore also isomorphic.

After the successful execution of algorithm $\mathcal{T}$, testing whether $I \setminus C = \emptyset$ indicates whether C is automorphic to I. Refer to Section 6.3.1 for the explanation. If this is the case, then it can be said that M is a good representation of I. The set differences $M \setminus C$ and $C \setminus M$ will give an indication of what is extraneous in M relative to I, and what is inferred to be in M from I, respectively. This information may be used to update M so that it matched I more closely.

Finally, using a technique similar to the one used by GraphViz to determine the *"root nodes"* of a digraph, it must be investigated whether it is possible for the algorithm to self determine *"entry point"* vertices. The addition of "*entry points*" has been illustrated by using the GraphViz visualisations and then manually determining the *"entry point(s)"* and linking them to the *"root nodes"* in Section 6.4.1. It was also applied in Sections 7.3 and 7.4. A slightly different application of the idea was used in Section 11.3.3.

12.2.3 Rules

From the discussion in Section 11.4.6 it is clear Rule 10.1 and Rule 10.2 are naïve in the assumption that there is a one-to-one mapping between vertices when dealing with equivalences. The rules therefore need to be updated to make provision for additional types of relationships between vertices. The first one of these relationships is given in Figure 11.21.

12.3 Knowledge representations

In this thesis a digraph representation was chosen for the ideal and model and by extension also the complier to represent the knowledge. Including a label in the tuples in which meta-knowledge is given provides a richer implementation of a digraph. It does however not change the classification of a digraph from a non-generative representation to a generative representation. To summarise from Chapter 1, a generative representation is one in which information can be deduced from the representation by looking at the relationships between concepts in the representation and reasoning about the concepts. Examples of generative knowledge representations in terms of Computer Science are concept lattices and ontologies [Andreasen et al., 2003].

A lattice in mathematics is a partially ordered set in which two elements have a unique supremum (join or least upper bound) and infimum

(meet - greatest lower bound). A concept lattice is defined as a triple. The triple comprises of two finite sets, one representing objects and the other attributes. The third element of the triple is a binary relationship between the objects and the attributes. By defining a partial order on the relations between objects and attributes, the lattice can be visualised using a Hasse diagram. A Hasse diagram is a directed acyclic graph with the constraint that each pair of vertices has a unique supremum and a unique infimum [Zhao and Halang, 2006; Berry and Sigayret, 2004].

An ontology represents knowledge in terms of concepts and the relationships between the concepts within a domain. Ontologies are used to model and reason about the domain [Wang and He, 2006]. A similarity between ontologies and concept lattices exist and it is possible to translate techniques used on one and apply it to the other [Zhao and Halang, 2006; Cho and Richards, 2007]. It has also been shown by Zhao et al. [2008] that it is possible to translate specifically from an ontology to a concept lattice. With concept lattices being modelled as Hasse diagrams [Zhao and Halang, 2006], which are a form of digraph, a mapping between ontologies and digraphs should be achievable and needs to be investigated.

Cassel et al. [2008] are in the process of building a Computing Body of Knowledge in Protégé, an ontology editor and knowledge-base framework. Protégé is an open source project and is developed by Stanford University. More information on Protégé can be found at `http://protege.stanford.edu`. It is envisaged that this Computing ontology could be used as a foundation for curriculum development in Computing. Once the Computing ontology has been completed, the extent to which the ontology could support and/or extend the groundwork laid in this thesis can be investigated.

12.4 Framework extensions

Extensions to the Graph Comparison Framework presented in Chapter 6 can be made. The framework also needs to be extended to include analysis components that can be used by the domain expert. This could either be the person with knowledge of the curriculum or the person with knowledge of the modules. How this analysis framework links to the existing framework will also need to be investigated.

12.4.1 Framework for comparison

The Graph Comparison framework consists of three parts. The Graph Trans-morphism Algorithm is the first part of the framework and provides the information to be compared. This information is manipulated by the Visualisation component and/or the Comparison component. Extension of the existing framework may include additional visualisations and comparison techniques.

Additional visualisations may make use of the existing information from the Graph Transmorphism Algorithm and/or the graph Comparison component. For example, the inclusion of a visualisation, such as the venn diagrams in Figures 11.18 and 11.19 can be included.

Additional components may be included that offer alternative ways of quantifying the differences between the ideal, model and complier. It is possible that these components make use of visualisation components as well.

The graph edit distance component which was briefly introduced in Section 6.3.2 could be further explored. For example, it would be interesting to determine if there is a correlation between the graph edit distance between two digraphs and the number of edge subdivisions that take place to complete a transformation from one graph to another. Definition 2.13 defines what is meant by edge subdivision and Rule 2.1 the transformation that needs to be applied.

12.4.2 Framework for the domain expert

It is envisage that an analysis framework will be used by the domain expert. It will take the results from the Graph Comparison Framework and present this information in a manner which is easy to understand and manipulate. It will also guide the domain expert in the analysis process.

A component that could be included in the analysis framework is one that enables the comparison of digraphs from previous comparisons to be compared with one another to illustrate progress in the curriculum development process. Such a component was already identified and used in Chapters 7 and 11. This will enable the domain expert to determine to what extent the changes made to the ideal and/or model have affected the comparison. An example of a visualisation of this information can be seen in Figures 11.18 and 11.19. The venn diagrams present the overlap between the vertices in the difference sets.

12.5 Computer Science curriculum development

The project related to Computer Science curriculum development has two sub-projects associated with it. The first relates to what has been presented in this thesis, specifically in terms of the application in Chapter 11. The way forward with regards to curriculum comparison and design is presented in Section 12.5.1. Section 12.5.2 introduces the second sub-project. This sub-project relates to the accreditation discussion presented in Section 9.3.

12.5.1 Curriculum comparison and design

Scenario 1 in Chapter 11 illustrated the value of equivalences for KAs and KUs between curriculum volumes. Scenario 2 looked at equivalences on the topic level for the Human Computer Interaction KA. The first priority within the application domain is to define topic equivalences for core topics. These equivalences need to be applied for all the KAs across the volumes CC2001, CS2008, CS2013S and CS2013I. The strategy to follow to achieve this is to consider each KA individually as was done in Scenario 2. The most interesting KA to begin with is PF in CC2001, CS2008 and SDF in CS2013 Strawman and Ironman.

After completing all the topic equivalences for core topics, the elective topics must be included. Elective topics for CC2001 and CS2008 have already been captured. They still need to be modelled and incorporated into the digraph for the particular curriculum volume. The topics for CS2013S and CS2013I must be captured and modelled. Once this has been done, topic equivalences across curriculum volumes needs to once again be determined. In some instances topics may be considered core in one curriculum volume and elective in another. This will enable the curriculum expert to understand how the volumes have changed on the micro-level, which is not discussed in the curriculum volumes. Having a complete curriculum breakdown for all published volumes, with equivalences, will enable curriculum developers to compare real-world curricula that have been developed using older volumes for partial compliance and use this as a starting point for curriculum change or improvement.

Some techniques for curriculum improvement have already been discussed when reviewing the toy application in Sections 5.4 and 6.5, the Outcomes of algorithm $\mathcal{T}$ in Chapter 7 and the application of the framework to curricula discussed in Chapter 11 for Scenarios 1, 2 in Area A and Scenario 3 in Area B. These techniques need to be formalised and included in the project relating to the analysis component mentioned in Section 12.4.2.

Real-world curricula need to be collected in order for Area 3, which was introduced in Chapter 11, to be experimented with. This experimentation may result in additional insights into curricula development and the processes involved which can be included in the analysis framework for the domain expert.

The ACM/IEEE Curriculum volumes specify learning outcomes for each KU. Up till now, these learning outcomes have not been considered at all. It would be a 'nice to have', if the relevant learning outcomes for a module could be generated. Automatic generation of learning outcomes, will make the writing of module documentation much easier. Learning outcomes will also be advantageous when applying for accreditation of the degree programme being developed.

12.5.2 Accreditation comparison

The notion of accreditation was introduced in Section 9.3. Accreditation comparison was introduced in Chapter 11 when Areas D and E were defined. It will be necessary to adapt the framework in order to be able to do accreditation comparison as well. This has a number of challenges of its own as the topics in the accreditation specification are not explicitly given. This means some additional inferencing needs to take place.

Area D compares a curriculum volume to accreditation requirements. As accreditation requirements are usually specified at a higher level of abstraction than curriculum requirements are, this area could be used by curriculum experts to develop curricula that compare favorably with both the accreditation specification as well as the specification of a curriculum volume by determining the match between the two.

The area defined as E, concerns the comparison of a real-world curriculum with accreditation criteria. It may be necessary to include the comparison of Area D to guide the comparison of topics from the real-world curriculum.

12.6 Conclusion

The ultimate goal of this research, as stated in the introduction, is to develop a tool for curriculum comparison and development. Each of the projects mentioned in this chapter, bring the study one step closer to being a more comprehensive tool for curriculum comparison and development. Some of these projects will require less work than others. In striving to develop a tool, it would be prudent to complete for each of the projects a needs analysis and assign a priority and difficulty level to each of them. These projects may then be completed by student researchers in the CSEDAR group.

Chapter 13

Conclusion

In this thesis, the extent to which the comparison of a given implementation with a specification can be automated, was considered. An algorithm was proposed that enables the comparison. Furthermore, a framework, which incorporates the algorithm forms an integral part, was proposed for quantifying the extent to which the implementation complies with the specification. The comparison and quantification was first illustrated using a toy application before it was applied to the real-world problem of curriculum comparison and development within the discipline of Computer Science.

In this chapter, the degree to which the objectives of the study have been attained will be discussed. A summary of contributions made to the Computer Science body of knowledge and to curriculum comparison and development of Computer Science will be given. The chapter will be concluded with a list of possible domains in which the framework can be applied.

13.1 Attainment of objectives

The main objective of the study originated in the domain of curriculum comparison. When re-designing a BSc CS curriculum to comply with the ACM/IEEE curriculum specification for Computer Science and the ABET accreditation specification, the overarching question that was asked was, "To what extent does the re-designed curriculum comply with the specifications?". Each time the curriculum changed, the question was asked. This resulted in a manual comparison process being launched. It was conceivable that this process could be automated. The extent to which this automation could take place became the main focus of this study.

For the purposes of the study, the problem of curriculum comparison was generalised to a problem of digraph comparison. Curricula were modelled as digraphs. The main objective of the study was generalised to determining to what extent the comparison and quantification of the differences between digraphs could be automated. To attain this objective the following sub-

objectives were considered.

Sub-objective 1: How can the digraph comparison problem be solved?

In Chapter 2, graph matching definitions were presented. These definitions, when implemented as algorithms, are considered to belong to the complexity class NP-Complete. This means an alternative algorithm to the graph matching problem was needed that would enable quantification of the comparison. Another problem that was encountered was that the digraphs being compared were not necessarily structurally similar, a requirement for the graph matching techniques. The problem was solved by developing an algorithm, the Graph Trans-Morphism Algorithm, that builds a graph using the information of the graph representing the implementation and the structure of the graph representing the specification. This new graph is then a sub-graph isomorphism of the graph which represents the specification.
The sub-objective of digraph comparison was achieved when the Graph Trans-morphism Algorithm, as described in Chapter 5, was developed.

Sub-objective 2: How can similarities/differences between digraphs be quantified?

The first challenge was to determine the similarities and differences between the digraphs before the quantification could take place. For small digraphs, 50 or so connected vertices, it is possible to represent them visually and then compare the graphs by inspection. This comparison becomes more difficult as the graphs become larger and an automatic method was needed to calculate the similarities and differences. It was not until the digraphs being compared were modelled as a *set of triples* that the problem could be solved. By modelling digraphs as sets, the set difference operator can be used to represent the differences. Once the differences can be calculated, so can the similarities.

Quantification of the similarities and differences reduced to a problem of finding the set differences between the combinations of the three graphs, the two input and the resultant graph given as output by the Graph Trans-Morphism Algorithm.
The sub-objective of the quantification of digraph similarity/difference was achieved when the Graph Comparison Framework, described in Chapter 6, was proposed. The framework makes provision for both visual and computational comparison of digraphs. A combination of visual and computational is also catered for.

Sub-objective 3: Can the generalised solution be applied to curricula?

Computer Science curricula specifications and implementations, when modelled as digraphs, result in models with 500 or more vertices and edges for core aspects of the specifications alone. Digraphs of this magnitude are near to impossible to compare using a manual process.
The sub-objective of applying the generalised digraph comparison solution to curricula has been discussed in Part II of the thesis. Chapter 11 illustrated the application of the framework. The sub-objective has been achieved. It has been shown that the generalised solution can be applied to curricula.

Attainment of the main objective — "To what extent can the comparison and quantification of the differences between digraphs be automated?".

From the discussions in Chapter 11 it can be deduced that the interpretation of the results still needs to be completed by a domain expert. The process of curriculum comparison and development can therefore be semi-automated.

13.2 Summary of contributions

The contributions that have been made can be categorised as contributions made towards the theory discussed in Part I and contributions towards application of the theory discussed in Part II. Aspects of the work has also been published in conferences and these contributions will be briefly discussed.

Theory contributions

- **An algorithm, the Graph Trans-morphism Algorithm**
  The algorithm builds a subgraph isomorphism given two digraphs, each represented as a *set of triples*. The algorithm takes these two digraphs, referred to the ideal and the model, and derives a third, referred to as the complier, which represents the information in the model in terms of the structure defined by the ideal. The algorithm was introduced and discussed in Chapter 5.

- **A partial definition for digraph isomorphism in terms of a *set of triples***
  Two definitions have been presented. The first is a definition that can be used for digraphs represented as a *set of triples* with labels. The second definition is a definition for a *set of triple* representation without labels. These definitions are presented by Definitons 6.2 and 6.4 respectively. Both the definitions take the structure of the digraph into account as well as the requirement for digraph comparison that

the information relating to vertices should also match. The digraph isomorphisms, represented by these two definitions, therefore take both the syntax of the digraphs into account as well as the implied semantics, discussed in Section 10.2.

- **A framework for digraph comparison**
  The framework for digraph comparison, referred to as the Graph Comparison Framework, was presented in Chapter 6. The framework is used to determine the similarities and differences between digraphs. These similarities and differences are also quantified for comparison purposes.

