UNIVERSITEIT • STELLENBOSCH • UNIVERSITY

Engineering process model: Detection of cycles and determination of paths

by

Mercia Cronje

Thesis presented in partial fulfilment of the requirements for the degree of

Master of Civil Engineering

at the University of Stellenbosch

Department of Civil Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Study leader: Dr G.C. Van Rooyen

April 2006

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: . . . .
M. Cronje

Date: . . . .

Abstract

Engineering process model: Detection of cycles and determination of paths

M. Cronje

Department of Civil Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Thesis: MScEng (Civil)

April 2006

In order to plan the engineering work of large construction projects efficiently, a model of the engineering process is required. An engineering process can be modelled by sets of persons, tasks, datasets and tools, as well as the relationships between the elements of these sets. Tasks are more often than not dependent on other tasks in the engineering process. In large projects these dependencies are not easily recognised, and if tasks are not executed in the correct sequence, costly delays may occur.

The homogeneous binary relation "has to be executed before" in the set of tasks can be used to determine the logical sequence of tasks algebraically. The relation can be described by a directed graph in the set of tasks, and the logical sequence of tasks can be determined by sorting the graph topologically, if the graph is acyclic. However, in an engineering process, this graph is not necessarily acyclic since certain tasks have to be executed in parallel, causing cycles in the graph. After generating the graph in the set of tasks, it is important to fuse all the cycles. This is achieved by finding the strongly connected components of the graph. The reduced graph, in which each strongly connected component is represented by a vertex, is a directed acyclic graph. The strongly connected components may be determined by different methods, including Kosaraju's, Tarjan's and Gabow's methods.

Considering the "has to be executed before" graph in the set of tasks, elementary paths through the graph, i.e. paths which do not contain any vertex more than once, are useful to investigate the influence of tasks on other tasks. For example, the longest elementary path of the graph is the logical critical path. The solution of such path problems in a network may be reduced to the solution of systems of equations using path algebras. The solution of the system of equations may be determined directly, i.e. through Gauss elimination, or iteratively, through Jacobi's or Gauss-Seidel's methods or the forward and back substitution method. The vertex sequence of an acyclic graph can be assigned in such a way that the coefficient matrix of the system of equations is reduced to staggered form, after which the solution is found by a simple back substitution. Since an engineering process has a start and an end, it is more acyclic than cyclic. Consequently we can usually reduce a substantial part of the coefficient matrix to staggered form. Using this technique, modifications of the solution methods mentioned above were implemented, and the efficiency of the technique is determined and compared between the various methods.

Uittreksel

Ingenieursproses model: Opsporing van siklusse en bepaling van paaie

M. Cronje

Departement van Siviele Ingenieurswese
Universiteit van Stellenbosch
Privaatsak X1, 7602 Matieland, Suid-Afrika

Tesis: MScIng (Siviel)

April 2006

'n Model van 'n ingenieursproses word benodig om die ingenieurswerk van groot konstruksie projekte effektief te beplan. 'n Ingenieursproses kan gemodelleer word deur versamelings van persone, take, datastelle en gereedskap, sowel as die verwantskappe tussen die elemente van die versamelings. Take is oor die algemeen afhanklik van ander take in die ingenieursproses. Hierdie afhanklikhede is nie altyd opsigtelik in groot projekte nie en duur vertragings kan ontstaan indien hierdie take nie in die regte volgorde uitgevoer word nie.

Die homogene binêre verwantskap "moet uitgevoer word voor", in die versameling van take, kan gebruik word om die logiese volgorde van take algebraïes te bepaal. Die verwantskap kan deur 'n gerigte grafiek op die versameling van take beskryf word. Die logiese volgorde van take kan dan bepaal word deur die grafiek topologies te sorteer, indien dit asiklies is. Die grafiek is egter nie noodwendig asiklies in 'n ingenieursproses nie, aangesien sommige take parallel uitgevoer moet word. Dit lei tot siklusse in die grafiek. Dit is belangrik om al die siklusse in die grafiek van die versameling van take te verwyder. Dit word vermag deur al die sterkverbinde komponente van die grafiek te vind. Die gereduseerde grafiek, waarin elkeen van die sterkverbinde komponente voorgestel word deur 'n nodus, is 'n gerigte asikliese grafiek. Die sterkverbinde komponente kan deur verskillende metodes, o.a. Kosaraju, Tarjan en Gabow se metodes bepaal word.

Elementêre paaie deur die grafiek, m.a.w. paaie waarin geen nodus meer as een keer voorkom nie, kan bepaal word deur gebruik te maak van die "moet uitgevoer word voor" grafiek op die versameling van take. Hierdie paaie is nuttig om die invloed van take op ander take te bestudeer. 'n Voorbeeld hiervan is die langste elementêre pad deur die grafiek, ook die sogenaamde logiese kritiese pad. Die oplossing van sulke pad-probleme in 'n netwerk kan vereenvoudig word tot die oplossing van 'n stelsel van vergelykings deur die gebruik van pad-algebras. Die oplossing van die stelsel van vergelykings kan direk bepaal word, deur byvoorbeeld Gauss eliminasie, of iteratief, deur Jacobi of Gauss-Seidel se metodes of die voorwaardse- en terugwaardse substitusie metode. Die opeenvolging van nodusse van 'n asikliese grafiek kan op so 'n wyse toegeken word dat die koëffisiënt matriks van die stelsel van vergelykings tot 'n trapsgewyse vorm vereenvoudig kan word. Daarna kan die oplossing gevind word deur 'n eenvoudige terugsubstitusie. Aangesien die ingenieursproses 'n begin en 'n einde het, is dit meer asiklies as siklies. Daaruit volg dat ons gewoonlik 'n aansienlike deel van die koëffisiënt matriks tot trapsgewyse vorm kan

vereenvoudig. Hierdie tegniek kan gebruik word om aanpassings aan die bogenoemde oplossingsmetodes aan te bring. Die effektiwiteit van die implementasies van hierdie aanpassings aan die verskeie metodes is bepaal en onderling met mekaar vergelyk.

Acknowledgements

I would like to thank the following persons without whom this thesis would not have been possible:

- My supervisor Dr G.C. Van Rooyen for his patience, maybe too much at times.
- Anton Eygelaar for his contribution to the path algebra implementation.
- M.J. Deacon for his contribution to additional software developed.
- The Southern African Institute of Steel Construction and Mittal Steel South Africa for financial support.
- D.N.J. Els and Stéfan van der Walt for their LaTeX and LyX University of Stellenbosch templates.
- Gerrit Greeff for the countless hours of companionship and emotional support.
- Pieter Mostert for his generous methods of motivation.
- Uli Huber for his years of friendship.
- My parents who have supported me without question throughout my studies.

Contents

Declaration **i**

Abstract **ii**

Uittreksel **iii**

Acknowledgements **v**

Contents **vi**

List of Figures **xiii**

List of Tables **xv**

List of Abbreviations **xvi**

List of Symbols **xvii**

1 Introduction **1**

1.1 The engineering process model . . . 1
1.2 The "has to be executed before" relation in the set of tasks . . . 2
1.3 Directed graph of the "has to be executed before" relation . . . 2
1.4 Logical sequence of tasks . . . 2
1.5 Elementary paths and the logical critical path . . . 3
1.6 Structure of the thesis . . . 3

2 Set theory and relations **5**

2.1 Introduction . . . 5
2.2 Set theory . . . 5
2.2.1 Definition of a set . . . 5
2.2.2 Formation of sets . . . 5
2.2.3 Quantifier . . . 6
2.2.4 Equal sets . . . 6
2.2.5 Subset . . . 6
2.2.6 Power set . . . 6
2.2.7 Family of elements . . . 6
2.3 Relations . . . 7
2.3.1 Ordered pair . . . 7

2.3.2 Cartesian product . . . 7
2.3.3 Unary relations . . . 7
2.3.4 Binary relations . . . 8
2.3.5 Heterogeneous binary relation . . . 8
2.3.6 Homogeneous binary relation . . . 8
2.3.7 Properties of relations . . . 8
2.3.8 Totality of a relation on A and B . . . 9
2.3.9 Uniqueness of a relation on A and B . . . 9
2.3.10 Relational diagram . . . 9
2.3.11 Types of relations . . . 10
2.3.11.1 Identity relation . . . 10
2.3.11.2 Inverse relation . . . 10
2.3.11.3 Composition . . . 10
2.3.11.4 Equivalence relation . . . 11
2.3.11.5 Equivalence class . . . 11
2.3.11.6 Partitioning by equivalence . . . 11
2.3.11.7 Quotient set . . . 11
2.3.12 Connection . . . 12
2.3.13 Closure . . . 12
2.3.13.1 Reflexive closure . . . 12
2.3.13.2 Symmetric closure . . . 13
2.3.13.3 Powers of a relation . . . 13
2.3.13.4 Stability index . . . 13
2.3.13.5 Transitive closure . . . 13
2.3.13.6 Reflexive transitive closure . . . 13
2.3.13.7 Reflexive symmetric transitive closure . . . 14
2.3.14 Algebra of homogeneous binary relations . . . 14
2.3.14.1 Graphical representation . . . 14
2.3.14.2 Special relations . . . 15
2.3.14.3 Equality and inclusion . . . 15
2.3.14.4 Binary operations . . . 15

3 Directed graphs **16**
3.1 Introduction . . . 16
3.2 Directed graphs . . . 17
3.2.1 Definition . . . 17
3.2.2 Properties . . . 17
3.2.3 Equality and inclusion . . . 18
3.2.4 Adjacency-matrix graph representation . . . 18
3.3 Mappings . . . 19
3.3.1 Mapping notation . . . 19
3.3.2 Image of an element . . . 19
3.3.3 Arrow diagram . . . 19
3.3.4 Types of mappings . . . 19
3.3.4.1 Injective mapping . . . 19
3.3.4.2 Surjective mapping . . . 20