Application contributions

- **Application of the framework to curriculum design**
  In Part II of the thesis, the application of the framework was applied to the real-world application of curriculum comparison and development, specifically for curricula in the discipline of Computer Science. The application of curriculum design has always been a manual process, specifically when designing a curriculum to comply with a specification such as the ACM/IEEE Computer Science curriculum volume. It has been shown that the framework contributes to the semi-automation of this process.

- **Ability to reveal micro-level changes between Computer Science Curriculum volumes**
  The curriculum volumes focus on macro-level changes, with regards to the previous volume, when discussing the structure of the specific curriculum volume. These changes are usually on the level of KAs and KUs. Changes on topic level are rarely mentioned and it is here where the detail is needed when trying to migrate an existing curriculum from one curriculum volume specification to another. Discussion of the application scenarios in Chapter 11 highlighted the need for topic equivalences when migrating between curriculum volume specifications.

- **Possible contribution to the changes in the HC/HCI KA between CS2013S and CS2013I**
  In Marshall [2012] a model of CS2013S was compared with CC2001 and CS2008. After the conference at which the paper was presented, the details regarding the comparison was asked for by one of the delegates. It is hoped that this comparison contributed to the changes in the HC/HCI from CS2013S to CS2013I as presented in Chapter 11, Scenario 2 of the thesis.

Publication contributions

Aspects of the work presented in this thesis has previously been published at conferences. The first paper, Marshall and Kourie [2010] presented the first iteration of the Graph Trans-morphism Algorithm. In 2011, the first paper applying what was then a very basic framework to curriculum design, was presented at CSERC 2011 in Heerlen, The Netherlands. This paper discussed the application of the algorithm and a very immature set difference theory to the application of developing a curriculum in the South African context. The third paper [Marshall, 2012], also looking at the application within Computer Science curricula, compared the core aspects of the curriculum volumes CC2001, CS2008 and CS2013S.

13.3 Suggestions for applications

The research has already been applied to curriculum comparison and development. Chapter 12 has included additional areas that still need to be considered towards building a tool for curriculum comparison. This work will form the key research focus of the CSEDAR research group at the University of Pretoria in the next three or so years.

Once the framework has matured, research that applies the framework to other domains will be considered. Possible application domains, but are not limited to, may include:

- Curriculum design in the other disciplines identified by the ACM/IEEE Joint Task Force for Computing Curricula [2005].
- The possibility of applying the framework to designing curricula for MOOCs. This has been mentioned on pages 48 and 49 of the CS2013I volume. Also, mention is made that the example course given on page 356 of the CS2013I curriculum volume is presented as a MOOC on Coursera [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2013]. An initial digraph taking prerequisites for entry level courses using MOOCs has been presented in Marshall [2013]. This can be used as a basis for the application of the framework in the future.
- Modelling and comparison of genes in the discipline of Bioinformatics.
- Comparison of images that have been modelled as digraphs to determine differences and similarities between the images.
- Translating management or manufacturing processes/workflows to digraphs and comparing best practices to practices that take place on the ground.

Bibliography

Aaronson, S., Fu, C., Kuperberg, G., and Granade, C. (2013). Complexity Zoo. Accessed 23 August 2013. URL: https://complexityzoo.uwaterloo.ca/Complexity_Zoo.

ABET Criteria (2010). Criteria for Accrediting Computing Programs 2011-2012. ABET. Accessed on 5 January 2011. URL: http://www.abet.org/Linked%20Documents-UPDATE/Program%20Docs/abet-cac-criteria-2011-2012.pdf.

ABET Review (2010). ABET Self-study Questionnaire: Template for a self-study report. ABET. Accessed on 5 January 2011. URL: http://www.abet.org/Linked%20Documents-UPDATE/Program%20Docs/self-study-quest-cac.doc.

Accord, Seoul (2013). Seoul Accord. Accessed on 24 August 2013. URL: http://www.seoulaccord.com/accord/index.jsp.

Accord, Sydney (2013). Sydney Accord. Last visited 23 September 2013. URL: http://www.washingtonaccord.org/sydney/.

Accord, Washington (2013). Washington Accord. Last visited 23 September 2013. URL: http://www.washingtonaccord.org.

ACM SIGITE 2008 Task Force on IT Curriculum (2008). Information Technology 2008 Curriculum Guidelines for Undergraduate Degree Programs in Information Technology. Last visited 10 September 2013. URL: http://www.acm.org/education/curricula.html.

ACM/IEEE-CS Joint Curriculum Task Force (1991). A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report - Computing Curricula 1991. *Commun. ACM*, 34(6):68–84.

ACM/IEEE-Curriculum 2001 Task Force (2001). Computing Curricula 2001: Computer Science. Last visited 22 August 2007. URL: http://www.acm.org/education/curricula.html.

ACM/IEEE-Curriculum CS2008 Joint Task Force (2008). Computer Science Curriculum 2008: An Interim Revision of CS 2001.

Alekseev, V. (2013). Graph homeomorphism. Encyclopedia of Mathematics. Accessed on 19 February 2013.

Andreasen, T., Bulskov, H., and Knappe, R. (2003). Similarity from conceptual relations. In *NAFIPS 2003: 22nd International Conference of the North American Fuzzy Information Processing Society, 2003*, pages 179–184.

Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.-J., Kuske, S., Plump, D., Schürr, A., and Taentzer, G. (1999). Graph transformation for specification and programming. *Science of Computer Programming*, 34(1):1–54.

ARWU CS 2013 (2013). Academic Ranking of World Universities in Computer Science - 2013. Shanghai Ranking. Accessed on 25 August 2013.

AT&T Labs Research and Contributors (2013). Graphviz - Graph Visualization Software - Drawing graphs since 1988. Accessed on 30 May 2013.

Babai, L. and Luks, E. M. (1983). Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 171–183, New York, NY, USA. ACM.

Bang-Jensen, J. and Gutin, G. (2007). *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, first edition.

Barla-Szabo, G. (2002). A Taxonomy of Graph Representations. Master's thesis, University of Pretoria.

Barla-Szabo, G., Watson, B., and Kourie, D. (2004). Taxonomy of directed graph representations. *IEE Proceedings - Software*, 151(6):257–264.

Bengoetxea, E. (2002). *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France.

Bennett, C., Ryall, J., Spalteholz, L., and Gooch, A. (2007). The aesthetics of graph visualization. In *Computational Aesthetics*, pages 57–64.

Berry, A. and Sigayret, A. (2004). Representing a concept lattice by a graph. *Discrete Appl. Math.*, 144(1-2):27–42.

Black, P. E. (2004a). "complexity", in Dictionary of Algorithms and Data Structures [online]. Accessed on 22 August 2013. URL:`http://www.nist.gov/dads/HTML/complexity.html`.

Black, P. E. (2004b). "subgraph isomorphism", in Dictionary of Algorithms and Data Structures [online]. Accessed on 26 March 2013. URL: `http://www.nist.gov/dads/HTML/subgraphiso.html`.

Bondy, J. and Murty, U. (1976). *Graph Theory With Applications*. Elsevier Science Ltd.

Bovet, D. P. and Crescenzi, P. (2006). *Introduction to the theory of complexity*. Creative Commons Attribution-Noncommercial 2.5 License.

Calitz, A. P. (2010). *A Model for the Alignment of ICT Education with Business ICT Skills Requirements*. PhD thesis, Nelson Mandela Metropolitan University, Port Elizabeth, South Africa.

Career Space Consortium (2001). Curriculum Development Guidelines. New ICT curricula for the 21st century: designing tomorrow's education. Accessed on 4 January 2011.

Cassel, L. N., Davies, G., LeBlanc, R., Snyder, L., and Topi, H. (2008). Using a Computing Ontology as a Foundation for Curriculum Development. SWEL 2008: Sixth International Workshop on Ontologies and Semantic Web for E-Learning in conjunction with ITS 2008: Ninth International Conference on Intelligent Tutoring Systems.

Chinese Government's Official Web Portal (2005). China's Education System. Accessed on 20 September 2013. URL: `http://english.gov.cn/2005-08/27/content_26661.htm`.

Cho, W. C. and Richards, D. (2007). Ontology construction and concept reuse with formal concept analysis for improved web document retrieval. *Web Intelli. and Agent Sys.*, 5(1):109–126.

Council of Higher Education, South Africa (2012). VitalStats Public Higher Education 2010. Accessed on 20 September 2013. URL: `http://www.che.ac.za/sites/default/files/publications/vital_stats_public_higher_education_2010.pdf`.

Diagram of higher education qualification levels in England, Wales and Northern Ireland (2013). Diagram of higher education qualification levels in England, Wales and Northern Ireland. Accessed on 24 August 2013. URL: `http://www.ecctis.co.uk/europass/documents/ds_chart.pdf`.

Diestel, R. (2005). *Graph Theory*. Springer-Verlag, third edition.

Drozdek, A. (2008). *Data Structures and Algorithms in Java*. Cengage Learning, third edition.

EHEA Framework (2005). A Framework for Qualifications of the European Higher Education Area. EHEA.

Fiscus, J., Ajot, J., Radde, N., and Laprun, C. (2006). Multiple dimension levenshtein edit distance calculations for evaluating automatic speech

recognition systems during simultaneous speech. In *Proceedings of the International Conference on Language Resources and Evaluation LREC 2006*, LREC 2006, pages 803–808.

Ford, G. (1991). The SEI undergraduate curriculum in software engineering. In *SIGCSE '91: Proceedings of the twenty-second SIGCSE technical symposium on Computer science education*, pages 375–385, New York, NY, USA. ACM Press.

Fuller, U., Pears, A., Amillo, J., Avram, C., and Mannila, L. (2006). A computing perspective on the bologna process. *SIGCSE Bull.*, 38:115–131.

Gao, X., Xiao, B., Tao, D., and Li, X. (2010). A survey of graph edit distance. *Pattern Analysis and Applications*, 13:113–129. 10.1007/s10044-008-0141-y.

Harel, D. (1992). *Algorithmics - The Spirit of Computing*. Addison Wesley Publishing Company, second edition.

Heckel, R. (2006). Graph transformation in a nutshell. *Electron. Notes Theor. Comput. Sci.*, 148(1):187–198.

Heerand, J., Bostock, M., and Ogievetsky, V. (2010). A Tour through the Visualisation Zoo. *Queue*, 8(5):20–30.

Herman, I., Melancon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43.

Homer, S. and Selman, A. L. (2011). *Computability and Complexity Theory*. Springer, second edition.

Impagliazzo, J., Cannon, R. L., Coulter, N. S., Frailey, D. J., and Jones, L. G. (1997). Accreditation in the Computing Profession. In *SCI '97: Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*.

International Education Association of South Africa (2009). Study SA 9th Edition - South Africa's higher education landscape. International Education Association of South Africa. Accessed on 3 January 2011. URL: `http://www.ieasa.studysa.org/map/SouthAfricaHELandscape.pdf`.

International Education Exchange Center (2009). Indian Academic Degree Structure. Accessed on 20 September 2013. URL: `http://www.ieec.in/indian.php`.

Johnson, D. S. (2005). The np-completeness column. *ACM Trans. Algorithms*, 1(1):160–176.

Joint Task Force for Computing Curricula (2005). Computing Curricula 2005: The Overview Report covering undergraduate degree programs in Computer Engineering, Computer Science, Information Systems, Information Technology, Software Engineering. Last visited 22 August 2007. URL: `www.acm.org/education/curric_vols/CC2005-March06Final.pdf`.

Jooste, N., editor (2009). *Study South Africa - The Guide to South African Higher Education*. International Education Association of South Africa, 9 edition.

Koopman, T. (2009). A Generative Graph Template Toolkit (GraTe-Tk) for C++. Master's thesis, University of Pretoria.

Korf, R. E. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109.

Kriegel, K. (1986). The space complexity of the accessibility problem for undirected graphs of log n bounded genus. In Gruska, J., Rovan, B., and Wiedermann, J., editors, *Mathematical Foundations of Computer Science 1986*, volume 233 of *Lecture Notes in Computer Science*, pages 484–492. Springer Berlin Heidelberg.

Luger, G. F. (2009). *Artificial Intelligence - Structures and Strategies for Complex Problem Solving*. Addison Wesley.

Marshall, L. (2011). Developing a Computer Science Curriculum in the South African Context. In *Computer Science Education Research Conference*, CSERC 2011. ACM.

Marshall, L. (2012). A comparison of the core aspects of the ACM/IEEE Computer Science Curriculum 2013 Strawman report with the specified core of CC2001 and CS2008 Review. In *Computer Science Education Research Conference*, CSERC 2012. ACM.

Marshall, L. (2013). Leveraging online courses to increase student success in a Computer Science degree. In *Computer Science Education Research Conference*, CSERC 2013. ACM.

Marshall, L. and Kourie, D. (2010). Deriving a digraph isomorphism for digraph compliance measurement. In *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '10, pages 160–169, New York, NY, USA. ACM.

Monroe, H. (2012). Complexity Garden. Accessed on 23 August 2013. URL: `https://complexityzoo.uwaterloo.ca/Complexity_Garden#graph_isomorphism`.

Myers, R., Wilson, R. C., and Hancock, E. R. (2000). Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:628–635.

NQF Bands (2010). Level Descriptors. SAQA. Accessed on 3 January 2011. URL: `http://www.saqa.org.za/show.asp?include=focus/ld.htm`.

OED Online (2013). framework, n. Accessed July 30, 2013.

Office of Qualifications and Examinations Regulations (2012). Comparing qualifications levels. Office of Qualifications and Examinations Regulations. Accessed on 24 August 2013. URL: `http://ofqual.gov.uk/help-and-advice/comparing-qualifications/`.

Pandor, G. N. M. (2007). The Higher Education Qualifications Framework. Government Gazette, South Africa.

Preiss, B. R. (1998). *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*. John Wiley & Sons.

Qualifications Assurance Authority (2001). The framework for qualifications of higher education institutions in Scotland. Accessed on 24 August 2013. URL: `http://www.qaa.ac.uk/Publications/InformationandGuidance/Documents/FHEQscotland.pdf`.