3.3.4.3 Bijective mapping . . . 20
3.3.4.4 Canonical mapping . . . 20
3.4 Structure of graphs . . . 21
3.4.1 Paths and cycles in directed graphs . . . 21
3.4.1.1 Predecessor and successor . . . 21
3.4.1.2 Indegree and outdegree . . . 21
3.4.1.3 Edge sequence . . . 22
3.4.1.4 Ancestors and descendents . . . 22
3.4.1.5 Path . . . 23
3.4.1.6 Cycle . . . 23
3.4.1.7 Acyclic graph . . . 23
3.4.1.8 Anticyclic graph . . . 23
3.4.1.9 Cyclic graph . . . 23
3.4.1.10 Properties . . . 23
3.4.1.11 Simple path . . . 24
3.4.1.12 Simple cycle . . . 24
3.4.1.13 Elementary path . . . 24
3.4.1.14 Elementary cycle . . . 24
3.4.2 Connectedness of directed graphs . . . 25
3.4.2.1 Reachability . . . 25
3.4.2.2 Strong connectedness . . . 25
3.4.2.3 Unilateral connectedness . . . 25
3.4.2.4 Weak connectedness . . . 25
3.4.2.5 Connectedness relations . . . 25
3.4.2.6 Properties of the connectedness relations . . . 26
3.4.2.7 Decomposition into connected components . . . 26
3.4.2.8 Decomposition into strongly connected components . . . 27
3.4.2.9 Decomposition into weakly connected components . . . 27
3.4.2.10 Strongly connected components example . . . 27
3.4.3 Acyclic graphs . . . 29
3.4.3.1 Directed acyclic graph . . . 29
3.4.3.2 Rank . . . 29
3.4.3.3 Topological Sorting . . . 30
3.4.3.4 Order structure . . . 30
3.4.3.5 Basic edges and chords . . . 31
3.4.3.6 Basic path . . . 31
3.4.3.7 Basic graph . . . 31
3.4.3.8 Order diagram . . . 31

4 Strongly connected components and the logical sequence of tasks 32
4.1 Introduction . . . 32
4.2 Rooted graphs and rooted trees . . . 33
4.2.1 Introduction . . . 33
4.2.2 Root . . . 33
4.2.3 Rooted graph . . . 33
4.2.4 Acyclic rooted graph . . . 33

4.2.5 Rooted tree . . . 33
4.2.6 Forest of rooted trees . . . 34
4.2.7 Search tree . . . 34
4.3 Depth-first search . . . 34
4.3.1 Trees and forests . . . 34
4.3.2 Pre- and post-order numbering . . . 34
4.3.3 Classification of edges . . . 34
4.3.4 Depth-first search algorithm . . . 36
4.3.5 Depth-first search example . . . 36
4.4 Decomposition into strongly connected components . . . 38
4.4.1 Kosaraju's algorithm . . . 38
4.4.1.1 Description . . . 38
4.4.1.2 Implementation . . . 39
4.4.1.3 Example . . . 39
4.4.2 Tarjan's algorithm . . . 40
4.4.2.1 Description . . . 40
4.4.2.2 Example . . . 41
4.4.3 Gabow's algorithm . . . 41
4.4.3.1 Description . . . 41
4.4.3.2 Example . . . 42
4.5 Logical sequence of tasks through topological sorting . . . 42
4.5.1 Topologically sorting a directed acyclic graph by removing sources . . . 42
4.5.1.1 Algorithm . . . 42
4.5.1.2 Topological sorting example . . . 43
4.5.2 Graphical representation of the logical sequence of tasks . . . 44

5 Path algebras and methods of solution 45
5.1 Introduction . . . 45
5.2 Path algebras . . . 46
5.2.1 Network . . . 46
5.2.2 Path problem . . . 46
5.2.3 Path and weights . . . 46
5.2.3.1 Alphabet and words . . . 46
5.2.3.2 Edge and path labels . . . 47
5.2.4 Path set and weighted path set . . . 47
5.2.4.1 Path sets . . . 47
5.2.4.2 Weighted path set . . . 47
5.2.5 Elementary path set matrix . . . 48
5.2.6 Elementary path weight matrix . . . 48
5.2.7 Operations in the path set . . . 48
5.2.8 Algebraic structure . . . 49
5.2.8.1 Path sets . . . 49
5.2.8.2 Weighted path sets . . . 49
5.2.9 Operations . . . 50
5.2.9.1 Path set matrices . . . 50
5.2.9.2 Weight matrices . . . 50

5.2.10 Algebraic structure . . . 50
5.2.10.1 Path set matrices . . . 50
5.2.10.2 Weight matrix . . . 51
5.2.11 Closure . . . 51
5.2.11.1 Elementary path set matrix . . . 51
5.2.11.2 Elementary weight matrix . . . 51
5.2.12 Path algebra . . . 52
5.2.12.1 System of equations for path sets . . . 52
5.2.12.2 System of equations for weights . . . 52
5.3 Literal path algebra . . . 53
5.3.1 Introduction . . . 53
5.3.2 Literal vertex labels . . . 53
5.3.3 Elementary paths . . . 53
5.3.3.1 Problem . . . 53
5.3.3.2 Weights . . . 54
5.3.3.3 Operations . . . 54
5.3.3.4 Weight matrices . . . 54
5.3.4 Extreme elementary paths . . . 54
5.3.4.1 Problem . . . 54
5.3.4.2 Weights . . . 55
5.3.4.3 Operations . . . 55
5.3.4.4 Weight matrices . . . 55
5.3.5 Properties of elementary path algebra . . . 55
5.3.5.1 Powers of an element . . . 55
5.3.5.2 Closure of an element . . . 56
5.3.5.3 Stability . . . 56
5.3.6 Elementary paths example . . . 56
5.4 Logical critical path . . . 57
5.5 Systems of equations . . . 57
5.5.1 Solution of systems of equations . . . 57
5.5.1.1 Introduction . . . 57
5.5.1.2 Solutions . . . 57
5.5.1.3 Staggered system of equations . . . 58
5.5.1.4 Equivalent systems of equations . . . 58
5.5.2 Direct methods of solution . . . 58
5.5.2.1 Introduction . . . 58
5.5.2.2 Forward substitution . . . 58
5.5.2.3 Back substitution . . . 58
5.5.2.4 Elimination . . . 59
5.5.2.5 Gaussian elimination method . . . 59
5.5.3 Iterative Methods of Solution . . . 60
5.5.3.1 Introduction . . . 60
5.5.3.2 General iteration . . . 60
5.5.3.3 Conditions . . . 60
5.5.3.4 Jacobi method . . . 61

5.5.3.5 Gauss-Seidel method . . . 61
5.5.3.6 Forward and back substitution method . . . 62
5.5.3.7 Number of iterations . . . 62
5.6 Relabelling of vertices . . . 62
5.6.1 Relabelling example . . . 63

6 Implementation of computer model for graphs and performance testing 65
6.1 Introduction . . . 65
6.2 Computer models for graphs . . . 65
6.2.1 Data structures . . . 65
6.2.1.1 Adjacency-matrix representation . . . 65
6.2.1.2 Adjacency-lists representation . . . 65
6.2.1.3 Alternative representation . . . 66
6.2.1.4 Example . . . 66
6.2.2 Graph generator . . . 67
6.2.2.1 Algorithm . . . 67
6.2.2.2 Example . . . 68
6.3 Unified Modelling Language view of implementation . . . 68
6.3.1 Introduction . . . 68
6.3.2 Class diagrams . . . 69
6.4 UML view of graph model . . . 70
6.4.1 Basic graph implementation . . . 70
6.5 Performance testing of solution methods . . . 70
6.5.1 Gauss elimination calculations . . . 71
6.5.1.1 Gauss elimination . . . 71
6.5.1.2 Back substitution . . . 71
6.5.1.3 Generalized . . . 72
6.5.2 Jacobi calculations . . . 72
6.5.2.1 Generalized . . . 73
6.5.3 Jacobi calculations after sorting . . . 73
6.5.3.1 Generalized . . . 73
6.5.4 Gauss-Seidel calculations . . . 73
6.5.4.1 Generalized . . . 74
6.5.5 Gauss-Seidel calculations after sorting . . . 74
6.5.5.1 Generalized . . . 74
6.5.6 Forward and back substitution calculations . . . 74
6.5.6.1 Generalized . . . 74
6.5.7 Forward and back substitution calculations after sorting . . . 75
6.5.7.1 Generalized . . . 75
6.5.8 Interpretation of results . . . 75
6.5.8.1 Number of iterations . . . 75
6.5.8.2 Influence of sorting on the number of iterations . . . 75
6.5.8.3 Number of calculations . . . 77
6.5.8.4 Influence of sorting on the number of calculations . . . 78
6.5.8.5 Duration . . . 79

7 Conclusions **82**

Bibliography **84**

A Elementary paths - example from Section 5.3.6 **A–1**

B UML implementation of graph model **B–1**

B.1 Graph B–1
B.2 Vertex B–2
B.3 Edge B–3
B.4 SuperVertex B–4
B.5 SuperEdge B–5
B.6 Equation B–5
B.7 ElementaryPath B–6
B.8 ElementaryPathSet B–6
B.9 ElementaryPathAlgebra B–7

C Useful algebraic equations **C–1**

D Test graph data **D–1**

D.1 Iterations D–1
D.2 Calculations D–8
D.3 Durations D–15

List of Figures

1.1 Binary relations . . . 1
1.2 The “has to be executed before” relationship . . . 2
1.3 Undirected edge in the directed graph . . . 2
1.4 Cycle in the directed graph . . . 3

2.1 Uniqueness of R . . . 10
2.2 Quotient set . . . 12
2.3 Homogeneous binary relations graph example . . . 14

3.1 Directed graph properties . . . 18
3.2 Canonical mapping . . . 21
3.3 Strongly connected components graph example . . . 28
3.4 Reduced graph . . . 29
3.5 Strongly connected components . . . 29

4.1 Edge classifications . . . 35
4.2 Graph example . . . 36
4.3 Depth-first search forest . . . 37
4.4 Depth-first search forest for alternative depth-first search . . . 38
4.5 Inverse of graph example . . . 39
4.6 Depth-first search forest of inverse graph . . . 39
4.7 Depth-first search forest of graph . . . 40
4.8 Strongly connected components . . . 40
4.9 Main and secondary vertex sequences . . . 41
4.10 Topological sorting graph example . . . 43
4.11 Graph, rearranged after topological sorting . . . 43
4.12 Graphical representation of the sequence of tasks . . . 44

5.1 Elementary paths example graph . . . 56
5.2 Elementary paths example sequence of tasks . . . 57
5.3 Triangular matrices . . . 58
5.4 Adjacency-matrix after relabelling . . . 63
5.5 Graph before relabelling . . . 63
5.6 Topologically sorted graph . . . 63
5.7 Graph after relabelling . . . 64

6.1 Graph representation . . . 66

6.2 Elements . . . 68
6.3 Random graph . . . 68
6.4 Class diagram basics . . . 69
6.5 Class diagram associations . . . 69
6.6 Class diagram inheritance . . . 70
6.7 UML diagram of basic Graph implementation . . . 70
6.8 Number of iterations for unsorted graphs . . . 76
6.9 Number of iterations for sorted graphs . . . 76
6.10 Difference in number of iterations before and after sorting . . . 77
6.11 Number of calculations for unsorted graphs . . . 78
6.12 Number of calculations for sorted graphs . . . 78
6.13 Difference in number of calculations before and after sorting . . . 79
6.14 Unsorted durations . . . 80
6.15 Sorted durations . . . 81
6.16 Differences in duration between unsorted and sorted . . . 81