Qualifications Assurance Authority (2008). The framework for higher education qualifications in England, Wales and Northern Ireland. Accessed on 24 August 2013. URL: `http://www.qaa.ac.uk/Publications/InformationandGuidance/Documents/FHEQ08.pdf`.

Rayward-Smith, V. J. (1986). *A First Course in Computability*. Blackwell Scientific Publications.

Sahami, M., Roach, S., Cuadros-Vargas, E., and Reed, D. (2012). Computer science curriculum 2013: reviewing the strawman report from the ACM/IEEE-CS task force. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 3–4, New York, NY, USA. ACM.

Southern African Customs Union (2011). Southern African Customs Union. SACU. Accessed on 3 January 2011. URL: `http://www.sacu.int/`.

Sutcliffe, P. J. (2009). *Moments over the Solution Space of the Travelling Salesman Problem*. PhD thesis, Faculty of Engineering and Information Technology University of Technology, Sydney.

The C++ Resources Network (2013). cplusplus.com. Accessed on 12 March 2013. URL: `http://www.cplusplus.com`.

The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society (2012). Computer Science Curricula 2013 Strawman Draft.

The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society (2013). Computer Science Curricula 2013 Ironman Draft V1.0.

United Nations (2010). Composition of macro geographical (continental) regions, geographical sub-regions, and selected economic and other groupings. United Nations Statistics Division. Accessed on 3 January 2011.

Wang, J. and He, K. (2006). Towards representing fca-based ontologies in semantic web rule language. In *CIT '06: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, page 41, Washington, DC, USA. IEEE Computer Society.

Wilson, R. C. and Hancock, E. R. (2004). Levenshtein distance for graph spectral features. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 2 - Volume 02*, ICPR '04, pages 489–492, Washington, DC, USA. IEEE Computer Society.

World Bank (2010). Country and Lending Groups. The World Bank. Accessed on 3 January 2011. URL: `http://data.worldbank.org/about/country-classifications/country-and-lending-groups`.

Zaslavskiy, M. (2010). *Graph matching and its application in computer vision and bioinformatics*. PhD thesis, ParisTech - Institut des Sciences et Technologies (Paris Institute of Technology).

Zhao, Y. and Halang, W. (2006). Rough concept lattice based ontology similarity measure. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 15, New York, NY, USA. ACM.

Zhao, Y., Halang, W., and Wang, X. (2008). A rough similarity measure for ontology mapping. In *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 136–141, Washington, DC, USA. IEEE Computer Society.

Part IV

Appendices

Appendix A

Algorithm Execution

A.1 Algorithm trace

A trace of the execution of Algorithm 2 for the toy application presented in Section 5.4.

Preconditions:

$I = \{(a, b, (e1)), (a, c, (e2)), (a, f, (e3)), (b, c, (e4)), (c, d, (e5)), (d, e, (e6)),$
$(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (h, e, (e11)), (i, d, (e12))\}$

$M = \{(a, b, (e1)), (a, d, (e2)), (d, f, (e3)), (b, f, (e4)), (b, j, (e5))\}$

$C_{result} = \textit{undefined}$

Call $C_{result} = \mathcal{T}(I, M)$

Initial state after the execution of lines 1 to 3

$I = \{(a, b, (e1)), (a, c, (e2)), (a, f, (e3)), (b, c, (e4)), (c, d, (e5)), (d, e, (e6)),$
$(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (h, e, (e11)), (i, d, (e12))\}$

$M = \{(a, b, (e1)), (a, d, (e2)), (d, f, (e3)), (b, f, (e4)), (b, j, (e5))\}$

$P_{set} = \emptyset$

$sourceSet = \{a, b, d\}$

$destinationSet = \{b, d, f, j\}$

State after iteration [1]: $i = 1$ **and** $j = 1$

$source = a$

$destination = b$

$\textbf{DFSID}(source, destination, I) = \{\{(a, b, (e1))\}\}$

$P_{set} = \{\{(a, b, (e1))\}\}$

State after iteration [2]: $i = 1$ **and** $j = 2$

$source = a$

$destination = d$

$\textbf{DFSID}(source, destination, I) = \{\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}\}$

$P_{set} = \{\{(a, b, (e1))\}\} \cup$
$\{\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}\}$

State after iteration [3]: $i = 1$ **and** $j = 3$

$source = a$

$destination = f$

$\textbf{DFSID}(source, destination, I) = \{\{(a, b, (e1)), (b, f, (e7))\}, \{(a, f, (e3))\}\}$

$P_{set} = \{\{(a, b, (e1))\},$
$\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}\} \cup$
$\{\{(a, b, (e1)), (b, f, (e7))\},$
$\{(a, f, (e3))\}\}$

State after iteration [4]: $i = 1$ **and** $j = 4$

$source = a$

$destination = j$

$\textbf{DFSID}(source, destination, I) = \emptyset$

$P_{set} = \{\{(a, b, (e1))\},$
$\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$

$\{(a,b,(e1)),(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\},$
$\{(a,f,(e3)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\},$
$\{(a,b,(e1)),(b,f,(e7))\},$
$\{(a,f,(e3))\}\} \cup \emptyset$

State after iteration [5]: $i = 2$ **and** $j = 1$

$source = b$

$destination = b$

Line 8 of Algorithm 2 ensures that loops are not searched for and therefore the DFSID call is not made. No change to P_{set} is made.

State after iteration [6]: $i = 2$ **and** $j = 2$

$source = b$

$destination = d$

DFSID$(source, destination, I) = \{\{(b,c,(e4)),(c,d,(e5))\},$
$\{(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}\}$

$P_{set} = \{\{(a,b,(e1))\},$
$\{(a,c,(e2)),(c,d,(e5))\},$
$\{(a,b,(e1)),(b,c,(e4)),(c,d,(e5))\},$
$\{(a,b,(e1)),(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\},$
$\{(a,f,(e3)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\},$
$\{(a,b,(e1)),(b,f,(e7))\},$
$\{(a,f,(e3))\}\} \cup$
$\{\{(b,c,(e4)),(c,d,(e5))\},$
$\{(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}\}$

State after iteration [7]: $i = 2$ **and** $j = 3$

$source = b$

$destination = f$

DFSID$(source, destination, I) = \{\{b,f,(e7))\}\}$

$P_{set} = \{\{(a,b,(e1))\},$
$\{(a,c,(e2)),(c,d,(e5))\},$
$\{(a,b,(e1)),(b,c,(e4)),(c,d,(e5))\},$
$\{(a,b,(e1)),(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\},$
$\{(a,f,(e3)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\},$
$\{(a,b,(e1)),(b,f,(e7))\},$

$\{(a, f, (e3))\}$
$\{(b, c, (e4)), (c, d, (e5))\},$
$\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}\} \cup$
$\{\{b, f, (e7))\}\}$

State after iteration [8]: $i = 2$ **and** $j = 4$

$source = b$

$destination = j$

DFSID$(source, destination, I) = \emptyset$

$P_{set} = \{\{(a, b, (e1))\},$
$\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, b, (e1)), (b, f, (e7))\},$
$\{(a, f, (e3))\}$
$\{(b, c, (e4)), (c, d, (e5))\},$
$\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}$
$\{\{b, f, (e7))\}\} \cup \emptyset$

State after iteration [9]: $i = 3$ **and** $j = 1$

$source = d$

$destination = b$

DFSID$(source, destination, I) = \emptyset$

$P_{set} = \{\{(a, b, (e1))\},$
$\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, b, (e1)), (b, f, (e7))\},$
$\{(a, f, (e3))\}$
$\{(b, c, (e4)), (c, d, (e5))\},$
$\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}$
$\{\{b, f, (e7))\}\} \cup \emptyset$

State after iteration [10]: $i = 3$ **and** $j = 2$

$source = d$

$destination = d$

Line 8 of Algorithm 2 ensures that loops are not searched for and therefore the DFSID call is not made. No change to P_{set} is made.

State after iteration [11]: $i = 3$ **and** $j = 3$

$source = d$

$destination = f$

DFSID$(source, destination, I) = \emptyset$

$P_{set} = \{\{(a, b, (e1))\},$
$\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, b, (e1)), (b, f, (e7))\},$
$\{(a, f, (e3))\}$
$\{(b, c, (e4)), (c, d, (e5))\},$
$\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}$
$\{\{b, f, (e7))\}\} \cup \emptyset$

State after iteration [12]: $i = 3$ **and** $j = 4$

$source = d$

$destination = j$

DFSID$(source, destination, I) = \emptyset$

$P_{set} = \{\{(a, b, (e1))\},$
$\{(a, c, (e2)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\},$
$\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\},$
$\{(a, b, (e1)), (b, f, (e7))\},$
$\{(a, f, (e3))\}$
$\{(b, c, (e4)), (c, d, (e5))\},$
$\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}$
$\{\{b, f, (e7))\}\} \cup \emptyset$

Transformation of P_{set} and assignment in Line 13 to C

$$
\begin{aligned}
C &= \mathscr{T}(P_{set}) \\
&= \mathscr{T}(\{\{(a,b,(e1))\}, \\
&\quad \{(a,c,(e2)),(c,d,(e5))\}, \\
&\quad \{(a,b,(e1)),(b,c,(e4)),(c,d,(e5))\}, \\
&\quad \{(a,b,(e1)),(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}, \\
&\quad \{(a,f,(e3)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}, \\
&\quad \{(a,b,(e1)),(b,f,(e7))\}, \\
&\quad \{(a,f,(e3))\} \\
&\quad \{(b,c,(e4)),(c,d,(e5))\}, \\
&\quad \{(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\} \\
&\quad \{\{b,f,(e7))\}\}) \\
&= \{(a,b,(e1)),(a,c,(e2)),(a,f,(e3)),(b,c,(e4)),(b,f,(e7)), \\
&\quad (c,d,(e5)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}
\end{aligned}
$$

Postconditions:

$$I = \{(a,b,(e1)),(a,c,(e2)),(a,f,(e3)),(b,c,(e4)),(c,d,(e5)),(d,e,(e6)), (b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(h,e,(e11)),(i,d,(e12))\}$$

$$M = \{(a,b,(e1)),(a,d,(e2)),(d,f,(e3)),(b,f,(e4)),(b,j,(e5))\}$$

$$C_{result} = \{(a,b,(e1)),(a,c,(e2)),(a,f,(e3)),(b,c,(e4)),(b,f,(e7)), (c,d,(e5)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}$$

A.2 Results tranformation

Application of Rule 5.7 to the results obtained by inspection given in Section 5.4.1 results in the transformation of the paths to sets of triples as given below:

(a,b): $[a\ (e1)\ b] \longrightarrow \{(a,b,(e1))\}$

(a,d): $[a\ (e2)\ c\ (e5)\ d] \longrightarrow \{(a,c,(e2)),(c,d,(e5))\}$

$[a\ (e1)\ b\ (e4)\ c\ (e5)\ d] \longrightarrow \{(a,b,(e1)),(b,c,(e4)),(c,d,(e5))\}$

$[a\ (e1)\ b\ (e7)\ f\ (e8)\ g\ (e9)\ h\ (e10)\ i\ (e12)\ d] \longrightarrow$
$\{(a,b,(e1)),(b,f,(e7)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}$

$[a\ (e3)\ f\ (e8)\ g\ (e9)\ h\ (e10)\ i\ (e12)\ d] \longrightarrow$
$\{(a,f,(e3)),(f,g,(e8)),(g,h,(e9)),(h,i,(e10)),(i,d,(e12))\}$

(a, f): $[a\ (e1)\ b\ (e7)\ f] \longrightarrow \{(a, b, (e1)), (b, f, (e7))\}$

(b, d): $[b\ (e4)\ c\ (e5)\ d] \longrightarrow \{(b, c, (e4)), (c, d, (e5))\}$

$[b\ (e7)\ f\ (e8)\ g\ (e9)\ h\ (e10)\ i\ (e12)\ d] \longrightarrow$
$\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}$

(b, f): $[b\ (e7)\ f] \longrightarrow \{(b, f, (e7))\}$

The union of the resultant sets of triples results is a complier given by:

$$\begin{aligned}
&\{(a, b, (e1))\} \cup \\
&\{(a, c, (e2)), (c, d, (e5))\} \cup \\
&\{(a, b, (e1)), (b, c, (e4)), (c, d, (e5))\} \cup \\
&\{(a, b, (e1)), (b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\} \cup \\
&\{(a, f, (e3)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\} \cup \\
&\{(a, b, (e1)), (b, f, (e7))\} \cup \\
&\{(b, c, (e4)), (c, d, (e5))\} \cup \\
&\{(b, f, (e7)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\} \cup \\
&\{(b, f, (e7))\} = \\
&\{(a, b, (e1)), (a, c, (e2)), (a, f, (e3)), (b, c, (e4)), (b, f, (e7)), \\
&(c, d, (e5)), (f, g, (e8)), (g, h, (e9)), (h, i, (e10)), (i, d, (e12))\}
\end{aligned}$$

Appendix B

Algorithm Output

The output from the program which implements the Graph Trans-morphism algorithm, $\mathcal{T}$, is given in this appendix for a selection of the possible outcomes of $\mathcal{T}$. $\mathcal{T}$ was introduced in Chapter 5. The possible outcomes of the algorithm were first mentioned in Section 5.3.2 of Chapter 5. A detailed discussion of these outcomes is presented in Chapter 7. The outcomes for which output is given are 2, 4 and 5 in Sections B.1, B.2 and B.3 respectively.

The output of the algorithm lists all the edges in the digraph for the ideal, model and complier. Note, all digraphs are represented using as a of the form (*source*,*destination*). Refer to Section 6.3.1 for a discussion of representing the edges as tuples when used in the Graph Comparison Framework.