List of Tables

2.1 Special unary relations . . . 7
2.2 Properties of relations . . . 9

3.1 Properties of directed graphs . . . 17
3.2 Strongly connected components . . . 29

4.1 Pre-order and post-order numbers . . . 37
4.2 Pre-order and post-order numbers for alternative depth-first search . . . 37
4.3 Post-order numbers . . . 39
4.4 Pre-order and low numbers . . . 41

5.1 Algebraic structure of path sets . . . 49
5.2 Algebraic structure of weighted path sets . . . 50
5.3 Algebraic structure of path set matrices . . . 51

List of Abbreviations

DFS	Depth-first search
SCC	Strongly connected component
UML	Unified Modelling Language

List of Symbols

$\subseteq$	Contained in
$\supseteq$	Includes
$\times$	Cartesian product
$\sim$	Equivalence relation
$\circ$	Composition
$\sqcup$	Union
$\sqcap$	Intersection
λ	Empty word
ϕ	Empty set
$\in$	Element of
Φ	Mapping
$\mathbb{A}$	Alphabet
e	All relation
$\mathbf{x}$	Solution vector
$\mathbf{A}$	Coefficient matrix
A^*	Reflexive transitive closure
E	Equivalence relation, All relation
G	Graph
I	Identity relation
M	Set
M/E	Canonical mapping
P	Power set
R	Relation, Edge set
R^*	Reflexive transitive closure
R^+	Transitive closure
S	Strong connectedness relation
V	Vertex set

W	Complete path set
Z	Weight set

Chapter 1

Introduction

1.1 The engineering process model

In order to plan the engineering work of large construction projects efficiently, a model of the engineering process is required (see references [1, 2]). An engineering process can be modelled by sets (see Section 2.2.1) of persons, tasks, data-sets and tools, as well as the relationships between the elements of these sets. There may be relationships between elements of different sets, heterogeneous binary relationships (see Section 2.3.5) or between elements of the same set, homogeneous binary relationships (see Section 2.3.6).

Twelve types of heterogeneous binary relations are possible on the basis of the four sets, as shown in Figure 1.1. The relation "access" includes relations such as "creates", "reads" and "modifies", similarly for the relation "is accessed by". The complete range of binary relations, the twelve types of heterogeneous relations as well as the four types of homogeneous relations, can be determined on the basis of three types of heterogeneous binary relations. These are shown in grey in Figure 1.1. Therefore, these are the only relations that need to be specified along with the four sets.

	persons	tasks	data-sets	tools
persons		executes	access	use
tasks	is executed by		access	requires
data-sets	is accessed by	is accessed by		can be edited by
tools	is used by	is required by	can edit	

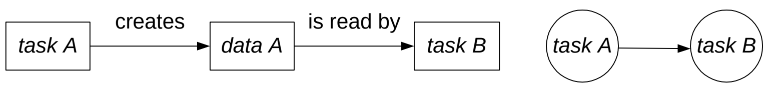
Figure 1.1: Binary relations

The remaining binary relations can be determined by either finding the inverse (see Section 2.3.11.2) of a specified relation, the composition (see Section 2.3.11.3) of more than one of the specified relations or by a combination of both operations.

1.2 The "has to be executed before" relation in the set of tasks

Tasks are more often than not dependent on other tasks in the engineering process. In large projects these dependencies are not easily recognised, and if tasks are not executed in the correct sequence, costly delays may occur.

The homogeneous binary relation "has to be executed before" in the set of tasks can be determined, given the heterogeneous binary relations "access" and "is accessed by" between the sets of tasks and data-sets. As can be seen in Figure 1.2, $task\,A$ has to be executed before $task\,B$, since $data\,A$, which is read by $task\,B$, has to be created first by $task\,A$.



⇒ $task\,A$ has to be executed before $task\,B$

Figure 1.2: The "has to be executed before" relationship

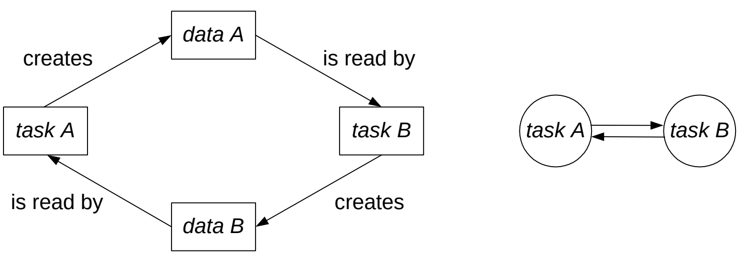
The homogeneous binary relation "has to be executed before" can be described by a directed graph in the set of tasks (see Section 3.2).

1.3 Directed graph of the "has to be executed before" relation

The homogeneous binary relation "has to be executed before" in the set of tasks can be used to determine the logical sequence of tasks (see Section 4.5) algebraically. The logical sequence of tasks can be determined by sorting the graph topologically (see Section 3.4.3.3), if the graph is acyclic (see Section 3.4.3).

1.4 Logical sequence of tasks

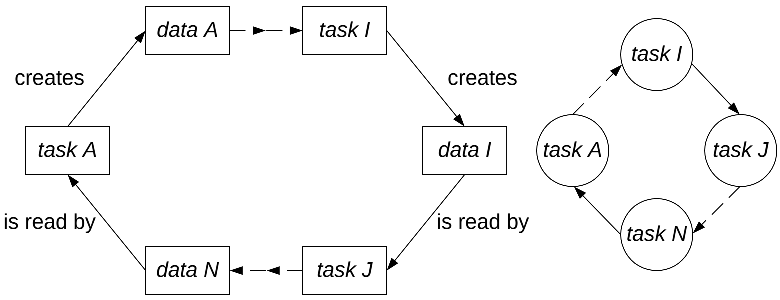
In an engineering process, the task-task graph is not necessarily acyclic since certain tasks have to be executed in parallel, causing cycles (see Section 3.4.1.6) in the graph.



⇒ $task\,A$ and $task\,B$ should be performed in parallel

Figure 1.3: Undirected edge in the directed graph

The creation of an undirected edge (i.e. a relationship in both directions between two tasks) in the graph is shown in Figure 1.3, while the creation of a cycle is shown in Figure 1.4. After generating the graph in the set of tasks, it is important to fuse all the cycles. This is achieved by finding the strongly connected components (see Section 3.4.2.8) of the graph. The reduced graph, in which each strongly connected component is represented by a vertex, is a directed acyclic graph. All the tasks in a strongly connected component have to be executed in parallel. The strongly connected components may be determined by different methods, including Kosaraju's (see Section 4.4.1), Tarjan's (see Section 4.4.2) and Gabow's (see Section 4.4.3) method.



$\Rightarrow$ *task A*, ..., *task I*, *task J*, ... and *task N* should be performed in parallel

Figure 1.4: Cycle in the directed graph

1.5 Elementary paths and the logical critical path

Considering the "has to be executed before" graph in the set of tasks, elementary paths (see Section 1.1) through the graph are useful to investigate the influence of tasks on other tasks. For example, the longest elementary path is the logical critical path (see Section 5.4). The solution of such path problems in a network may be reduced to the solution of systems of equations (see Section 1.1) using path algebras (see Section 5.3.3). The solution of the system of equations may be determined directly, i.e. through Gauss elimination (see Section 5.5.2.5), or iteratively, through Jacobi's (see Section 5.5.3.4) or Gauss-Seidel's (see Section 5.5.3.5) method or through the forward and back substitution method (see Section 5.5.3.6). The vertex sequence of an acyclic graph can be assigned in such a way that the coefficient matrix of the system of equations is reduced to staggered form, after which the solution is found by a simple backward sweep (see Section 5.5.2.3). Since an engineering process has a start and an end, it is more acyclic than cyclic. Consequently a substantial part of the coefficient matrix can be reduced to staggered form (see Section 5.6). Using this technique, modifications of the solution methods mentioned above were implemented, and the efficiency of the technique is determined and compared between the various methods (see Section 6.5).

1.6 Structure of the thesis

The engineering process model consists of sets of elements, as well as the relationships between the elements of these sets. Set theory and relations will be discussed in detail in Chapter 2. This theory will be applied to the set of tasks of the engineering process model, as well as the "has to be executed before" relation in the set of tasks.

The "has to be executed before" relation in the set of tasks can be described by a directed graph. The set of tasks is equipped with structure by the "has to be executed before" relation. Directed graphs, as well as the structural properties of directed graphs will be discussed in detail in Chapter 3.

The logical sequence of tasks can be determined by sorting the task-task graph topologically. However, only acyclic graphs can be sorted topologically. Therefore, the task-task graph has to be reduced to an acyclic graph, if it contains cycles. This is achieved by decomposing the graph into its strongly connected components. A graph can be decomposed into its strongly connected components using the algorithms of Kosaraju, Tarjan or Gabow. After this has been done, the reduced acyclic graph can be sorted topologically to determine the logical sequence of tasks. The decomposition of a graph into its strongly connected components, as well as determination of the logical sequence of tasks are discussed in detail in Chapter 4.

Elementary paths, most importantly of which is the logical critical path, through the task-task graph can be determined using the algebra of elementary paths. The use of the elementary path algebra reduces this problem to a system of equations. This system of equations can be solved by direct methods, such as Gauss elimination, followed by a back substitution, or through iterative methods, such as Jacobi's, Gauss-Seidel's or the forward and back substitution method. The elementary path algebra, as well as methods of solution of the systems of equations are discussed in detail in Chapter 5.

A computer model is developed and implemented for the graphs and graph algorithms. This is discussed in Chapter 6. The performance of the implementation of the methods of solution is also considered in this chapter.

Chapter 2

Set theory and relations

2.1 Introduction

A collection of the task elements of an engineering process as discussed in Section 1.1 is a set. Therefore, set theory is considered in Section 2.2.

There may be relationships between the elements of sets, such as the “has to be executed before” relation in the set of tasks as discussed in Section 1.2. The relevance of the properties of relations, as well as the different types of relations, discussed in Section 2.3, will become apparent when we look at the graph representation of the “has to be executed before” relation in Chapter 3. See reference [4] for a detailed discussion of set theory and relations.

2.2 Set theory

2.2.1 Definition of a set

Objects which are separable and can be identified uniquely are called elements. A collection of elements with similar properties is called a set. Each property of an element is described either by its value or by rules for determining its value. The elements of a set are uniquely identified using a property of the elements which takes different values for all elements. This property is called the name (label, identifier) of the element.

2.2.2 Formation of sets

A set M is specified either by enumerating the names of the elements or by describing the properties of the elements. The order of enumeration of the elements is irrelevant. If two elements in the enumeration bear the same designation, they represent the same element. This element is contained in the set only once. The set without elements is called the empty set and is designated by ϕ.

$$\begin{array}{rcll} M & = & \{a, b, c\} & \text{set } M \text{ consists of the elements } a, b, c \\ M & = & \{x \mid E(x)\} & \text{set } M \text{ contains every element for which the logical expression } E(x) \text{ is true} \\ \phi & := & \{x \mid x \neq x\} & \text{empty set} \end{array} \tag{2.1}$$

The membership of an element a in a set M is represented using the symbols $\in$ and $\notin$:

$$\begin{array}{ll} a \in M & a \text{ is an element of } M \\ a \notin M & a \text{ is not an element of } M \end{array}$$

2.2.3 Quantifier

There are statements which are true for certain elements of a set M and false for other elements of M.

$$\begin{array}{ll} \bigwedge\limits_{x \in M} a(x) & \text{for every } x \text{ in the set } M,\ a(x) \text{ holds} \\ \bigvee\limits_{x \in M} a(x) & \text{there is an } x \text{ in the set } M \text{ for which } a(x) \text{ holds} \end{array} \tag{2.2}$$

2.2.4 Equal sets

Two sets A and B are said to be equal if they contain the same elements. If the sets A and B are equal, they contain the same elements. The statement $A = B$ (A equals B) can either be true or false.

$$(A = B) :\Leftrightarrow \bigwedge_x (x \in A \Leftrightarrow x \in B) \tag{2.3}$$

$$\begin{array}{ll} A = B & \text{sets } A \text{ and } B \text{ are equal} \\ A \neq B & \text{sets } A \text{ and } B \text{ are not equal} \end{array}$$

2.2.5 Subset

A set A is called a subset of a set B if every element of A is also an element of B. If the set B contains at least one element not contained in A, then A is called a proper subset of B.

$$\begin{array}{lll} (A \subseteq B) & :\Leftrightarrow & \bigwedge\limits_x (x \in A \Rightarrow x \in B) \\ (A \subset B) & :\Leftrightarrow & (A \subseteq B) \wedge \neg (A = B) \end{array} \tag{2.4}$$

In addition to the symbols $\subseteq$ (contained in) and $\subset$ (properly contained in), the symbols $\supseteq$ (includes) and $\supset$ (properly includes) are also used.

$$\begin{array}{llll} B \supseteq A & \text{set } B \text{ includes set } A & A \subseteq B & A \text{ is a subset of } B \\ B \supset A & \text{set } B \text{ properly includes set } A & A \subset B & A \text{ is a proper subset of } B \end{array}$$

2.2.6 Power set

From a given set M of n elements, 2^n subsets can be formed, including ϕ and M. The set of all subsets of M, including ϕ and M, is called the power set of M and is designated by $P(M)$. The set M is called the reference set of the power set $P(M)$.

$$\begin{array}{l} M = \{a, b, c\} \qquad n = 3, \quad 2^3 = 8 \\ P(M) = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\} \end{array} \tag{2.5}$$

2.2.7 Family of elements

Designating the elements of a set by different names is inconvenient for sets with a large number of elements. The elements of a set X are therefore often designated by $x_1, x_2, x_3, \ldots$. The common designation by the lowercase letter x symbolizes membership in the set X, while the index $i \in \{1, 2, 3, \ldots\}$ identifies the element. The elements x_i are called a family of elements. The family of elements is designated by $\{x_i\}$.

$$X = \{x_i \mid i \in I = \{1, 2, 3, \ldots\}\} \tag{2.6}$$

2.3 Relations

2.3.1 Ordered pair

In a set, the order of elements is irrelevant, so that $\{a, b\} = \{b, a\}$. Two elements a and b whose order is relevant are called an ordered pair. An ordered pair is enclosed in parentheses. The elements a and b may be contained in different sets. Two ordered pairs (a, b) and (c, d) are equal if and only if $a = c$ and $b = d$.

$$\begin{array}{ll} \text{ordered pair} & (a,b) := \{\{a\}, \{a,b\}\} \\ a & \text{first component of the ordered pair } (a,b) \\ b & \text{second component of the ordered pair } (a,b) \end{array} \tag{2.7}$$

2.3.2 Cartesian product

Let the sets A and B be given. The set of all ordered pairs (a, b) that can be formed using elements $a \in A$ and $b \in B$ is called the cartesian product (direct product) of the sets A and B. The cartesian product is designated by $A \times B$ (A times B).

$$A \times B := \{(a, b) \mid a \in A \wedge b \in B\} \tag{2.8}$$

2.3.3 Unary relations

A unary relation is a subset of a set. Let a non-empty set M of elements and a unary operation on these elements be given. The value of the unary operation Ra for an element a is true or false.

$$u := \{a \in M \mid Ra\} \subseteq M \tag{2.9}$$

$$u \subseteq M$$

The empty relation ϕ and the universal relation $e = M$ are special unary relations in the set M. They are also called the null relation and the all (complete, total) relation. A unary relation with exactly one element $x \in M$ is called a point relation or a point. These are shown in Table 2.1.

Table 2.1: Special unary relations

null relation	ϕ	:=	$\{\}$
point relation	x	:=	$\{x\}$
all relation	e	:=	M

2.3.4 Binary relations

A relation on two sets is called a binary relation. A binary relation is a set of ordered pairs of elements. It is a subset of the cartesian product of two sets. A relation on two sets, or a heterogeneous binary relation, is a subset of the cartesian product of two different sets. A relation in a set, or a homogeneous binary relation, is a subset of the cartesian product, where the two factors of the product are equal.

2.3.5 Heterogeneous binary relation

Let two non-empty sets A and B be given, with a binary operation for a relation R on the elements $a \in A$ and $b \in B$ whose value is a logical constant. The value of the operation for the ordered pair (a, b) in the product $A \times B$ is designated by aRb (a is related to b) and is either true or false.

The subset R of pairs (a, b) for which aRb is true is called a relation on A and B, or a heterogeneous binary relation. Thus the relation is a set containing the ordered pairs of elements for which the relationship specified by the operation holds. The order of the elements a and b in the operation is relevant to the result of the operation. The relation R is a subset of the heterogeneous cartesian product $A \times B$.

$$R := \{(a, b) \in A \times B \mid aRb\} \subseteq A \times B \tag{2.10}$$

2.3.6 Homogeneous binary relation

Let a non-empty set M of elements be given, with a binary operation for a relation R on the elements $a \in M$ and $b \in M$ whose value is a logical constant. The value of the operation for the ordered pair (a, b) in the product $A \times A$ is designated by aRb and is either true or false.

The subset R of pairs (a, b) for which aRb is true is called a relation in M, or a homogeneous binary relation. Thus the relation is a set containing pairs of elements for which the relationship specified by the operation holds. The corresponding homogeneous relation is the set of all ordered pairs (a, b) for which the binary operation aRb is true. It is a subset of the homogeneous cartesian product $M \times M$.

$$R := \{(a, b) \in M \times M \mid aRb\} \subseteq M \times M \tag{2.11}$$

2.3.7 Properties of relations

The subset $R \subseteq M \times M$ of the cartesian product of a set with itself for which aRb is true is called a relation in M. The relationships between the statement values aRb and bRa of the pairs (a, b) and (b, a) determine the properties of the relation. These properties are defined in Tabel 2.2 for $a, b, c \in M$.

$$R := \{(a, b) \in M \times M \mid aRb\}$$

Table 2.2: Properties of relations

$$
\begin{aligned}
&R \text{ is reflexive} &&:\Leftrightarrow \bigwedge_{a} (aRa)\\
&R \text{ is antireflexive} &&:\Leftrightarrow \textstyle\bigwedge_{a} (\neg aRa)\\
&R \text{ is symmetric} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b} (aRb \Rightarrow bRa)\\
&R \text{ is asymmetric} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b} (aRb \Rightarrow \neg bRa)\\
&R \text{ is antisymmetric} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b} (aRb \wedge bRa \Rightarrow a = b)\\
&R \text{ is linear} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b} (aRb \vee bRa)\\
&R \text{ is connex} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b} (a \neq b \Rightarrow aRb \vee bRa)\\
&R \text{ is transitive} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b}\bigwedge_{c} (aRb \wedge bRc \Rightarrow aRc)
\end{aligned}
$$

2.3.8 Totality of a relation on A and B

The subset $R \subseteq A \times B$ for which aRb is true is a relation on the sets A and B. The subset of A for which there exists $b \in B$ such that aRb is true is called the domain of R. The subset of B for which there exists $a \in A$ such that aRb is true is called the range of R. The relation is said to be left-total if its domain is A. The relation is said to be right-total if its range is B. A relation which is left- and right-total is said to be bitotal.

$$
\begin{aligned}
&R \text{ is left-total} &&:\Leftrightarrow \bigwedge_{a}\bigvee_{b} (aRb)\\
&R \text{ is right-total} &&:\Leftrightarrow \bigwedge_{b}\bigvee_{a} (aRb)\\
&R \text{ is bitotal} &&:\Leftrightarrow R \text{ is left-total } \wedge R \text{ is right-total}
\end{aligned}
\tag{2.12}
$$

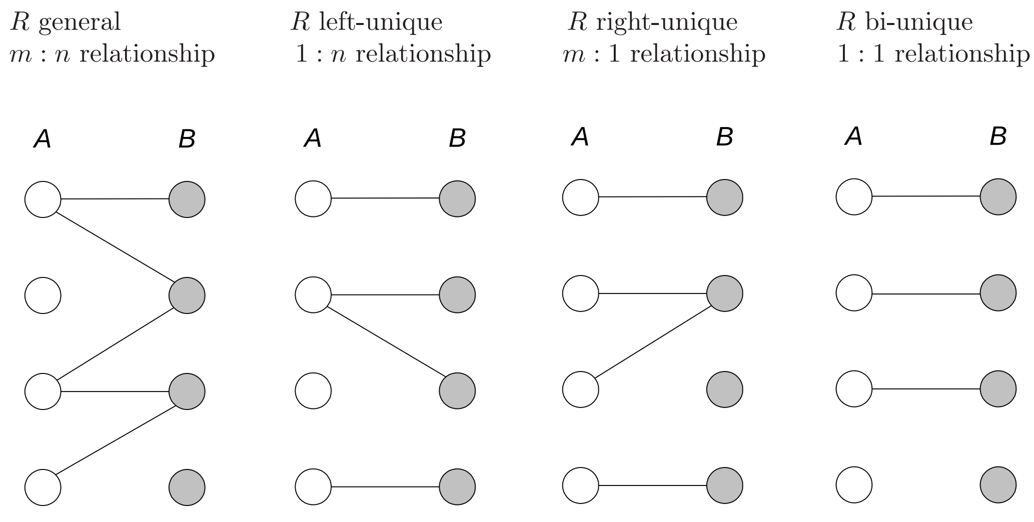
2.3.9 Uniqueness of a relation on A and B

A relation on A and B is said to be left-unique if the statements aRb and cRb are true only for $a = c$. The relation is said to be right-unique if the statements aRb and aRc are true only for $b = c$. A relation which is left-unique and right-unique is said to be bi-unique.

$$
\begin{aligned}
&R \text{ is left-unique} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b}\bigwedge_{c} (aRb \wedge cRb \Rightarrow a = c)\\
&R \text{ is right-unique} &&:\Leftrightarrow \bigwedge_{a}\bigwedge_{b}\bigwedge_{c} (aRb \wedge aRc \Rightarrow b = c)\\
&R \text{ is bi-unique} &&:\Leftrightarrow R \text{ is left-unique } \wedge R \text{ is right-unique}
\end{aligned}
\tag{2.13}
$$

2.3.10 Relational diagram

A relational diagram shows three sets: the sets A and B as well as the relation R. The elements of A and B are represented by different symbols, for instance empty and filled circles. The elements of R are represented by line segments. For $R \subseteq A \times B$ the elements $a \in A$ and $b \in B$ for which aRb is true are joined by line segments. The following relational diagrams illustrate the uniqueness of R.