B.1 Output - Outcome 2, Section 7.3

```
Printing out the Ideal, Model and Complier
Ideal
Printing set of tuples:  (a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5)
(b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8)
(b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d3) (c3,d4) (c3,d5)
(c4,d6) (c5,d5) (c6,d7) (c6,d8) (c6,d9) (c7,d11) (c7,d7) (c8,d10) (c8,d13)
(d10,e3) (d11,e12) (d13,f3) (d13,f6) (d2,e1) (d2,e2) (d3,e3) (d3,e4)
(d5,e3) (d6,e3) (d6,e7) (d6,e8) (d7,e5) (d8,f1) (d8,f2) (d9,e10) (d9,e11)
(d9,e9) (e1,f1) (e12,f3) (e12,f4) (e12,f5) (e8,f2)
Model
Printing set of tuples:  (c3,d3) (c3,d4) (d1,f2) (d11,f1) (d11,f3) (d11,f4)
(d2,f1) (d9,e10) (d9,e9) (e8,f2) (x,y1) (x,y2) (x,y3) (y1,z1) (y1,z2) (y2,z3)
(y2,z4) (y3,z5) (z1,c3) (z2,d1) (z2,d2) (z3,e3) (z3,e8) (z4,d9) (z5,d11)
Complier
Printing set of tuples:  (c3,d2) (c3,d3) (c3,d4) (c3,d5) (d11,e12) (d2,e1)
(d3,e3) (d5,e3) (d9,e10) (d9,e9) (e1,f1) (e12,f3) (e12,f4) (e8,f2)
```

B.2 Output - Outcome 4, Section 7.5

```
Ideal
Printing set of tuples:  (a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5)
(b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8)
(b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d3) (c3,d4) (c3,d5)
(c4,d6) (c5,d5) (c6,d7) (c6,d8) (c6,d9) (c7,d11) (c7,d7) (c8,d10) (c8,d13)
(d10,e3) (d11,e12) (d13,f3) (d13,f6) (d2,e1) (d2,e2) (d3,e3) (d3,e4)
(d5,e3) (d6,e3) (d6,e7) (d6,e8) (d7,e5) (d8,f1) (d8,f2) (d9,e10) (d9,e11)
(d9,e9) (e1,f1) (e12,f3) (e12,f4) (e12,f5) (e8,f2)
Model
Printing set of tuples:  (a1,d1) (a1,d4) (a1,e2) (a1,e3) (a1,e4) (a1,e5)
(a1,e7) (a1,f1) (a2,e10) (a2,e11) (a2,e9) (a2,f2) (a2,f3) (a2,f4) (a2,f5)
(a2,f6)
Complier
Printing set of tuples:  (a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5)
(b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8)
(b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d3) (c3,d4) (c3,d5)
(c4,d6) (c5,d5) (c6,d7) (c6,d8) (c6,d9) (c7,d11) (c7,d7) (c8,d10) (c8,d13)
(d10,e3) (d11,e12) (d13,f3) (d13,f6) (d2,e1) (d2,e2) (d3,e3) (d3,e4)
(d5,e3) (d6,e3) (d6,e7) (d6,e8) (d7,e5) (d8,f1) (d8,f2) (d9,e10) (d9,e11)
(d9,e9) (e1,f1) (e12,f3) (e12,f4) (e12,f5) (e8,f2)
```

B.3 Output - Outcome 5, Section 7.6

```
Printing out the Ideal, Model and Complier
Ideal
Printing set of tuples:  (a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5)
(b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8)
(b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d3) (c3,d4) (c3,d5)
(c4,d6) (c5,d5) (c6,d7) (c6,d8) (c6,d9) (c7,d11) (c7,d7) (c8,d10) (c8,d13)
(d10,e3) (d11,e12) (d13,f3) (d13,f6) (d2,e1) (d2,e2) (d3,e3) (d3,e4)
(d5,e3) (d6,e3) (d6,e7) (d6,e8) (d7,e5) (d8,f1) (d8,f2) (d9,e10) (d9,e11)
(d9,e9) (e1,f1) (e12,f3) (e12,f4) (e12,f5) (e8,f2) (g,a1) (g,a2)
Model
Printing set of tuples:  (c3,d3) (c3,d4) (d1,f2) (d11,f1) (d11,f3)
(d11,f4) (d2,f1) (d9,e10) (d9,e9) (e8,f2) (g,x) (x,y1) (x,y2) (x,y3)
(y1,z1) (y1,z2) (y2,z3) (y2,z4) (y3,z5) (z1,c3) (z2,d1) (z2,d2) (z3,e3)
(z3,e8) (z4,d9) (z5,d11)
Complier
Printing set of tuples:  (a1,b1) (a1,b2) (a1,b3) (a1,b4) (a2,b3) (a2,b5)
(b1,c1) (b1,c2) (b2,c1) (b2,c3) (b2,c5) (b3,c4) (b3,c6) (b3,c7) (b3,c8)
(b4,c4) (b5,c8) (c1,d1) (c1,d2) (c2,d6) (c3,d2) (c3,d3) (c3,d4) (c3,d5)
(c4,d6) (c5,d5) (c6,d8) (c6,d9) (c7,d11) (c8,d10) (c8,d13)
(d10,e3) (d11,e12) (d13,f3) (d2,e1) (d3,e3)
(d5,e3) (d6,e3) (d6,e8) (d8,f1) (d8,f2) (d9,e10)
(d9,e9) (e1,f1) (e12,f3) (e12,f4) (e8,f2) (g,a1) (g,a2)
```

Appendix C

Significance of Ratios

For each of the ratios identified in Section 6.3.1, the significance of the ratios defined by Definitions 6.5 and 6.6 will be provided. An overview of the information is presented in Table 6.1.

$R(I, I)$**- I relative to I:**

Ratio values		
< 1	1	> 1
	The values for both vertices and edges will always be 1.	

$R(M, I)$ - M **relative to** I**:**

Ratio values		
< 1	1	> 1
For both vertices and edges, the cardinality of the respective sets in M are smaller than the corresponding cardinalities of the sets in I. This means that M is much smaller in size than what I is.	The number of respective vertices or edges in M is the same as in I. This is just a ratio in terms of a count, and therefore it cannot be said that I and M represent the same information. It will be necessary to take other quantities into consideration in order to determine the similarities or differences between I and M	The size of M in relation to I is much larger. This could mean that M is over specifying or that there is no match between M and I. Other quantities will need to be considered to determine the exact cause.

$R(C, I)$ - C **relative to** I**:**

Ratio values		
< 1	1	> 1
C is a subgraph isomorphism of I. The closer the value is to 1, the better C matches the information in I.	C is isomorphic to I.	This can never happen because C is derived from I. C is therefore either a copy of I or tends toward representing I. It can never over-represent I.

$R(I \setminus C, I)$ - $I \setminus C$ **relative to** I**:**

Ratio values		
< 1	1	> 1
The closer the ratio is to 0, the better the match between C, and by implication M, are to I. C is a subgraph isomorphism of I.	C is the empty set and therefore there is no match between C and I, and therefore no match possible between M and I.	C is derived from I and therefore can either be smaller or equal to I in cardinality. It can never be larger than I.

$R(I \setminus M, I)$ -$I \setminus M$ **relative to** I**:**

Ratio values		
< 1	1	> 1
The closer the ratio is to 0, the better M matches I.	There are no common elements between I and M, that is $I \cup M = \emptyset$ and therefore no possibility of matching M in terms of C onto I.	With the first operand of the set difference being I, the resultant of the set difference operation will never be larger than I itself.

$R(M \setminus C, I)$ **-$M \setminus C$ relative to I:**

Ratio values		
< 1	1	> 1
The closer the ratio if to 0, the better C represents M. A ratio closer to 1 implies that C does not represent the information presented in M. M has too many extraneous elements not in C and therefore not in I. It is possible that M is being matched with an incompatible I, or that the content of M should be re-evaluated taking the resultant sets into account.	A ratio of exactly 1 is ver unlikely, but will occur if the cardinality of $M \setminus C$ and I are the same.	A ratio greater than 1 implies that M is to begin with a large digraph, at least as large as I.

$R(C \setminus M, I)$ **-**$C \setminus M$ **relative to** I**:**

Ratio values		
< 1	1	> 1
The closer the ratio is to 0, the less there has been inferred from M when constructing C.	A ratio that equals 1 is highly unlikely. The closer the ratio is to 1, the more has been inferred when constructing C and it would be advisable to revisit the representation on M taking the resultant sets of this set difference into account.	C is ether isomorphic to I or a subgraph isomorphism and therefore the ratios will never be greater than 1.

$R(M \setminus C, M)$ **-**$M \setminus C$ **relative to** M**:**

Ratio values		
< 1	1	> 1
The closer the ratio is to 0, the better the match is between M and C and the more accurate the C is in representing M in the structure or I.	A ratio of 1 indicates that there is no common elements between M and C. In this case C must be the empty set.	The ratio will never be larger than 1 because the resultant set can never be larger than M.

$R(C \setminus M, C)$ -$C \setminus M$ relative to C:

It is possible that $C = \emptyset$ which means that there are no common elements between M and I. In this case the quantity ratio would be undefined. For $C \neq \emptyset$, the following quantity ratios are possible.

Ratio values		
< 1	1	> 1
As the ratio tends to 0, the better the representation of M in terms of C. A ratio that tends towards 1 means that more than exists in M and has been transferred to C has been inferred in C.	A ratio of exactly 1 will never occur because this would mean that C in its entirety has been inferred from M.	The ratio will never be greater than 1 because the difference cannot get bigger than what C is.

Appendix D

Excerpts from the ACM/IEEE CS Curriculum Volumes

The excepts from the ACM/IEEE Curriculum volumes which describe the Computer Science curriculum illustrate how the curricula are presented in each of the volumes. The CC2001 and CS2008 volumes always presented a table in which all KAs and their core and elective KUs are specified. In CS2013, both Strawman and Ironman, this single overview table representing the Computer Science Body of Knowledge (CS-BoK) is presented per KA. Figures D.1, D.3, D.5 and D.7 present the overview for the respective volumes.

Figures D.2, D.4, D.6 and D.8 provide an example for each of the volumes as to how the topics and learning outcomes are specified for the Data Structures KU.

There excepts presented in the appendix have reference to the discussion in Chapter 8.

D.1 CC2001

DS. Discrete Structures (43 core hours)
DS1. Functions, relations, and sets (6)
DS2. Basic logic (10)
DS3. Proof techniques (12)
DS4. Basics of counting (5)
DS5. Graphs and trees (4)
DS6. Discrete probability (6)

PF. Programming Fundamentals (38 core hours)
PF1. Fundamental programming constructs (9)
PF2. Algorithms and problem-solving (6)
PF3. Fundamental data structures (14)
PF4. Recursion (5)
PF5. Event-driven programming (4)

AL. Algorithms and Complexity (31 core hours)
AL1. Basic algorithmic analysis (4)
AL2. Algorithmic strategies (6)
AL3. Fundamental computing algorithms (12)
AL4. Distributed algorithms (3)
AL5. Basic computability (6)
AL6. The complexity classes P and NP
AL7. Automata theory
AL8. Advanced algorithmic analysis
AL9. Cryptographic algorithms
AL10. Geometric algorithms
AL11. Parallel algorithms

AR. Architecture and Organization (36 core hours)
AR1. Digital logic and digital systems (6)
AR2. Machine level representation of data (3)
AR3. Assembly level machine organization (9)
AR4. Memory system organization and architecture (5)
AR5. Interfacing and communication (3)
AR6. Functional organization (7)
AR7. Multiprocessing and alternative architectures (3)
AR8. Performance enhancements
AR9. Architecture for networks and distributed systems

OS. Operating Systems (18 core hours)
OS1. Overview of operating systems (2)
OS2. Operating system principles (2)
OS3. Concurrency (6)
OS4. Scheduling and dispatch (3)
OS5. Memory management (5)
OS6. Device management
OS7. Security and protection
OS8. File systems
OS9. Real-time and embedded systems
OS10. Fault tolerance
OS11. System performance evaluation
OS12. Scripting

NC. Net-Centric Computing (15 core hours)
NC1. Introduction to net-centric computing (2)
NC2. Communication and networking (7)
NC3. Network security (3)
NC4. The web as an example of client-server computing (3)
NC5. Building web applications
NC6. Network management
NC7. Compression and decompression
NC8. Multimedia data technologies
NC9. Wireless and mobile computing

PL. Programming Languages (21 core hours)
PL1. Overview of programming languages (2)
PL2. Virtual machines (1)
PL3. Introduction to language translation (2)
PL4. Declarations and types (3)
PL5. Abstraction mechanisms (3)
PL6. Object-oriented programming (10)
PL7. Functional programming
PL8. Language translation systems
PL9. Type systems
PL10. Programming language semantics
PL11. Programming language design

Note: The numbers in parentheses represent the minimum number of hours required to cover this material in a lecture format. It is always appropriate to include more.

HC. Human-Computer Interaction (8 core hours)
HC1. Foundations of human-computer interaction (6)
HC2. Building a simple graphical user interface (2)
HC3. Human-centered software evaluation
HC4. Human-centered software development
HC5. Graphical user-interface design
HC6. Graphical user-interface programming
HC7. HCI aspects of multimedia systems
HC8. HCI aspects of collaboration and communication

GV. Graphics and Visual Computing (3 core hours)
GV1. Fundamental techniques in graphics (2)
GV2. Graphic systems (1)
GV3. Graphic communication
GV4. Geometric modeling
GV5. Basic rendering
GV6. Advanced rendering
GV7. Advanced techniques
GV8. Computer animation
GV9. Visualization
GV10. Virtual reality
GV11. Computer vision

IS. Intelligent Systems (10 core hours)
IS1. Fundamental issues in intelligent systems (1)
IS2. Search and constraint satisfaction (5)
IS3. Knowledge representation and reasoning (4)
IS4. Advanced search
IS5. Advanced knowledge representation and reasoning
IS6. Agents
IS7. Natural language processing
IS8. Machine learning and neural networks
IS9. AI planning systems
IS10. Robotics

IM. Information Management (10 core hours)
IM1. Information models and systems (3)
IM2. Database systems (3)
IM3. Data modeling (4)
IM4. Relational databases
IM5. Database query languages
IM6. Relational database design
IM7. Transaction processing
IM8. Distributed databases
IM9. Physical database design
IM10. Data mining
IM11. Information storage and retrieval
IM12. Hypertext and hypermedia
IM13. Multimedia information and systems
IM14. Digital libraries

SP. Social and Professional Issues (16 core hours)
SP1. History of computing (1)
SP2. Social context of computing (3)
SP3. Methods and tools of analysis (2)
SP4. Professional and ethical responsibilities (3)
SP5. Risks and liabilities of computer-based systems (2)
SP6. Intellectual property (3)
SP7. Privacy and civil liberties (2)
SP8. Computer crime
SP9. Economic issues in computing
SP10. Philosophical frameworks

SE. Software Engineering (31 core hours)
SE1. Software design (8)
SE2. Using APIs (5)
SE3. Software tools and environments (3)
SE4. Software processes (2)
SE5. Software requirements and specifications (4)
SE6. Software validation (3)
SE7. Software evolution (3)
SE8. Software project management (3)
SE9. Component-based computing
SE10. Formal methods
SE11. Software reliability
SE12. Specialized systems development

CN. Computational Science (no core hours)
CN1. Numerical analysis
CN2. Operations research
CN3. Modeling and simulation
CN4. High-performance computing

Figure D.1: Overview of Knowledge Areas- CS2001 BoK [ACM/IEEE-Curriculum 2001 Task Force, 2001, Page 85]

PF3. Fundamental data structures [core]

Minimum core coverage time: 14 hours

Topics:

Primitive types
Arrays
Records
Strings and string processing
Data representation in memory
Static, stack, and heap allocation
Runtime storage management
Pointers and references
Linked structures
Implementation strategies for stacks, queues, and hash tables
Implementation strategies for graphs and trees
Strategies for choosing the right data structure

Learning objectives:

1. Discuss the representation and use of primitive data types and built-in data structures.
2. Describe how the data structures in the topic list are allocated and used in memory.
3. Describe common applications for each data structure in the topic list.
4. Implement the user-defined data structures in a high-level language.
5. Compare alternative implementations of data structures with respect to performance.
6. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables.
7. Compare and contrast the costs and benefits of_dynamic and static data structure implementations.
8. Choose the appropriate data structure for modeling a given problem.

Figure D.2: Example of the CS2001 Programming Fundamentals KA, Data Structures KU [ACM/IEEE-Curriculum 2001 Task Force, 2001, Page 90]

D.2 CS2008

DS. Discrete Structures (43 core hours)
DS/FunctionsRelationsAndSets (6)
DS/BasicLogic (10)
DS/ProofTechniques (12)
DS/BasicsOfCounting (5)
DS/GraphsAndTrees (4)
DS/DiscreteProbability (6)

PF. Programming Fundamentals (47 core hours)
PF/FundamentalConstructs (9)
PF/AlgorithmicProblemSolving (6)
PF/DataStructures (10)
PF/Recursion (4)
PF/EventDrivenProgramming (4)
PF/ObjectOriented (8)
PF/FoundationsInformationSecurity (4)
PF/SecureProgramming (2)

AL. Algorithms and Complexity (31 core hours)
AL/BasicAnalysis (4)
AL/AlgorithmicStrategies (6)
AL/FundamentalAlgorithms (12)
AL/DistributedAlgorithms (3)
AL/BasicComputability (6)
AL/PversusNP
AL/AutomataTheory
AL/AdvancedAnalysis
AL/CryptographicAlgorithms
AL/GeometricAlgorithms
AL/ParallelAlgorithms

AR. Architecture and Organization (36 core hours)
AR/DigitalLogicAndDataRepresentation (7)
AR/ComputerArchitectureAndOrganization (9)
AR/InterfacingAndI/OStrategies (3)
AR/MemoryArchitecture (5)
AR/FunctionalOrganization (6)
AR/Multiprocessing (6)
AR/PerformanceEnhancements
AR/DistributedArchitectures
AR/Devices
AR/DirectionsInComputing

OS. Operating Systems (18 core hours)
OS/OverviewOfOperatingSystems (2)
OS/OperatingSystemPrinciples (2)
OS/Concurrency (6)
OS/SchedulingandDispatch (3)
OS/MemoryManagement (3)
OS/DeviceManagement
OS/SecurityAndProtection (2)
OS/FileSystems
OS/RealTimeAndEmbeddedSystems
OS/FaultTolerance
OS/SystemPerformanceEvaluation
OS/Scripting
OS/DigitalForensics
OS/SecurityModels

NC. Net-Centric Computing (15 core hours)
NC/Introduction(2)
NC/NetworkCommunication (7)
NC/NetworkSecurity (6)
NC/WebOrganization
NC/NetworkedApplications
NC/NetworkManagement
NC/Compression
NC/MultimediaTechnologies
NC/MobileComputing

PL. Programming Languages (21 core hours)
PL/Overview(2)
PL/VirtualMachines(1)
PL/BasicLanguageTranslation(2)
PL/DeclarationsAndTypes(3)
PL/AbstractionMechanisms(3)
PL/ObjectOrientedProgramming(10)
PL/FunctionalProgramming
PL/LanguageTranslationSystems
PL/TypeSystems
PL/ProgrammingLanguageSemantics
PL/ProgrammingLanguageDesign

HC. Human-Computer Interaction (8 core hours)
HC/Foundations (6)
HC/BuildingGUIInterfaces (2)
HC/UserCcenteredSoftwareEvaluation
HC/UserCenteredSoftwareDevelopment
HC/GUIDesign
HC/GUIProgramming
HC/MultimediaAndMultimodalSystems
HC/CollaborationAndCommunication
HC/InteractionDesignForNewEnvironments
HC/HumanFactorsAndSecurity

GV. Graphics and Visual Computing (3 core hours)
GV/FundamentalTechniques (2)
GV/GraphicSystems (1)
GV/GraphicCommunication
GV/GeometricModeling
GV/BasicRendering
GV/AdvancedRendering
GV/AdvancedTechniques
GV/ComputerAnimation
GV/Visualization
GV/VirtualReality
GV/ComputerVision
GV/ComputationalGeometry
GV/GameEngineProgramming

IS. Intelligent Systems (10 core hours)
IS/FundamentalIssues (1)
IS/BasicSearchStrategies (5)
IS/KnowledgeBasedReasoning (4)
IS/AdvancedSearch
IS/AdvancedReasoning
IS/Agents
IS/NaturaLanguageProcessing
IS/MachineLearning
IS/PlanningSystems
IS/Robotics
IS/Perception

IM. Information Management (11 core hours)
IM/InformationModels (4)
IM/DatabaseSystems (3)
IM/DataModeling (4)
IM/Indexing
IM/RelationalDatabases
IM/QueryLanguages
IM/RelationalDatabaseDesign
IM/TransactionProcessing
IM/DistributedDatabases
IM/PhysicalDatabaseDesign
IM/DataMining
IM/InformationStorageAndRetrieval
IM/Hypermedia
IM/MultimediaSystems
IM/DigitalLibraries

SP. Social and Professional Issues (16 core hours)
SP/HistoryOfComputing (1)
SP/SocialContext (3)
SP/AnalyticalTools (2)
SP/ProfessionalEthics (3)
SP/Risks (2)
SP/SecurityOperations
SP/IntellectualProperty (3)
SP/PrivacyAndCivilLiberties (2)
SP/ComputerCrime
SP/EconomicsOfComputing
SP/PhilosophicalFrameworks

SE. Software Engineering (31 core hours)
SE/SoftwareDesign (8)
SE/UsingAPIs (5)
SE/ToolsAndEnvironments (3)
SE/SoftwareProcesses (2)
SE/RequirementsSpecifications (4)
SE/SoftwareVerificationValidation (3)
SE/SoftwareEvolution (3)
SE/SoftwareProjectManagement (3)
SE/ComponentBasedComputing
SE/FormalMethods
SE/SoftwareReliability
SE/SpecializedSystems
SE/RiskAssessment
SE/RobustAndSecurity-EnhancedProgramming

CN. Computational Science (no core hours)
CN/ModelingAndSimulation
CN/OperationsResearch
CN/ParallelComputation

Figure D.3: Overview of Knowledge Areas - CS2008 BoK [ACM/IEEE-Curriculum CS2008 Joint Task Force, 2008, Appendix A]

PF/DataStructures [core]
Minimum core coverage time: 10 hours

Topics:
- Representation of numeric data
- Range, precision, and rounding errors
- Arrays
- Representation of character data
- Strings and string processing
- Runtime storage management
- Pointers and references
- Linked structures
- Implementation strategies for stacks, queues, and hash tables
- Implementation strategies for graphs and trees
- Strategies for choosing the right data structure

Learning Objectives:
1. Describe the representation of numeric and character data.
2. Understand how precision and round-off can affect numeric calculations.
3. Discuss the use of primitive data types and built-in data structures.
4. Describe common applications for each data structure in the topic list.
5. Implement the user-defined data structures in a high-level language.
6. Compare alternative implementations of data structures with respect to performance.
7. Write programs that use each of the following data structures: arrays, strings, linked lists, stacks, queues, and hash tables.
8. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
9. Choose the appropriate data structure for modeling a given problem.

Figure D.4: Example of the CS2008 Programming Fundamentals KA, Data Structures KU [ACM/IEEE-Curriculum CS2008 Joint Task Force, 2008, Appendix B]

D.3 CS2013 Strawman

SDF. Software Development Fundamentals (42 Core-Tier1 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
SDF/Algorithms and Design	11		N
SDF/Fundamental Programming Concepts	10		N
SDF/Fundamental Data Structures	12		N
SDF/Development Methods	9		N

Figure D.5: Overview of the CS2013 Strawman SDF BoK [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2012, Page 139]

SDF/Fundamental Data Structures

[12 Core-Tier1 hours]

This unit builds the foundation for core concepts in the Algorithms & Complexity knowledge area, most notably in the Fundamental Data Structures & Algorithms and Basic Computability & Complexity units.

Topics:

- Arrays
- Records/structs (heterogeneous aggregates)
- Strings and string processing
- Stacks, queues, priority queues, sets & maps
- References and aliasing
- Simple linked structures
- Strategies for choosing the appropriate data structure

Learning Outcomes:

1. Discuss the appropriate use of built-in data structures. [Knowledge]
2. Describe common applications for each data structure in the topic list. [Knowledge]
3. Compare alternative implementations of data structures with respect to performance. [Evaluation]
4. Write programs that use each of the following data structures: arrays, strings, linked lists, stacks, queues, sets, and maps. [Application]
5. Compare and contrast the costs and benefits of dynamic and static data structure implementations. [Evaluation]
6. Choose the appropriate data structure for modeling a given problem. [Evaluation]

Figure D.6: Example of the CS2013 Strawman Software Development Fundamentals KA, Data Structures KU [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2012, Page141]

D.4 CS2013 Ironman

SDF. Software Development Fundamentals (43 Core-Tier1 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
SDF/Algorithms and Design	11		N
SDF/Fundamental Programming Concepts	10		N
SDF/Fundamental Data Structures	12		N
SDF/Development Methods	10		N

Figure D.7: Overview of the CS2013 Ironman SDF BoK [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2013, Page 163]

SDF/Fundamental Data Structures

[12 Core-Tier1 hours]

This unit builds the foundation for core concepts in the Algorithms & Complexity knowledge area, most notably in the Fundamental Data Structures & Algorithms and Basic Computability & Complexity units.

Topics:

- Arrays
- Records/structs (heterogeneous aggregates)
- Strings and string processing
- Stacks, queues, priority queues, sets & maps
- References and aliasing
- Simple linked structures
- Strategies for choosing the appropriate data structure

Learning Outcomes:

1. Discuss the appropriate use of built-in data structures. [Knowledge]
2. Describe common applications for each data structure in the topic list. [Knowledge]
3. Compare alternative implementations of data structures with respect to performance. [Evaluation]
4. Write programs that use each of the following data structures: arrays, strings, linked lists, stacks, queues, sets, and maps. [Application]
5. Compare and contrast the costs and benefits of dynamic and static data structure implementations. [Evaluation]
6. Choose the appropriate data structure for modeling a given problem. [Evaluation]

Figure D.8: Example of the CS2013 Ironman Software Development Fundamentals KA, Data Structures KU [The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, 2013, Page165]

Appendix E

Application Scenarios - Cardinalities and ratios

E.1 Scenario 1

	Original				KA equivalences				KA and KU equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$
$R(I,I)$	476	477	1	1	495	501	1	1	576	611	1	1
$R(M,I)$	510	519	1.07	1.09	529	543	1.07	1.08	611	647	1.06	1.06
$R(C,I)$	356	356	0.75	0.75	375	380	0.76	0.76	457	491	0.79	0.80
$R(I \setminus C,I)$	120	121	0.25	0.25	120	121	0.24	0.24	119	120	0.21	0.20
$R(I \setminus M,I)$	138	248	0.29	0.52	138	248	0.28	0.50	121	158	0.21	0.26
$R(M \setminus C,I)$	172	290	0.36	0.61	172	290	0.35	0.58	156	194	0.27	0.32
$R(C \setminus M,I)$	18	127	0.04	0.27	18	127	0.04	0.25	2	38	0.01	0.06
$R(M \setminus C,M)$	172	290	0.34	0.56	172	290	0.33	0.53	156	194	0.26	0.30
$R(C \setminus M,C)$	18	127	0.05	0.36	18	127	0.05	0.33	3	38	0.01	0.08

Table E.1: Scenario 1 - CC2001(I) and CS2008(M)

	Original				KA equivalences				KA and KU equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_{\mathcal{X}}\|$	$\|E'_{\mathcal{X}}\|$	$V^{ratio}_{\mathcal{X}}$	$E^{ratio}_{\mathcal{X}}$	$\|V_{\mathcal{X}}\|$	$\|E'_{\mathcal{X}}\|$	$V^{ratio}_{\mathcal{X}}$	$E^{ratio}_{\mathcal{X}}$	$\|V_{\mathcal{X}}\|$	$\|E'_{\mathcal{X}}\|$	$V^{ratio}_{\mathcal{X}}$	$E^{ratio}_{\mathcal{X}}$
$R(I,I)$	476	477	1	1	495	501	1	1	576	611	1	1
$R(M,I)$	683	683	1.43	1.43	701	707	1.42	1.41	781	831	1.36	1.36
$R(C,I)$	192	192	0.40	0.40	211	216	0.43	0.43	300	334	0.52	0.55
$R(I \setminus C,I)$	284	285	0.60	0.60	284	285	0.57	0.57	276	277	0.48	0.45
$R(I \setminus M,I)$	300	350	0.63	0.73	298	346	0.60	0.69	280	303	0.49	0.50
$R(M \setminus C,I)$	507	556	1.07	1.17	504	552	1.02	1.10	485	523	0.84	0.86
$R(C \setminus M,I)$	16	65	0.03	0.14	14	61	0.03	0.12	4	26	0.01	0.04
$R(M \setminus C,M)$	507	556	0.74	0.81	504	552	0.72	0.78	485	523	0.62	0.63
$R(C \setminus M,C)$	16	65	0.08	0.34	14	61	0.07	0.28	4	26	0.01	0.08