(a) general (b) left-unique (c) right-unique (d) bi-unique

Figure 2.1: Uniqueness of R

2.3.11 Types of relations

Every relation is a subset of a direct product. Relations often have additional properties. Relations with common properties belong to a type of relations. Some types of relations are defined in the following.

2.3.11.1 Identity relation

The set of all ordered pairs (a, a) in the product $A \times A$ is called the identity relation I_A in the set A.

$$I_A := \{(a, a) \mid a \in A\} \tag{2.14}$$

2.3.11.2 Inverse relation

The set R^{-1} is called the inverse (dual) relation of the relation R if the order of the elements in the ordered pairs (a, b) of R is exchanged in R^{-1}.

$$R^{-1} := \{(b, a) \mid (a, b) \in R\} \tag{2.15}$$

2.3.11.3 Composition

Let a relation R on the sets A and B and a relation S on the sets B and C be given. The set of ordered pairs $(a, c) \in A \times C$ for which there is a common element in B is called the composition of R and S. The order of R and S is relevant, as b is the second element of R and the first element of S. The composition is designated by $R \circ S$.

$$R \circ S := \left\{ (a, c) \in A \times C \mid \bigvee_{b \in B} (aRb \wedge bSc) \right\} \tag{2.16}$$

2.3.11.4 Equivalence relation

A relation $E \subseteq M \times M$ is called an equivalence relation in the set M if it is reflexive, symmetric and transitive. The elements x and y of the set M are said to be equivalent if the set E contains the pair (x, y); this relationship is designated by $x \sim y$ or xEy.

$$\begin{array}{lllllll} E \text{ is reflexive} & x \sim x & & & & & \\ E \text{ is symmetric} & x \sim y & \Rightarrow & y \sim x & & & \\ E \text{ is transitive} & x \sim y & \wedge & y \sim z & \Rightarrow & x \sim z & \end{array} \tag{2.17}$$

2.3.11.5 Equivalence class

A subset of a set M is called an equivalence class in M if the elements of the subset are pairwise equivalent. An equivalence class is designated by choosing an arbitrary element a of the class and enclosing it in square brackets $[a]$. The selected element a is called a representative of its class.

$$[a] := \{x \in M \mid (a, x) \in E\} \tag{2.18}$$

2.3.11.6 Partitioning by equivalence

The equivalence classes in a set M for a given equivalence relation E form a partition of M:

1. Every element x of the set M is contained in at least one equivalence class, since (x, x) is an element of the reflexive relation E.
2. None of the equivalence classes $[x]$ is empty, since $(x, x) \in E$ and hence at least x itself is an element of $[x]$.
3. Every element z of the set M is contained in exactly one equivalence class. In fact, if z is an element of the classes $[x]$ and $[y]$, then since E is symmetric and transitive z x and z y imply x z and x y; hence $[x] = [y]$.

2.3.11.7 Quotient set

The set of equivalence classes of a set M for an equivalence relation E is called a quotient set and is designated by M/E (M modulo E). A subset $R \subseteq M$ is called a system of representatives of the quotient set M/E if it contains exactly one representative from each class of M/E.

$$M/E := \{[x] \mid x \in M\} \tag{2.19}$$

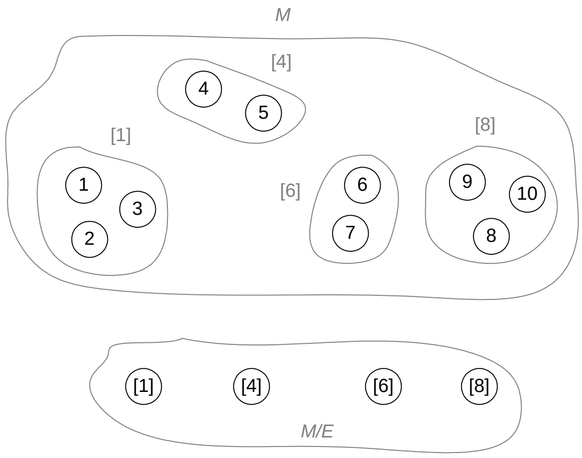


Figure 2.2: Quotient set

2.3.12 Connection

Consider the relation $R \subseteq M \times M$ in the set M. An n-tuple $(x_1, x_2, \ldots, x_n) \in M^n$ is called a connection of the elements a anb b by R in M if all ordered pairs (x_i, x_{i+1}) are contained in the relation R and $x_1 = a$, $x_n = b$. The number $n - 1$ of ordered pairs is called the length of the connection. For given elements a, b in M, there may be several connections with equal or different lengths. The statement "The elements a and b are connected by R" is designated by aV_Rb.

$$\begin{aligned} V_R &:= \left\{ (x_1, x_2, \ldots, x_n) \mid \bigwedge_{i \in \{1,\ldots,n-1\}} ((x_i, x_{i+1}) \in R) \right\} \\ aV_Rb &:\Leftrightarrow \bigvee_{(x_1,\ldots,x_n) \in V_R} (x_1 = a \wedge x_n = b) \end{aligned} \tag{2.20}$$

2.3.13 Closure

An extension of a homogeneous binary relation $R \subseteq M \times M$ is called a closure and is designated by $< R >$ if the following conditions are satisfied:

$$\begin{array}{lll} \text{inclusion} & : & R \sqsubseteq < R > \\ \text{isotonicity} & : & R \sqsubseteq S \Rightarrow < R > \sqsubseteq < S > \\ \text{idempotency} & : & << R >> = < R > \end{array} \tag{2.21}$$

The extension is performed such that the closure has special properties which the relation itself does not necessarily possess. Reflexive, symmetric and transitive closures are defined in the following. Closures may also have several of these properties.

2.3.13.1 Reflexive closure

The reflexive closure $< R >_r$ of a relation $R \subseteq M \times M$ is formed by adding the elements $(x, x) \in M \times M$ to R. The closure $< R >_r$ satisfies the condition for reflexive relations.

$$
\begin{array}{rcl}
< R >_r & := & \{(x,y) \mid (x,y) \in R \vee x = y \in M\} \\
< R >_r & = & R \sqcup I \\
I & \sqsubseteq & < R >_r \Rightarrow < R >_r \text{ is reflexive}
\end{array} \tag{2.22}
$$

2.3.13.2 Symmetric closure

The symmetric closure $< R >_s$ of a relation $R \subseteq M \times M$ is the union of R with its transpose R^T. If $< R >_s$ contains the element (x,y), then (y,x) is also an element of $< R >_s$. The closure $< R >_s$ satisfies the condition for symmetric relations.

$$
\begin{array}{rcl}
< R >_s & := & \{(x,y) \mid (x,y) \in R \vee (y,x) \in R\} \\
< R >_s & = & R \sqcup R^T \\
< R >_s & = & < R >_s^T \Rightarrow < R >_s \text{ is symmetric}
\end{array} \tag{2.23}
$$

2.3.13.3 Powers of a relation

In the algebra of relations, connections are represented by products of the relation R with itself. For example, if R contains the elements (a,b) and (b,c), then by definition the product $R \circ R$ contains the element (a,c). The element (a,c) is a connection of length 2 in R. Each of the elements of $R \circ R$ is a connection of length 2 in R. The power $R^m = R \circ \ldots \circ R$ (m-fold) contains all connections of length m between two elements of M. To determine all connections of length $m \le q$ in M by R, the union of the relations $R \sqcup R^2 \sqcup \ldots \sqcup R^q$ is formed.

2.3.13.4 Stability index

The least exponent s for which the union $R \sqcup R^2 \sqcup \ldots \sqcup R^s$is not changed by adding terms R^m with $m > s$ is called the stability index of the relation R. The union $R \sqcup R^2 \sqcup \ldots \sqcup R^s$ contains all connections by R in M.

The stability index s of a relation R may be interpreted as follows. If there are several connections between two elements of M, then there is a shortest connection, of length q, which is contained in R^q. Among all the shortest connections between pairs of elements, there is a shortest connection of maximal length s, which is contained in the power R^s. Hence the union $R \sqcup R^2 \sqcup \ldots \sqcup R^s$ contains all connections in M by R. For a set M with n elements, the stability index s of the relation $R \subseteq M \times M$ is less than n, since the maximal length of all shortest connections in M by R cannot be greater than $n-1$.