Table E.2: Scenario 1 - CC2001(I) and CS2013S(M)

	Original				KA equivalences				KA and KU equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$
$R(I,I)$	476	477	1	1	495	501	1	1	576	611	1	1
$R(M,I)$	732	732	1.54	1.53	750	756	1.52	1.51	830	876	1.44	1.44
$R(C,I)$	185	184	0.39	0.39	204	208	0.41	0.42	293	326	0.51	0.53
$R(I \setminus C,I)$	291	293	0.61	0.61	291	293	0.59	0.58	283	285	0.49	0.47
$R(I \setminus M,I)$	308	355	0.65	0.74	305	349	0.62	0.70	287	313	0.50	0.52
$R(M \setminus C,I)$	564	610	1.18	1.28	560	604	1.13	1.21	541	578	0.94	0.95
$R(C \setminus M,I)$	17	62	0.04	0.13	14	56	0.03	0.11	5	28	0.01	0.05
$R(M \setminus C,M)$	564	610	0.77	0.83	560	604	0.75	0.80	541	578	0.65	0.66
$R(C \setminus M,C)$	17	62	0.09	0.34	14	56	0.07	0.27	5	28	0.02	0.09

Table E.3: Scenario 1 - CC2001(I) and CS2013I(M)

	Original				KA equivalences				KA and KU equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$
$R(I,I)$	683	683	1	1	701	707	1	1	781	831	1	1
$R(M,I)$	732	732	1.07	1.07	750	756	1.07	1.07	830	876	1.06	1.05
$R(C,I)$	593	593	0.87	0.87	611	617	0.87	0.87	692	742	0.89	0.89
$R(I \setminus C,I)$	90	90	0.13	0.13	90	90	0.13	0.13	89	89	0.11	0.11
$R(I \setminus M,I)$	93	208	0.14	0.30	92	205	0.13	0.29	89	203	0.11	0.25
$R(M \setminus C,I)$	142	257	0.21	0.38	141	254	0.20	0.36	138	248	0.18	0.30
$R(C \setminus M,I)$	3	118	0	0.17	2	115	0	0.16	0	114	0	0.14
$R(M \setminus C,M)$	142	257	0.19	0.35	141	254	0.19	0.34	138	248	0.17	0.28
$R(C \setminus M,C)$	3	118	0.01	0.20	2	115	0	0.19	0	114	0	0.15

Table E.4: Scenario 1 - CS2013S(I) and CS2013I(M)

E.2 Scenario 2

	Original				KU equivalence				Topic equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_{\mathcal{X}}\|$	$\|E'_{\mathcal{X}}\|$	$V^{ratio}_{\mathcal{X}}$	$E^{ratio}_{\mathcal{X}}$	$\|V_{\mathcal{X}}\|$	$\|E'_{\mathcal{X}}\|$	$V^{ratio}_{\mathcal{X}}$	$E^{ratio}_{\mathcal{X}}$	$\|V_{\mathcal{X}}\|$	$\|E'_{\mathcal{X}}\|$	$V^{ratio}_{\mathcal{X}}$	$E^{ratio}_{\mathcal{X}}$
$R(I,I)$	14	12	1	1	16	16	1	1	36	41	1	1
$R(M,I)$	16	15	1.14	1.25	19	19	1.19	1.19	34	39	0.94	0.95
$R(C,I)$	2	1	0.14	0.08	7	7	0.44	0.44	26	31	0.72	0.76
$R(I \setminus C,I)$	12	11	0.86	0.92	9	9	0.56	0.56	10	10	0.28	0.24
$R(I \setminus M,I)$	11	11	0.79	0.92	9	9	0.56	0.56	10	10	0.28	0.24
$R(M \setminus C,I)$	14	14	1	1.17	12	12	0.75	0.75	8	8	0.22	0.2
$R(C \setminus M,I)$	0	0	0	0	0	0	0	0	0	0	0	0
$R(M \setminus C,M)$	14	14	0.88	0.93	12	12	0.63	0.63	8	8	0.24	0.21
$R(C \setminus M,C)$	0	0	0	0	0	0	0	0	0	0	0	0

Table E.5: Scenario 2 - CC2001(I) and CS2008(M)

	Original				KU equivalence				Topic equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$
$R(I,I)$	14	12	1	1	19	19	1	1	34	39	1	1
$R(M,I)$	16	15	1.14	1.25	16	16	0.84	0.84	36	41	1.06	1.05
$R(C,I)$	2	1	0.14	0.08	7	7	0.37	0.37	26	31	0.76	0.79
$R(I \setminus C,I)$	12	11	0.86	0.92	12	12	0.63	0.63	8	8	0.24	0.21
$R(I \setminus M,I)$	11	11	0.79	0.92	12	12	0.63	0.63	8	8	0.24	0.21
$R(M \setminus C,I)$	14	14	1	1.17	9	9	0.47	0.47	10	10	0.29	0.26
$R(C \setminus M,I)$	0	0	0	0	0	0	0	0	0	0	0	0
$R(M \setminus C,M)$	14	14	0.88	0.93	9	9	0.56	0.56	10	10	0.28	0.24
$R(C \setminus M,C)$	0	0	0	0	0	0	0	0	0	0	0	0

Table E.6: Scenario 2 - CS2008(I) and CC2001(M)

	Original				KA equivalence				Topic equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$
$R(I,I)$	21	20	1	1	24	24	1	1	39	44	1	1
$R(M,I)$	21	20	1	1	24	24	1	1	42	48	1.08	1.09
$R(C,I)$	14	12	0.67	0.6	18	18	0.75	0.75	37	42	0.95	0.95
$R(I \setminus C,I)$	7	8	0.33	0.4	6	6	0.25	0.25	2	2	0.05	0.05
$R(I \setminus M,I)$	7	8	0.33	0.4	6	6	0.25	0.25	2	2	0.05	0.05
$R(M \setminus C,I)$	7	8	0.33	0.4	6	6	0.25	0.25	5	6	0.13	0.14
$R(C \setminus M,I)$	0	0	0	0	0	0	0	0	0	0	0	0
$R(M \setminus C,M)$	7	8	0.33	0.4	6	6	0.25	0.25	5	6	0.12	0.13
$R(C \setminus M,C)$	0	0	0	0	0	0	0	0	0	0	0	0

Table E.7: Scenario 2 - CS2013S(I) and CS2013I(M)

	Original				KA equivalence				Topic equivalences			
Ratio $(R(\mathcal{X},\mathcal{Y}))$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$	$\|V_\mathcal{X}\|$	$\|E'_\mathcal{X}\|$	$V^{ratio}_\mathcal{X}$	$E^{ratio}_\mathcal{X}$
$R(I,I)$	21	20	1	1	24	24	1	1	42	48	1	1
$R(M,I)$	21	20	1	1	24	24	1	1	39	44	0.93	0.92
$R(C,I)$	14	12	0.67	0.6	18	18	0.75	0.75	37	42	0.88	0.88
$R(I \setminus C,I)$	7	8	0.33	0.4	6	6	0.25	0.25	5	6	0.12	0.13
$R(I \setminus M,I)$	7	8	0.33	0.4	6	6	0.25	0.25	5	6	0.12	0.13
$R(M \setminus C,I)$	7	8	0.33	0.4	6	6	0.25	0.25	2	2	0.05	0.04
$R(C \setminus M,I)$	0	0	0	0	0	0	0	0	0	0	0	0
$R(M \setminus C,M)$	7	8	0.33	0.4	6	6	0.25	0.25	2	2	0.05	0.05
$R(C \setminus M,C)$	0	0	0	0	0	0	0	0	0	0	0	0

Table E.8: Scenario 2 - CS2013I(I) and CS2013S(M)

Appendix F

BSc Computer Science Degree

FUNDAMENTAL MODULES				
Code	**Module**	**Credits**	**Semester Hours**	**Period**
Year-level 1 (at least 20 credits)				
Pass an exemption examination in CIL 111 **or**				
CIL111	Computer Literacy	4	2	S1
CIL121	Information Literacy	4	2	S2
Pass an exemption examination in Academic Literacy **AND**				
EOT162	Academic writing skills	6	1.5	Q2
EOT164	Communication in organisations	6	3	S2
OR				
EOT110	Academic Literacy	6	3	S1
EOT120	Academic Literacy	6	3	S2
Year-level 2 (8 credits)				
JCP202	Community-based Project	8	1.71	Year

CORE MODULES				
Code	**Module**	**Credits**	**Semester Hours**	**Period**
Year-level 1 (120 credits)				
COS110	Program Design: Introduction	16	5	S2
COS121	Software Modelling	16	5	S2
COS132	Introduction to Programming	16	5	S1
COS151	Introduction to Computer Science	8	3	S1
ERA284	Computer Architecture	16	4	S2
WTW114	Calculus	16	5	S1
WTW115	Discrete Structures	8	3	S1
WTW126	Linear Algebra	8	3	S2
WTW128	Calculus	8	3	0
WTW152	Mathematical Modeling	8	3	S1
Year-level 2 (110 credits)				
COS212	Data Structures and Algorithms	16	5	S1
COS222	Operating Systems	16	5	S1
COS226	Concurrent Systems	16	5	S2
COS216	Netcentric Computer Systems	16	5	S1
INF214	Informatics: Database Design	14	4	S1
INL240	Information Science: Social and ethical impact	20	4	S1
WTW285	Discrete Structures	12	3	S2
Year-level 3 (81 credits)				
COS301	Software Engineering	27	5	Year
COS330	Computer Security and Ethics	18	3	S2
COS332	Computer Networks	18	3	S1
COS333	Programming Languages	18	3	S1

ELECTIVE MODULES				
Code	**Module**	**Credits**	**Semester Hours**	**Period**
Year-level 1 (at least 78 credits)				
Statistics (at least 26 credits)				
A choice between Mathemetical Statistics or Statistics subject to the grade 12 Mathematics level				
WST111	Mathematical Statistics	16	5	S1
WST121	Mathematical Statistics	16	5	S2
OR				
STK110	Descriptive Statistics	13	3	S1
STK120	Multivariate Statistics	13	3	S2
Science (32 credits)				
Students with Physical Science Level 4 in grade 12 can choose between Physics, Chemistry or Biological Sciences				
Physics				
PHY171	First Course in Physics	32	10	Year
OR				
Chemistry				
CMY117	General Chemistry	16	5	S1
CMY127	General Chemistry	16	5	S2
OR				
Biological Sciences				
MLB111	Molecular and Cell Biology	16	5	S1
BOT161	Plant Biology	8	3	S2
MBY161	Introduction to Microbiology	8	3	S2
OR				
Students without Physical Science in grade 12 are required to take Geology				
GLY151	Introductory Geology	8	3	Q1
GLY152	Physical Geology	8	3	Q2
GLY161	Historical Geology	8	3	Q4
GLY162	Environmental Geology	8	3	Q3
Other (at least 20 credits)				
At least 20 credits from the Faculties of Humanities or Economic and Management Sciences for which the student has the prerequisites				
Year-level 2				
Additional electives from second year modules in order to satisfy third year elective module prerequisites				
Year-level 3 (at least 63 credits)				
63 credits on third year level for which the student has the prerequisites from: *Computer Science including EMK310* *Information Science* *Mathematics* *Mathematical Statistics* *Physics* *Chemistry*				

Appendix G

Scenario 3 - Vertices of difference sets

This appendix provides the output of the Graph Trans-morphism Algorithm for the difference sets for vertices. The totals presented are the total number of verities in the set as provided in Table 11.7. The vertices in each of the sets have been clustered in the presentation in terms of their function within the set. This means that for the curricula volumes the clusters are in terms of KAs, KUs and Topics. The clustering for the real-world curriculum is in terms of year-levels, modules and topics. The topics are of interest for the discussion in Section 11.4.5.

G.1 CC2001 (I) and BSc CS (M)

Quantity	Vertices
$I \setminus C$ Total: 88	T00001 T00013_1 T00019 T00022 T00053 T00088 T00092 T00101 T00105 T00106 T00112 T00116 T00123 T00125_1 T00145 T00148 T00167 T00214 T00215 T00218 T00220 T00230 T00231 T00249 T00258 T00262 T00263 T00280 T00281 T00284 T00292 T00295 T00304 T00319 T00327 T00328 T00333 T00339 T00347 T00348 T00349 T00355 T00364 T00390 T00393 T00396 T00402 T00417 T00449 T00461 T00494 T00524 T00531 T00535 T00576 T00593 T00610 T00614 T00632 T00653_1 T00668 T00674 T00674_1 T00674_2 T00682_3 T00688 T00711 T00726 T00734 T00736 T00739 T00758 T00761 T00780 T00792 T00803 T00812_2 T00812_4 T00822 T00943 T00944 T00945 T00946 T00973 U0038 U0041 U0053 U0107

Quantity	Vertices
$I \setminus M$ Total: 165	AL AR DS GV HC IM IS NC OS PF PL SE SP T00001 T00013_1 T00019 T00022 T00053 T00065 T00088 T00092 T00101 T00105 T00106 T00112 T00116 T00123 T00125 T00125_1 T00145 T00148 T00167 T00214 T00215 T00218 T00220 T00230 T00231 T00249 T00258 T00262 T00263 T00280 T00281 T00284 T00292 T00295 T00304 T00319 T00327 T00328 T00333 T00339 T00347 T00348 T00349 T00355 T00364 T00390 T00393 T00396 T00402 T00417 T00449 T00461 T00489 T00494 T00524 T00531 T00535 T00576 T00593 T00610 T00614 T00632 T00653_1 T00668 T00674 T00674_1 T00674_2 T00682 T00682_3 T00688 T00711 T00722 T00726 T00734 T00736 T00739 T00758 T00761 T00780 T00792 T00803 T00812_2 T00812_4 T00822 T00943 T00944 T00945 T00946 T00973 U0001 U0009 U0010 U0012 U0014 U0015 U0016 U0018 U0019 U0021 U0027 U0030 U0032 U0033 U0036 U0037 U0038 U0041 U0045 U0046 U0048 U0049 U0050 U0051 U0052 U0053 U0057 U0060 U0064 U0068 U0070 U0071 U0072 U0073 U0074 U0077 U0078 U0079 U0080 U0084 U0087 U0089 U0090 U0092 U0093 U0098 U0099 U0102 U0104 U0107 U0109 U0111 U0113 U0114 U0115 U0116 U0117 U0119 U0120 U0121 U0125 U0128 U0129

Quantity	Vertices
$M \setminus C$ Total: 21	COS110 COS121 COS132 COS151 COS212 COS216 COS222 COS226 COS301 COS330 COS332 COS333 ERA284 INF214 INL204 INL240 WTW115 WTW285 Year1 Year2 Year3

Quantity	Vertices
$C \setminus M$ Total: 77	AL AR DS GV HC IM IS NC OS PF PL SE SP T00065 T00125 T00489 T00682 T00722 U0001 U0009 U0010 U0012 U0014 U0015 U0016 U0018 U0019 U0021 U0027 U0030 U0032 U0033 U0036 U0037 U0045 U0046 U0048 U0049 U0050 U0051 U0052 U0057 U0060 U0064 U0068 U0070 U0071 U0072 U0073 U0074 U0077 U0078 U0079 U0080 U0084 U0087 U0089 U0090 U0092 U0093 U0098 U0099 U0102 U0104 U0109 U0111 U0113 U0114 U0115 U0116 U0117 U0119 U0120 U0121 U0125 U0128 U0129

G.2 CS2013I (I) and BSc CS with CC2001 topics (M)

Quantity	Vertices
$I \setminus C$ Total: 575	CN_1 GV_1 IAS NC_1 PD SF T00001 T00019 T00022 T00055_1 T00058_1 T00058_2 T00058_3 T00068_1 T00086_1 T00086_2 T00089_1 T00089_2 T00095_2 T00096_1 T00101 T00105 T00106 T00107_1 T00116 T00117_2 T00120_1 T00123_2 T00125_1 T00135_1 T00135_2 T00139_1 T00145_1 T00148 T00153_3 T00156_1 T00163_1 T00167 T00193 T00211_3 T00211_5 T00212_3 T00216_1 T00216_2 T00217_1 T00217_2 T00230 T00237_1 T00248_1 T00250_1 T00250_2 T00254_1 T00256_1 T00256_2 T00269_1 T00272_1 T00273_1 T00305_1 T00327 T00328 T00333_1 T00336_1 T00339 T00356_2 T00357_2 T00358_1 T00362_1 T00371_1 T00371_2 T00371_3 T00372_2 T00382_1 T00402 T00403_1 T00403_2 T00403_4 T00453_1 T00458_1 T00461 T00475 T00482_1 T00482_2 T00482_3 T00482_4 T00484_1 T00486_1 T00495_1 T00497_1 T00508 T00517_1 T00522_1 T00524 T00526_1 T00531_1 T00535_1 T00536_1 T00543_1 T00546_2 T00551_1 T00555_1 T00557_1 T00567_1 T00582_1 T00588_1 T00593_1 T00594 T00598_1 T00600_1 T00609_2 T00610_1 T00612_1 T00613_1 T00623_1 T00624_1 T00637 T00645_1 T00654_1 T00655_1 T00663_1 T00664_2 T00671_1 T00671_2 T00674 T00674_3 T00677_1 T00689_1 T00699_1 T00711 T00728_1 T00729_1 T00733_1 T00739 T00741_1 T00743_1 T00752_1 T00758_1 T00773_1 T00792_2 T00793_2 T00793_3 T00795_1 T00796_1 *Continued on next page...*

Quantity	Vertices
$I \setminus C$ (*cont.*)	T00796_2 T00797_1 T00822 T00829_2 T00832 T00840_1 T00860_1 T00862_1 T00862_3 T00883 T00903_1 T00928_1 T01016_1 T01016_2 T01027_1 T01028 T01029 T01031_1 T01032_1 T01055 T01055_1 T01056 T01057 T01058 T01059 T01060 T01064_2 T01069 T01070 T01087 T01088 T01089 T01090 T01091 T01092 T01093 T01094 T01095 T01096 T01096_1 T01097 T01098 T01099 T01100 T01101 T01102 T01103 T01104 T01105 T01106 T01107 T01108 T01109 T01113 T01117 T01162 T01163 T01164 T01165 T01166 T01167 T01168 T01171_1 T01172_1 T01174 T01193_1 T01211_2 T01215_1 T01256_1 T01259_1 T01267 T01268 T01269 T01270 T01271 T01272 T01286 T01288 T01290 T01291 T01292 T01293 T01294 T01295 T01296 T01296_1 T01297 T01298 T01299 T01300 T01301 T01302 T01303 T01304 T01305 T01306 T01307 T01398 T01399 T01400 T01401 T01402 T01403 T01404 T01405 T01406 T01407 T01408 T01409 T01410_1 T01411 T01412 T01413 T01414 T01415 T01416 T01417 T01418 T01419 T01420 T01421 T01448 T01449 T01451 T01451_1 T01451_2 T01451_3 T01451_4 T01452 T01453 T01454 T01455 T01456 T01457 T01458_1 T01459 T01460 T01461_1 T01462 T01463 T01463_1 T01463_2 T01463_3 T01464 T01464_1 T01464_2 T01464_3 T01464_4 T01465 T01466 T01470 T01471 T01472 T01473 T01474 T01477 T01478 T01479 T01495 T01496 T01497 T01498 T01499 T01500 T01501 T01502_1 T01503 T01504 T01505 T01507 T01507_1 T01508 T01509 T01510 T01511 T01512_1 T01513 T01514 T01514_1 T01514_2 T01514_3 T01515 T01516 T01517 T01517_1 T01517_2 T01518 T01519 *Continued on next page...*

Quantity	Vertices
$I \setminus C$ (*cont.*)	T01521_1 T01522 T01523 T01524 T01524_1 T01524_3 T01525 T01525_1 T01525_2 T01525_3 T01525_4 T01526 T01527 T01528 T01529 T01530 T01530_1 T01531_1 T01531_2 T01533 T01534 T01535_1 T01535_2 T01535_3 T01535_4 T01535_5 T01536 T01537_1 T01538 T01538_1 T01539 T01540 T01540_1 T01540_2 T01540_3 T01541 T01542 T01543 T01544 T01545 T01546 T01547 T01548 T01549 T01550 T01551 T01557 T01559 T01560 T01561 T01562 T01563 T01564 T01565 T01566 T01567 T01568 T01569 T01570_1 T01571 T01572 T01573 T01574 T01575 T01576 T01577 T01578 T01579 T01580 T01581 T01582 T01583 T01584 T01586 T01587 T01588 T01589 T01590_1 T01591 T01592 T01593 T01594_1 T01596 T01597 T01598 T01599 T01600 T01601 T01602 T01603 T01604_1 T01605 T01606 T01607 T01608 T01609 T01610_1 T01611 T01612 T01613 T01614 T01615 T01616 T01617_1 T01618 T01619 T01620 T01621 T01622 T01623 T01624 T01625 T01626 T01627 T01628 T01629 T01630 T01631 T01632 T01633 T01634 T01670 T01670_1 T01671 T01672 T01673 T01674 T01675 T01676 T01677 T01678 T01679 T01680 T01681 T01682 T01683 T01684 T01685 T01686 T01687 T01688 T01689 T01690 T01691 T01692 T01693 T01694 T01695 T01696 T01697 T01698 T01699 T01700 T01701 T01702 T01703 T01704 T01705 T01706 T01707 T01708 T01709 T01710 T01711 T01712 T01713 T01714 T01715 T01716 T01717 T01718 T01719 T01720 T01721 T01722 T01723 T01724 T01725 T01726 T01727 T01728 T01729 T01730 T01731 T01732 T01733 T01734 T01735 T01736 T01737 T01738 T01739 T01740 *Continued on next page...*

Quantity	Vertices
$I \setminus C$ (*cont.*)	U0037 U0041_1 U0047 U0048_1 U0070 U0073_1 U0074_1 U0087 U0094_1 U0098 U0112 U0114 U0116 U0118 U0127_1 U0133_1 U0144 U0147_1 U0152_1 U0156 U0157 U0164_1 U0165 U0169_1 U0170 U0173 U0176 U0177 U0178 U0179 U0179_1 U0180 U0186 U0187 U0188 U0189 U0193 U0206 U0207 U0208 U0209 U0210 U0211_1 U0212_1 U0213 U0214 U0215 U0216 U0217 U0218 U0236 U0237

Quantity	Vertices
$I \setminus M$ Total: 625	AL AR CN_1 DS GV_1 HCI IAS IM IS NC_1 OS PD PL SDF SE SF SP T00001 T00019 T00022 T00055_1 T00058_1 T00058_2 T00058_3 T00068_1 T00086_1 T00086_2 T00089_1 T00089_2 T00095_2 T00096_1 T00101 T00105 T00106 T00107_1 T00116 T00117_2 T00120_1 T00123_2 T00125 T00125_1 T00135_1 T00135_2 T00139_1 T00145_1 T00148 T00153_3 T00156_1 T00163_1 T00167 T00193 T00211_3 T00211_5 T00212_3 T00216_1 T00216_2 T00217_1 T00217_2 T00230 T00237_1 T00248_1 T00250_1 T00250_2 T00254_1 T00256_1 T00256_2 T00269_1 T00272_1 T00273_1 T00305_1 T00327 T00328 T00333_1 T00336_1 T00339 T00356_2 T00357_2 T00358_1 T00362_1 T00371_1 T00371_2 T00371_3 T00372_2 T00382_1 T00402 T00403_1 T00403_2 T00403_4 T00453_1 T00458_1 T00461 T00475 T00482_1 T00482_2 T00482_3 T00482_4 T00484_1 T00486_1 T00489 T00495_1 T00497_1 T00508 T00517_1 T00522_1 T00524 T00526_1 T00531_1 T00535_1 T00536_1 T00543_1 T00546_2 T00551_1 T00555_1 T00557_1 T00567_1 T00582_1 T00588_1 T00593_1 T00594 T00598_1 T00600_1 T00609_2 T00610_1 T00612_1 T00613_1 T00623_1 T00624_1 T00637 T00645_1 T00654_1 T00655_1 T00663_1 T00664_2 T00671_1 T00671_2 T00674 T00674_3 T00677_1 T00689_1 T00699_1 T00711 T00728_1 T00728_2 T00729_1 T00733_1 T00739 T00741_1 T00743_1 T00752_1 T00758_1 T00773_1 T00792_2 T00793_2 T00793_3 T00795_1 T00796_1 T00796_2 T00797_1 T00822 T00829_2 T00832 T00840_1 T00860_1 T00862_1 T00862_3 T00883 T00903_1 T00928_1 T01016_1 T01016_2 T01027_1 T01028 T01029 T01030 *Continued on next page...*

Quantity	Vertices
$I \setminus M$ *(cont.)*	T01031_1 T01032_1 T01055 T01055_1 T01056 T01057 T01058 T01059 T01060 T01064_2 T01069 T01070 T01087 T01088 T01089 T01090 T01091 T01092 T01093 T01094 T01095 T01096 T01096_1 T01097 T01098 T01099 T01100 T01101 T01102 T01103 T01104 T01105 T01106 T01107 T01108 T01109 T01113 T01117 T01162 T01163 T01164 T01165 T01166 T01167 T01168 T01171_1 T01172_1 T01174 T01193_1 T01211_2 T01215_1 T01256_1 T01259_1 T01267 T01268 T01269 T01270 T01271 T01272 T01286 T01287 T01288 T01290 T01291 T01292 T01293 T01294 T01295 T01296 T01296_1 T01297 T01298 T01299 T01300 T01301 T01302 T01303 T01304 T01305 T01306 T01307 T01398 T01399 T01400 T01401 T01402 T01403 T01404 T01405 T01406 T01407 T01408 T01409 T01410_1 T01411 T01412 T01413 T01414 T01415 T01416 T01417 T01418 T01419 T01420 T01421 T01448 T01449 T01451 T01451_1 T01451_2 T01451_3 T01451_4 T01452 T01453 T01454 T01455 T01456 T01457 T01458_1 T01459 T01460 T01461_1 T01462 T01463 T01463_1 T01463_2 T01463_3 T01464 T01464_1 T01464_2 T01464_3 T01464_4 T01465 T01466 T01470 T01471 T01472 T01473 T01474 T01477 T01478 T01479 T01495 T01496 T01497 T01498 T01499 T01500 T01501 T01502_1 T01503 T01504 T01505 T01507 T01507_1 T01508 T01509 T01510 T01511 T01512_1 T01513 T01514 T01514_1 T01514_2 T01514_3 T01515 T01516 T01517 T01517_1 T01517_2 T01518 T01519 T01521_1 T01522 T01523 T01524 T01524_1 T01524_3 T01525 T01525_1 T01525_2 T01525_3 T01525_4 T01526 T01527 T01528 T01529 T01530 T01530_1 T01531_1 T01531_2 T01533 T01534 T01535_1 T01535_2 T01535_3 *Continued on next page...*