2.3.13.5 Transitive closure

The transitive closure $< R >_t$ of a relation $R \subseteq M \times M$ contains all elements $(x,y) \in M \times M$ which are connected in M by R. The closure $< R >_t$ satisfies the condition for transitive closures.

$$
\begin{array}{l}
< R >_t := \{(x,y) \in M \times M \mid x \text{ and } y \text{ are connected in } M \text{ by } R\} \\
< R >_t := R \sqcup \ldots \sqcup R^s \\
< R >_t \circ < R >_t \sqsubseteq < R >_t \Rightarrow < R >_t \text{ is transitive} \\
s \text{ stability index of } R \text{ with } < R >_t \sqcup R^{s+1} = < R >_t
\end{array} \tag{2.24}
$$

2.3.13.6 Reflexive transitive closure

The reflexive transitive closure $< R >_{rt}$ of a relation $R \subseteq M \times M$ may alternatively be regarded as the transitive closure $<< R >_r>_t$ of the reflexive closure $< R >_r$ or as the reflexive closure $<< R >_t>_r$ of the transitive closure $< R >_t$. The two viewpoints lead to identical relations. The closure $< R >_{rt} = < R >$ satisfies the condition for transitive relations in the special form of an equation, given in Equation 2.25.

$$
\begin{array}{rcll}
< R >_{rt} & := & << R >_r >_t & < R >_{tr} := << R >_t >_r \\
< R >_{rt} & = & < R >_{tr} & \\
< R >_{rt} & \circ & < R >_{rt} = < R >_{rt} \Rightarrow & < R >_{rt} \text{ is transitive}
\end{array}
\tag{2.25}
$$

2.3.13.7 Reflexive symmetric transitive closure

The reflexive symmetric transitive closure $< R >_{rst}$ of a relation $R \subseteq M \times M$ is the transitive closure of the symmetric closure of the transitive closure of R . It coincides with the reflexive symmetric transitive closure $< R >_{rst}$. The closure $< R >_{rst}$ is of special importance, as it is an equivalence relation and therefore yields a classification of the set M.

$$
\begin{array}{rcl}
< R >_{rst} & := & <<< R >_r >_s >_t = << R >_s >_{rt} = < R \sqcup R^T >_{rt} \\
< R >_{rst} & = & < R \sqcup I \sqcup R^T >_t
\end{array}
\tag{2.26}
$$

2.3.14 Algebra of homogeneous binary relations

Directed graphs will be considered in the following sections. Since the edge set of a directed graph is a homogeneous binary relation on the vertex set, the properties of homogeneous binary relations and their rules of calculation may be directly transferred to directed graphs. Therefore the algebra of homogeneous binary relations will now be explained in greater detail.

Since every relation is a set, the rules of the algebra of sets also hold for homogeneous binary relations. Additional properties and rules result from the duality and composition of relations.

2.3.14.1 Graphical representation

A homogeneous binary relation R on a set M can be visually represented in a graph diagram. The graph diagram consist of a point set which represents the set M of elements with their designations. If an element x is related to an element y, an arrow is drawn from the point x to the point y. The homogeneous relation R corresponds to the resulting set of arrows. The graph diagram shows the elements of the set M and the relationships in a network-like structure. It is the representation used in graph theory. The points used to represent the elements are called vertices, the arrows are called directed edges.

Example

$$
\begin{array}{rcl}
M & = & \{a, b, c, d, e\} \\
R & = & \{(a, b), (a, d), (b, a), (c, a), (c, d), (d, c), (d, e), (e, e)\}
\end{array}
$$

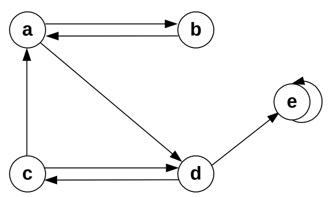


Figure 2.3: Homogeneous binary relations graph example

2.3.14.2 Special relations

The null relation (empty relation) ϕ, the identity relation I and the all relation (universal relation) E are special homogeneous binary relations on the set M.

$$\begin{array}{llll} \text{null relation} & \phi & = & \{\} \\ \text{identity relation} & I & = & \{(a,a) \mid a \in M\} \\ \text{all relation} & E & = & M \times M \end{array} \tag{2.27}$$

2.3.14.3 Equality and inclusion

The operations of equality $R = S$ and inclusion $R \sqsubseteq S$ on homogeneous relations R and S are equal. If $R \sqsubseteq S$ is true, then R is contained in S.

$$\begin{array}{llll} \text{equality} & R = S & :\Leftrightarrow & \bigwedge_a \bigwedge_b ((a,b) \in R \Leftrightarrow (a,b) \in S) \\ \text{inclusion} & R \sqsubseteq S & :\Leftrightarrow & \bigwedge_a \bigwedge_b ((a,b) \in R \Rightarrow (a,b) \in S) \\ \text{equality} & \mathbf{R} = \mathbf{S} & :\Leftrightarrow & \bigwedge_i \bigwedge_j (r_{ij} \Leftrightarrow s_{ij}) \\ \text{inclusion} & \mathbf{R} \sqsubseteq \mathbf{S} & :\Leftrightarrow & \bigwedge_i \bigwedge_j (r_{ij} \Rightarrow s_{ij}) \end{array} \tag{2.28}$$

2.3.14.4 Binary operations

The intersection $R \sqcap S$, the union $R \sqcup S$ and the product $R \circ S$ are binary operations on the homogeneous relations R and S. The intersection and the union are defined as in set theory. The product corresponds to the composition of two relations; the operation of forming products is called multiplication. In the algebra of relations it is convenient to define the composition $R \circ S$ of the relations in the order "first R, then S". This definition allows direct transfer to boolean matrix algebra.

$$\begin{array}{lllll} \text{intersection} & R \sqcap S & := & \{(x,y) \mid & (x,y) \in R \wedge (x,y) \in S\} \\ \text{union} & R \sqcup S & := & \{(x,y) \mid & (x,y) \in R \vee (x,y) \in S\} \\ \text{product} & R \circ S & := & \{(x,y) \mid \bigvee_z & ((x,z) \in R \wedge (z,y) \in S)\} \end{array} \tag{2.29}$$

$$\begin{array}{llll} \text{intersection} & \mathbf{R} \sqcap \mathbf{S} & := & [r_{ij} \wedge s_{ij}] \\ \text{union} & \mathbf{R} \sqcup \mathbf{S} & := & [r_{ij} \vee s_{ij}] \\ \text{product} & \mathbf{R} \circ \mathbf{S} & := & \left[\bigvee_k r_{ik} \wedge s_{kj}\right] \end{array} \tag{2.30}$$

Chapter 3

Directed graphs

3.1 Introduction

A directed graph (see Section 3.2) is suitable for describing relationships between the elements of a set such as the “has to be executed before” relation in the set of tasks. The task elements of the set are called vertices of the graph and are identified by their labels. The relationships between the vertices are called edges of the graph and are identified by an ordered vertex pair. Therefore, the edge set is a homogeneous binary relation on the vertex set. The properties of homogeneous binary relations and their rules of calculation (see Section 2.3.6) may be directly transferred to directed graphs.

A directed graph is a structured set. It consists of the vertex set V and a homogeneous binary vertex relation R which corresponds to a set of directed edges. The vertex set V is equipped with structure by the vertex relation R. The structural properties of a directed graph are entirely determined by the properties of the relation R.

A graph may be decomposed into subgraphs which have simple structural characteristics and yield insight into the essential structural properties of the graph. Paths and cycles (see Section 3.4.1) are examples of such subgraphs. The definition of paths and cycles in a directed graph forms the basis of the structural analysis of directed graphs. The existence of paths and cycles between two vertices leads to the formation of the transitive closure R^+ of the relation R. The properties of the transitive closure allow a classification into acyclic, anticyclic and cyclic graphs.

In a directed graph, a vertex may or may not be reachable from another vertex along the directed edges. The concept of reachability forms the basis for a definition of the connectedness of vertices (see Section 3.4.2). Different kinds of connectedness may be defined, such as strong and weak connectedness. Directed graphs which are not strongly or weakly connected may be decomposed uniquely into strongly or weakly connected subgraphs. These subgraphs are called strongly or weakly connected components, respectively. The decomposition of a graph into its strongly connected components (see Section 3.4.2.8) leads to an acyclic reduced graph.

It may be convenient to assign to each element of a set A exactly one element of a set Z. The same element of Z may be assigned to different elements of A. Relations of this type are called mappings (see Section 3.3). Each vertex can be mapped in this way to a vertex in its strongly connected component.

The acyclicity of a graph (see Section 3.4.3) leads to special structural properties of the graph. Directed acyclic graphs possess an order structure. The vertex set is an ordered set. The directed edges describe the order relation in the vertex set. Due to the order structure, the vertices can be sorted topologically.

See reference [4] for a detailed description of directed graphs, mappings and the structural properties of directed graphs.

3.2 Directed graphs

3.2.1 Definition

$G := (V; R)$ is called a directed graph if V is the vertex set and $R \subseteq V \times V$ is the edge set of the graph. An edge from the vertex $x \in V$ to the vertex $y \in V$ is designated by the ordered pair $(x, y) \in R$. The edge (x, y) is said to be directed from x to y. The vertex x is called the start vertex of the edge. The vertex y is called the end vertex of the edge.

$$\begin{array}{lll} G & := & (V; R) \qquad R \subseteq V \times V \\ V & & \text{set of vertices} \\ R & & \text{set of ordered vertex pairs (edge set)} \end{array} \tag{3.1}$$

The graph G is called a null graph if the vertex set is empty. It is called an empty graph if the edge set is empty. It is called a complete graph if the edge set R is the all relation $E = V \times V$.

3.2.2 Properties

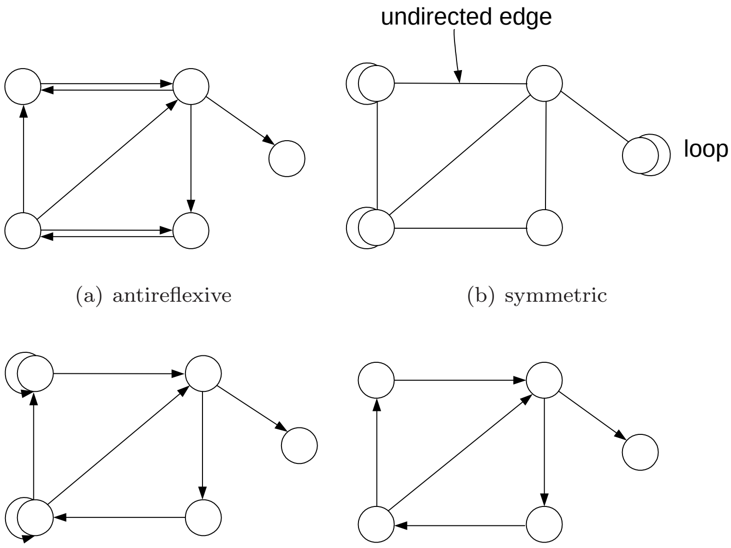
The properties of a directed graph $(V; R)$ are determined by the properties of the homogeneous binary relation R. The properties of homogeneous relations described in Table 2.2 are therefore transferred to directed graphs in Table 3.1. Antireflexive, symmetric, antisymmetric and asymmetrix graphs are important in applications:

$$G = (V; R)$$

Table 3.1: Properties of directed graphs

G is antireflexive	$:\Leftrightarrow$	$I \sqsubseteq \bar{R}$
G is symmetric	$:\Leftrightarrow$	$R = R^T$
G is antisymmetric	$:\Leftrightarrow$	$R \sqcap R^T \sqsubseteq I$
G is asymmetric	$:\Leftrightarrow$	$R \sqcap R^T = \emptyset$

For an antireflexive graph, the edge set does not contain vertex pairs of the form (x, x), and the graph diagram is free of loops (see Section 3.4.1.6). Between two different vertices in the graph diagram, a symmetric graph contains either no edge or a pair of edges with opposite directions, which are combined into an undirected edge. An antisymmetric graph contains either no edges or only one directed edge between two vertices in the graph diagram. Symmetric and antisymmetric graphs may contain loops. An asymmetric graph is antisymmetric and antireflexive, and hence free of loops. The graphs we will be considering are asymmetric.



(a) antireflexive (b) symmetric

(c) antisymmetric (d) asymmetric

Figure 3.1: Directed graph properties

3.2.3 Equality and inclusion

Let two directed graphs G_1 and G_2 be given. Using the algebra of relations, equality and inclusion are defined as follows for these graphs:

$$\begin{array}{llllllllll} \text{equality} & G_1 & = & G_2 & :\Leftrightarrow & V_1 = V_2 & \wedge & R_1 & = & R_2 \\ \text{partial graph} & G_1 & \sqsubseteq & G_2 & :\Leftrightarrow & V_1 = V_2 & \wedge & R_1 & \sqsubseteq & R_2 \\ \text{subgraph} & G_1 & \subseteq & G_2 & :\Leftrightarrow & V_1 \subseteq V_2 & \wedge & R_1 & \sqsubseteq & R_2 \sqcap (V_1 \times V_1) \end{array} \tag{3.2}$$

A partial graph (spanning subgraph) G_1 is generated from a graph G_2 by removing edges from G_2. A subgraph G_1 is generated from a graph G_2 by first removing vertices together with the incident edges and then removing further edges form G_2.

3.2.4 Adjacency-matrix graph representation

Graphs can be represented using different data structures, one of which is the adjacency-matrix.

Let V be a set with n elements. The elements of V are indexed by a mapping $\Phi : N \to V$ with $\Phi(i) = x_i$ and $1 \leq i \leq n$, so that $V = \{x_1, \ldots, x_n\}$. A homogeneous binary relation $R \subseteq V \times V$ is a subset of $V \times V$. The elements of $V \times V$ which belong to the relation are specified by a boolean matrix $\mathbf{R}$ of dimension $n \times n$. Every element $(x_i, x_j) \in V \times V$ is bijectively associated with an element $r_{ij} \in \mathbf{R}$. If the relation R contains the element (x_i, x_j), then r_{ij} has the value true (1); otherwise r_{ij} has the value false (0).

A boolean matrix $\mathbf{R}$ of a homogeneous relation R is an n^2-tuple of the truth values $W = \{0, 1\}$, and hence an element of the n^2-fold cartesian product $W^{n \cdot n}$. The elements of a matrix $\mathbf{R}$ are usually arranged in a row and column scheme by regarding the indices i, j of the element r_{ij} as row and column indices, respectively. In formulations of general properties and rules, a matrix $\mathbf{R}$ is represented by a general element r_{ij} in square brackets.

$$\mathbf{R} = [r_{ij}] = \begin{matrix} r_{11} & \cdots & r_{1j} & \cdots & r_{1n} \\ \vdots & & \vdots & & \vdots \\ r_{i1} & \cdots & r_{ij} & \cdots & r_{in} \\ \vdots & & \vdots & & \vdots \\ r_{n1} & \cdots & r_{nj} & \cdots & r_{nn} \end{matrix} \qquad \begin{matrix} W = \{0,1\} \\ \mathbf{R} \in W^{n \cdot n} \end{matrix} \tag{3.3}$$

3.3 Mappings

3.3.1 Mapping notation

A relation $\Phi \subseteq A \times Z$ is called a mapping if it is left-total and right-unique. The following notation is used:

$$\begin{array}{ll} \Phi : A \to Z & \Phi \text{ is a mapping from } A \text{ of } Z \\ A & \text{domain of } \Phi \\ Z & \text{target of } \Phi \end{array} \tag{3.4}$$

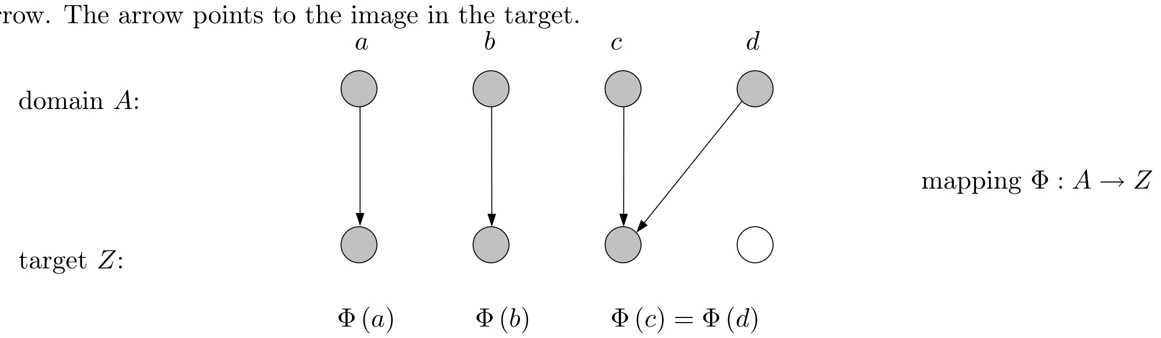
3.3.2 Image of an element

If the mapping Φ assigns the element $z \in Z$ to the element $a \in A$, then z is called the image of a under the mapping Φ. The element a is called a preimage (inverse image) of z. The following notation is used:

$$\Phi : a \to z \quad \text{or} \quad \Phi(a) = z \tag{3.5}$$

3.3.3 Arrow diagram

Mappings are depicted using arrow diagrams. Every element of the domain is the starting point of an arrow. The arrow points to the image in the target.

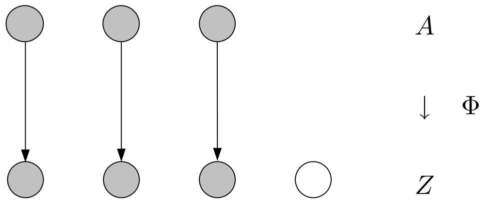


3.3.4 Types of mappings

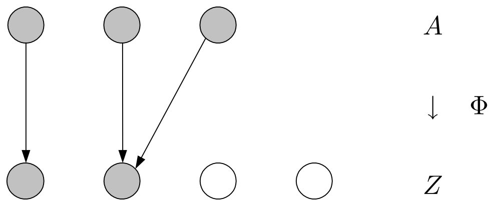
All mappings are left-total and right-unique relations. Mappings often have additional properties. Mappings with common additional properties belong to a type of mappings.

3.3.4.1 Injective mapping

A mapping $\Phi : A \to Z$ is said to be injective (an injection) if two different elements $a \neq b$ of the set A always possess two different images $\Phi(a) \neq \Phi(b)$. An injection is a left-total, bi-unique relation. From $\Phi(a) = \Phi(b)$ it follows that $a = b$.



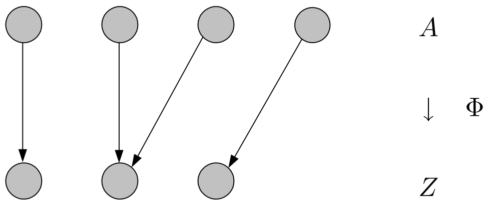
injection



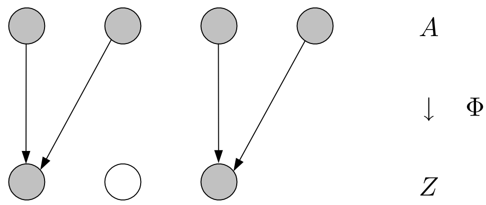
not an injection

3.3.4.2 Surjective mapping

A mapping $\Phi : A \rightarrow Z$ is said to be surjective (a surjection) if each element of the target Z is the image of at least one element of A. A surjection is a bitotal, right-unique relation. An element $z \in Z$ may be the image of more than one element in A.



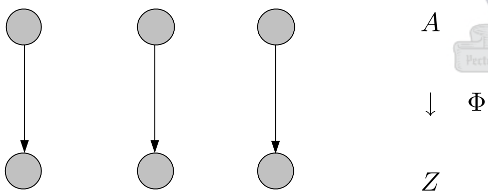
surjection



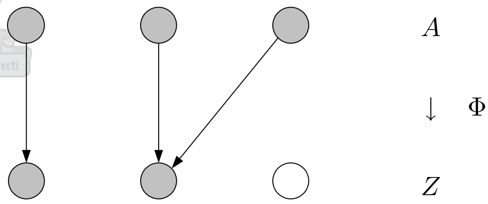
not a surjection

3.3.4.3 Bijective mapping

A mapping $\Phi : A \rightarrow Z$ is said to be bijective (a bijection) if every element of Z is the image of exactly one element of A. A bijection if a bitotal, bi-unique relation. The number of elements in A and Z is the same.



bijection



not a bijection

3.3.4.4 Canonical mapping

The surjection from a set M to its quotient set M/E for a given equivalence relation E is called a canonical mapping of M. The image of the element $a \in M$ is the equivalence class $[a]$.

$$k \quad : \quad M \rightarrow M/E \quad \text{with} \quad k(a) = [a] \tag{3.6}$$

Example

The example in Section 2.3.11.7 will be used to show the canonical mapping between the set M and its quotient set M/E.