Quantity	Vertices
$I \setminus M$ *(cont.)*	T01535_4 T01535_5 T01536 T01537_1 T01538 T01538_1 T01539 T01540 T01540_1 T01540_2 T01540_3 T01541 T01542 T01543 T01544 T01545 T01546 T01547 T01548 T01549 T01550 T01551 T01556 T01557 T01559 T01560 T01561 T01562 T01563 T01564 T01565 T01566 T01567 T01568 T01569 T01570_1 T01571 T01572 T01573 T01574 T01575 T01576 T01577 T01578 T01579 T01580 T01581 T01582 T01583 T01584 T01586 T01587 T01588 T01589 T01590_1 T01591 T01592 T01593 T01594_1 T01596 T01597 T01598 T01599 T01600 T01601 T01602 T01603 T01604_1 T01605 T01606 T01607 T01608 T01609 T01610_1 T01611 T01612 T01613 T01614 T01615 T01616 T01617_1 T01618 T01619 T01620 T01621 T01622 T01623 T01624 T01625 T01626 T01627 T01628 T01629 T01630 T01631 T01632 T01633 T01634 T01670 T01670_1 T01671 T01672 T01673 T01674 T01675 T01676 T01677 T01678 T01679 T01680 T01681 T01682 T01683 T01684 T01685 T01686 T01687 T01688 T01689 T01690 T01691 T01692 T01693 T01694 T01695 T01696 T01697 T01698 T01699 T01700 T01701 T01702 T01703 T01704 T01705 T01706 T01707 T01708 T01709 T01710 T01711 T01712 T01713 T01714 T01715 T01716 T01717 T01718 T01719 T01720 T01721 T01722 T01723 T01724 T01725 T01726 T01727 T01728 T01729 T01730 T01731 T01732 T01733 T01734 T01735 T01736 T01737 T01738 T01739 T01740 *Continued on next page...*

Quantity	Vertices
$I \setminus M$ *(cont.)*	U0009 U0012 U0014_1 U0016 U0018 U0027 U0030 U0032 U0036 U0037 U0041_1 U0045 U0047 U0048_1 U0050 U0050_2 U0051 U0052_2 U0060 U0070 U0071 U0073_1 U0074_1 U0077 U0078 U0079 U0087 U0089 U0090 U0092 U0094_1 U0098 U0099_1 U0102 U0109 U0112 U0114 U0115 U0116 U0117 U0118 U0120_1 U0121_1 U0127_1 U0133_1 U0144 U0147_1 U0148 U0152_1 U0156 U0157 U0164_1 U0165 U0169_1 U0170 U0172 U0173 U0176 U0177 U0178 U0179 U0179_1 U0180 U0186 U0187 U0188 U0189 U0193 U0204 U0205 U0206 U0207 U0208 U0209 U0210 U0211_1 U0212_1 U0213 U0214 U0215 U0216 U0217 U0218 U0236 U0237

Quantity	Vertices
$M \setminus C$ Total: 225	COS110 COS121 COS132 COS151 COS212 COS216 COS222 COS226 COS301 COS330 COS332 COS333 ERA284 INF214 INL204 INL240 WTW115 WTW285 T00005 T00008 T00009 T00012 T00021 T00040 T00045 T00046 T00047 T00055 T00058 T00065_1 T00065_2 T00065_3 T00065_4 T00065_5 T00068 T00072 T00077 T00082 T00082_1 T00082_2 T00082_3 T00086 T00087 T00089 T00095 T00096 T00097 T00102 T00103 T00107 T00110 T00117_1 T00120 T00135 T00137 T00147 T00150 T00151 T00160 T00163 T00166 T00173 T00207 T00216 T00217 T00237 T00248 T00250 T00254 T00256 T00261 T00269 T00271 T00273 T00287 T00289 T00290 T00291 T00293 T00297 T00298 T00299_1 T00303 T00305 T00314 T00315 T00316 T00317 T00318 T00320 T00330 T00336 T00338 T00340 T00345 T00346 T00350 T00353 T00357 T00358 T00362 T00363 T00371 T00379 T00382 T00388 T00398 T00401 T00403 T00404 T00418 T00421 T00422 T00428 T00439 T00442 T00444 T00453 T00456 T00458 T00459 T00462 T00469 T00478 T00479 T00481 T00486 T00491 T00495 T00496 T00497 T00499 T00500 T00506 T00510 T00522 T00526 T00541 T00543 T00545 T00546 T00547 T00548 T00550 T00551 T00559 T00562 T00564 T00567 T00582 T00588 T00591 T00598 T00608 T00609 T00612 T00613 T00617 T00619 T00624 T00633 T00635 T00645 T00649 T00654 T00655 T00656 T00659 T00661 T00663 T00664 T00665 T00666 T00669 T00670 T00677 T00682_1 T00682_2 T00682_4 T00689 T00694 T00699 T00700 T00703 T00704 T00705 T00717 T00720 T00723 T00728 T00729 T00731 T00738 T00740 T00741 T00746 T00752 T00760 T00763 T00764 T00766 T00773 T00774 T00781 T00786 T00793 T00794 T00795 *Continued on next page...*

Quantity	Vertices
$M \setminus C$ (*cont.*)	T00796 T00797 T00801 T00810 T00812 T00812_1 T00812_3 T00824 T00972 T00974 T00975 T00977 T00978 T00979 Year1 Year2 Year3

Quantity	Vertices
$C \setminus M$ Total: 50	AL AR DS HCI IM IS OS PL SDF SE SP T00125 T00489 T00728_2 T01030 T01287 T01556 U0009 U0012 U0014_1 U0016 U0018 U0027 U0030 U0032 U0036 U0045 U0050 U0050_2 U0051 U0052_2 U0060 U0071 U0077 U0078 U0079 U0089 U0090 U0092 U0099_1 U0102 U0109 U0115 U0117 U0120_1 U0121_1 U0148 U0172 U0204 U0205

G.3 CS2013I (I) and BSc CS with CS2013I topics (M)

Quantity	Vertices
$I \setminus C$ Total: 271	T00001 T00019 T00022 T00086_2 T00089_1 T00096_1 T00101 T00105 T00116 T00117_2 T00123_2 T00148 T00153_3 T00156_1 T00167 T00193 T00211_3 T00211_5 T00212_3 T00216_2 T00217_2 T00230 T00250_1 T00272_1 T00328 T00339 T00402 T00482_2 T00482_3 T00508 T00582_1 T00593_1 T00594 T00664_2 T00671_1 T00671_2 T00674_3 T00711 T00733_1 T00739 T00743_1 T00758_1 T00793_2 T00795_1 T00796_2 T00797_1 T00832 T00860_1 T00883 T01027_1 T01028 T01031_1 T01032_1 T01056 T01057 T01058 T01060 T01070 T01088 T01096_1 T01099 T01103 T01104 T01106 T01107 T01108 T01109 T01113 T01163 T01167 T01168 T01171_1 T01174 T01215_1 T01256_1 T01267 T01268 T01271 T01286 T01288 T01291 T01292 T01293 T01294 T01296 T01296_1 T01297 T01298 T01299 T01300 T01302 T01304 T01305 T01306 T01307 T01400 T01402 T01404 T01406 T01408 T01410_1 T01411 T01412 T01413 T01415 T01416 T01420 T01421 T01448 T01451 T01451_1 T01451_2 T01451_3 T01452 T01453 T01454 T01459 T01461_1 T01463_1 T01463_2 T01464_2 T01465 T01466 T01470 T01471 T01478 T01499 T01500 T01503 T01504 T01505 T01507 T01507_1 T01508 T01509 T01512_1 T01514_1 T01514_3 T01515 T01517 T01517_1 T01517_2 T01518 T01519 T01524 T01524_1 T01524_3 T01525_1 T01525_2 T01525_4 T01526 T01527 T01528 T01529 T01533 T01534 T01535_4 T01535_5 T01538_1 T01539 T01540 *Continued on next page...*

Quantity	Vertices
$I \setminus C$ (*cont.*)	T01540_1 T01540_2 T01540_3 T01541 T01543 T01546 T01547 T01548 T01561 T01563 T01565 T01566 T01573 T01574 T01575 T01578 T01580 T01581 T01582 T01584 T01586 T01587 T01588 T01590_1 T01591 T01592 T01594_1 T01597 T01598 T01599 T01600 T01601 T01602 T01603 T01604_1 T01605 T01606 T01607 T01608 T01610_1 T01611 T01614 T01615 T01616 T01617_1 T01620 T01621 T01622 T01624 T01625 T01627 T01628 T01630 T01632 T01633 T01634 T01670 T01671 T01672 T01673 T01674 T01675 T01676 T01677 T01679 T01682 T01684 T01685 T01686 T01688 T01689 T01690 T01691 T01692 T01693 T01694 T01696 T01698 T01699 T01700 T01701 T01702 T01703 T01704 T01705 T01707 T01708 T01709 T01710 T01711 T01717 T01719 T01721 T01725 T01726 T01729 T01730 T01731 T01732 T01733 T01734 T01735 T01736 T01739 T01740 U0118 U0173 U0214 U0215 U0218

Quantity	Vertices
$I \setminus M$ Total: 383	AL AR CN_1 DS GV_1 HCI IAS IM IS NC_1 OS PD PL SDF SE SF SP T00001 T00019 T00022 T00086_2 T00089_1 T00096_1 T00101 T00105 T00116 T00117_2 T00123_2 T00125 T00148 T00153_3 T00156_1 T00167 T00193 T00211_3 T00211_5 T00212_3 T00216_2 T00217_2 T00230 T00250_1 T00272_1 T00328 T00339 T00402 T00482_2 T00482_3 T00489 T00508 T00582_1 T00593_1 T00594 T00598_1 T00664_2 T00671_1 T00671_2 T00674 T00674_3 T00711 T00733_1 T00739 T00743_1 T00758_1 T00793_2 T00795_1 T00796_2 T00797_1 T00832 T00860_1 T00883 T01027_1 T01028 T01031_1 T01032_1 T01056 T01057 T01058 T01060 T01070 T01088 T01096_1 T01099 T01103 T01104 T01106 T01107 T01108 T01109 T01113 T01163 T01167 T01168 T01171_1 T01174 T01215_1 T01256_1 T01267 T01268 T01271 T01286 T01287 T01288 T01290 T01291 T01292 T01293 T01294 T01296 T01296_1 T01297 T01298 T01299 T01300 T01302 T01304 T01305 T01306 T01307 T01400 T01402 T01404 T01406 T01408 T01410_1 T01411 T01412 T01413 T01415 T01416 T01420 T01421 T01448 T01451 T01451_1 T01451_2 T01451_3 T01452 T01453 T01454 T01455 T01457 T01459 T01461_1 T01463 T01463_1 T01463_2 T01464 T01464_2 T01465 T01466 T01470 T01471 T01478 T01497 T01499 T01500 T01503 T01504 T01505 T01507 T01507_1 T01508 T01509 T01512_1 T01514 T01514_1 T01514_3 T01515 T01517 T01517_1 T01517_2 T01518 T01519 T01524 T01524_1 T01524_3 T01525 T01525_1 T01525_2 T01525_4 T01526 T01527 T01528 T01529 T01533 T01534 T01535_4 T01535_5 *Continued on next page...*

Quantity	Vertices
$I \setminus M$ (*cont.*)	T01538 T01538_1 T01539 T01540 T01540_1 T01540_2 T01540_3 T01541 T01543 T01546 T01547 T01548 T01561 T01563 T01565 T01566 T01573 T01574 T01575 T01578 T01580 T01581 T01582 T01584 T01586 T01587 T01588 T01590_1 T01591 T01592 T01594_1 T01597 T01598 T01599 T01600 T01601 T01602 T01603 T01604_1 T01605 T01606 T01607 T01608 T01610_1 T01611 T01614 T01615 T01616 T01617_1 T01620 T01621 T01622 T01624 T01625 T01627 T01628 T01630 T01632 T01633 T01634 T01670 T01670_1 T01671 T01672 T01673 T01674 T01675 T01676 T01677 T01679 T01682 T01684 T01685 T01686 T01688 T01689 T01690 T01691 T01692 T01693 T01694 T01696 T01698 T01699 T01700 T01701 T01702 T01703 T01704 T01705 T01707 T01708 T01709 T01710 T01711 T01717 T01719 T01721 T01725 T01726 T01729 T01730 T01731 T01732 T01733 T01734 T01735 T01736 T01739 T01740 U0009 U0012 U0014_1 U0016 U0018 U0027 U0030 U0032 U0036 U0037 U0041_1 U0045 U0047 U0048_1 U0050 U0050_2 U0051 U0052_2 U0060 U0070 U0071 U0073_1 U0074_1 U0077 U0078 U0079 U0087 U0089 U0090 U0092 U0094_1 U0098 U0099_1 U0102 U0109 U0112 U0114 U0115 U0116 U0117 U0118 U0120_1 U0121_1 U0127_1 U0133_1 U0144 U0147_1 U0148 U0152_1 U0156 U0157 U0164_1 U0165 U0169_1 U0170 U0172 U0173 U0176 U0177 U0178 U0179 U0179_1 U0180 U0186 U0187 U0188 U0189 U0193 U0204 U0205 U0206 U0207 U0208 U0209 U0210 U0211_1 U0212_1 U0213 U0214 U0215 U0216 U0217 U0218 U0236 U0237

Quantity	Vertices
$M \setminus C$ Total: 21	COS110 COS121 COS132 COS151 COS212 COS216 COS222 COS226 COS301 COS330 COS332 COS333 ERA284 INF214 INL204 INL240 WTW115 WTW285 Year1 Year2 Year3

Quantity	**Vertices**
$C \setminus M$ Total: 112	AL AR CN_1 DS GV_1 HCI IAS IM IS NC_1 OS PD PL SDF SE SF SP T00125 T00489 T00598_1 T00674 T01287 T01290 T01455 T01457 T01463 T01464 T01497 T01514 T01525 T01538 T01670_1 U0009 U0012 U0014_1 U0016 U0018 U0027 U0030 U0032 U0036 U0037 U0041_1 U0045 U0047 U0048_1 U0050 U0050_2 U0051 U0052_2 U0060 U0070 U0071 U0073_1 U0074_1 U0077 U0078 U0079 U0087 U0089 U0090 U0092 U0094_1 U0098 U0099_1 U0102 U0109 U0112 U0114 U0115 U0116 U0117 U0120_1 U0121_1 U0127_1 U0133_1 U0144 U0147_1 U0148 U0152_1 U0156 U0157 U0164_1 U0165 U0169_1 U0170 U0172 U0176 U0177 U0178 U0179 U0179_1 U0180 U0186 U0187 U0188 U0189 U0193 U0204 U0205 U0206 U0207 U0208 U0209 U0210 U0211_1 U0212_1 U0213 U0216 U0217 U0236 U0237