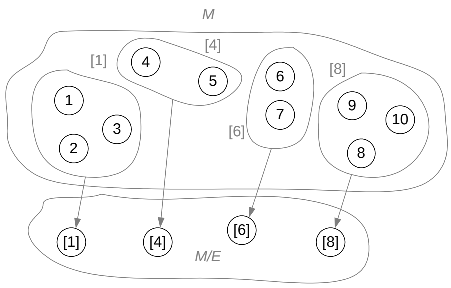


Figure 3.2: Canonical mapping

3.4 Structure of graphs

3.4.1 Paths and cycles in directed graphs

3.4.1.1 Predecessor and successor

A vertex x is called a predecessor of a vertex y if there is an edge from x to y in the graph, so that the ordered vertex pair (x, y) is contained in the relation R. If x is a predecessor of y, then y is called a successor of x.

$$\begin{array}{lcl} x \text{ predecessor of } y & \Leftrightarrow & (x, y) \in R \\ y \text{ successor of } x & \Leftrightarrow & (x, y) \in R^T \end{array} \tag{3.7}$$

A vertex x in a vertex set V may be regarded as a unary point relation in V. In the following, this unary point relation is also designated by x. The predecessorship and the successorship of vertices $x, y \in V$ are formulated as an inclusion using such unary relations:

$$\begin{array}{lcl} x \text{ predecessor of } y & \Leftrightarrow & xy^T \sqsubseteq R \\ y \text{ successor of } x & \Leftrightarrow & yx^T \sqsubseteq R^T \end{array}$$

The set of all predecessors of a vertex $x \in V$ is designated by $t_p(x)$ and the set of all successors of x by $t_s(x)$. The sets $t_p(x)$ and $t_s(x)$ are unary relations in V and are determined as follows using the edge relation R:

$$\begin{array}{llcl} \text{predecessors of } x: & t_p(x) & = & Rx \\ \text{successors of } x: & t_s(x) & = & R^T x \end{array}$$

3.4.1.2 Indegree and outdegree

The number of predecessors of a vertex x is called the indegree of x and is designated by $g_p(x)$. The indegree of $g_p(x)$ corresponds to the number of elements in the set $t_p(x) = |t_p(x)|$ and hence to the number of directed edges which end at the vertex $x = |Rx|$. The number of successors of a vertex x is called the outdegree of x and is designated by $g_s(x)$. The outdegree $g_s(x)$ corresponds to the number of elements in the set $t_s(x) = |t_s(x)|$, and hence to the number of directed edges which emanated from the vertex $x = \left|R^T x\right|$.

$$\begin{array}{llclcl} \text{indegree} & g_p(x) & = & |t_p(x)| & = & |Rx| \\ \text{outdegree} & g_s(x) & = & |t_s(x)| & = & \left|R^T x\right| \end{array} \tag{3.8}$$

The sum of the indegrees of all vertices $x \in V$ is equal to the number of directed edges of the directed graph, and hence coincides with the number of elements of the relation $R = |R|$. The same is true for the outdegrees.

$$\text{sum} \quad \sum_{x \in V} g_p(x) = \sum_{x \in V} g_s(x) = |R|$$

3.4.1.3 Edge sequence

A chain of edges is called an edge sequence if the end vertex of each edge except for the last edge is the start vertex of the following edge.

$$\begin{gathered} < (x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n) > \\ \bigwedge_{j=1}^{n} ((x_{j-1}, x_j) \in R) \end{gathered} \tag{3.9}$$

The start vertex x_0 of the first edge and the end vertex x_n of the last edge are called the start vertex and the end vertex of the edge sequence, respectively. The vertices x_1 to x_{n-1} are called intermediate vertices of the edge sequence. The number n of edges is called the length of the edge sequence. An edge may occur more than once in an edge sequence.

3.4.1.4 Ancestors and descendents

A vertex x is called an n^{th} ancestor of a vertex y if there is an edge sequence of length n from x to y in the graph. If x is an n^{th} ancestor of y, then y is called an n^{th} descendant of x. A 1^{st} ancestor or 1^{st} descendant of x is a predecessor or successor of x, respectively. The n^{th} ancestors and descendants of x are determined recursively from the relationships for predecessors and successors according to the following rule:

n^{th} ancestors of x:

$$\begin{array}{lll} t_p^{(k)}(x) = R t_p^{(k-1)}(x) & \text{for} \quad k = 1, \ldots, n & \text{with} \quad t_p^{(0)}(x) = x \\ t_p^{(n)}(x) = R^n x & \text{for} \quad n > 0 & \end{array} \tag{3.10}$$

n^{th} descendants of x:

$$\begin{array}{lll} t_s^{(k)}(x) = R^T t_s^{(k-1)}(x) & \text{for} \quad k = 1, \ldots, n & \text{with} \quad t_s^{(0)}(x) = x \\ t_s^{(n)}(x) = (R^n)^T x & \text{for} \quad n > 0 & \end{array} \tag{3.11}$$

The set of all ancestors of a vertex x is designated by $t_p^+(x)$; it is determined as the union of the sets of n^{th} ancestors of x. The set $t_s^+(x)$ of all descendants of x is determined analogously. The transitive closure R^+ of a relation R with stability index s, may be used to determine these sets:

ancestors of x:

$$t_p^+(x) = t_p^{(1)}(x) \sqcup \ldots \sqcup t_p^{(s)}(x) = Rx \sqcup \ldots \sqcup R^s x = R^+ x$$

descendants of x:

$$t_s^+(x) = t_s^{(1)}(x) \sqcup \ldots \sqcup t_s^{(s)}(x) = Rx \sqcup \ldots \sqcup R^{sT} x = R^{+T} x$$

3.4.1.5 Path

A path from a start vertex x via intermediate vertices to an end vertex y is an edge sequence. In a directed graph, a path may be uniquely represented as a vertex sequence $< x, \ldots, y >$. A path $< x >$ with the same start and end vertex x contains no edges and is called an empty path. The length of an empty path is 0. There is an empty path for every vertex of a directed graph. The existence of non-empty paths in a directed graph is established as follows:

$$\begin{array}{lcl} \text{there is a path of length } n \text{ from } x \text{ to } y & \Leftrightarrow & xy^T \sqsubseteq R^n \\ \text{there is a non-empty path from } x \text{ to } y & \Leftrightarrow & xy^T \sqsubseteq R^+ \end{array} \tag{3.12}$$

3.4.1.6 Cycle

A non-empty path whose start vertex and end vertex coincide is called a cycle. A loop at a vertex is a cycle of length 1. A cycle which contains no loops is called a proper cycle. If there is a non-empty path from x to y and a non-empty path from y to x, then the concatenation of the two paths yields a cycle through x and y. The existence of cycles in a directed graph is established as follows:

$$\begin{array}{lcl} \text{there is a cycle of length } n > 0 \text{ through } x & \Leftrightarrow & xx^T \sqsubseteq R^n \\ \text{there is a cycle through } x & \Leftrightarrow & xx^T \sqsubseteq R^+ \\ \text{there is a cycle through } x \text{ and } y & \Leftrightarrow & xy^T \sqsubseteq R^+ \sqcap R^{+T} \end{array} \tag{3.13}$$

3.4.1.7 Acyclic graph

A directed graph $G = (V; R)$ is said to be acyclic if it does not contain any cycles. The transitive closure R^+ of an acyclic graph is asymmetric. If there is a non-empty path from x to y, then there is no non-empty path from y to x, since otherwise the concatenation of the two paths would yield a cycle.

$$R^+ \sqcap R^{+T} = 0 \tag{3.14}$$

3.4.1.8 Anticyclic graph

A directed graph $G = (V; R)$ is said to be anticyclic if it does not contain any proper cycles. In contrast to acyclic graphs, an anticylic graph may contain loops at the vertices. The transitive closure R^+ of an anticyclic graph is antisymmetric.

$$R^+ \sqcap R^{+T} \sqsubseteq I \tag{3.15}$$

3.4.1.9 Cyclic graph

A directed graph $G = (V; R)$ is said to be cyclic if every non-empty path in G belongs to a cycle. The transitive closure R^+ of a cyclic graph is symmetric. If there is a non-empty path from x to y, then there is also a non-empty path from y to x, so that the concatenation of the two paths yields a cycle.

$$R^+ = R^{+T} \tag{3.16}$$

3.4.1.10 Properties

The following relationships hold between the properties of a relation R and of its transitive closure R^+. If the transitive closure R^+ is asymmetric or antisymmetric, then the relation R is asymmetric or antisymmetric, respectively. If the relation R is symmetric, then the transitive closure R^+ is symmetric. These relationships lead to the following implications:

$$
\begin{array}{lcl}
\text{acyclic graph} & \Rightarrow & \text{asymmetric graph} \\
\text{anticyclic graph} & \Rightarrow & \text{antisymmetric graph} \\
\text{cyclic graph} & \Leftarrow & \text{symmetric graph}
\end{array}
$$

3.4.1.11 Simple path

A non-empty path is said to be simple if it does not contain any edge more than once. The vertices and the edges of a simple path form a subgraph of the directed graph. If the start vertex and end edge of a simple path are different, the following relationships hold between the indegrees and the outdegrees of the vertices of the corresponding subgraph:

subgraph for a simple path $< x, \ldots, z, \ldots, y >$ with $x \neq y$

$$
\begin{array}{ll}
\text{start vertex} & g_s(x) = g_p(x) + 1 \\
\text{intermediate vertex} & g_s(z) = g_p(z) \\
\text{end vertex} & g_p(y) - 1
\end{array}
\tag{3.17}
$$

3.4.1.12 Simple cycle

A simple path whose start vertex and end vertex coincide is called a simple cycle. In the subgraph for a simple cycle, the indegree and the outdegree of each vertex are equal.

subgraph for a simple cycle with vertex z

$$
\text{vertex} \quad g_s(z) = g_p(z)
\tag{3.18}
$$

3.4.1.13 Elementary path

A non-empty path is said to be elementary if it does not contain any vertex more than once. The vertices and the edges of an elementary path form a subgraph. If the start vertex and the end vertex of an elementary path are different, then the vertices of the corresponding subgraph have the following indegrees and outdegrees:

subgraph for an elementary path $< x, \ldots, z, \ldots, y >$ with $x \neq y$

$$
\begin{array}{lll}
\text{start vertex} & g_s(x) = 1 & g_p(x) = 0 \\
\text{intermediate vertex} & g_s(z) = 1 & g_p(z) = 1 \\
\text{end vertex} & g_s(y) = 0 & g_p(y) = 1
\end{array}
\tag{3.19}
$$

3.4.1.14 Elementary cycle

An elementary path whose start vertex and end vertex coincide is called an elementary cycle. In the subgraph of an elementary cycle, the indegree and the outdegree of every vertex are equal to 1. Note that the identical start and end vertex is counted once, not twice.

subgraph for an elementary cycle with vertex z

$$
\text{vertex} \quad g_s(z) = g_p(z) \quad = 1
\tag{3.20}
$$

3.4.2 Connectedness of directed graphs

3.4.2.1 Reachability

In a directed graph $G = (V; R)$, a vertex $y \in V$ is said to be reachable from a vertex $x \in V$ if there is an empty or non-empty path from x to y. Vertex y is reachable from vertex x if and only if the product xy^T of the associated point relations x and y is contained in the reflexive transitive closure R^*.

$$y \text{ is reachable from } x :\Leftrightarrow xy^T \sqsubseteq R^* \qquad R^* = I \sqcup R^+ \tag{3.21}$$

3.4.2.2 Strong connectedness

Two vertices x and y of a directed graph are said to be strongly connected if x is reachable from y and y is reachable from x. A directed graph is said to be strongly connected if all vertices are pairwise strongly connected.

$$\begin{array}{lll} x \text{ and } y \text{ are strongly connected} & :\Leftrightarrow & xy^T \sqsubseteq R^* \sqcup R^{*T} \\ \text{the graph is strongly connected} & :\Leftrightarrow & R^* \sqcap R^{*T} = E \Leftrightarrow R^* = E \end{array} \tag{3.22}$$

3.4.2.3 Unilateral connectedness

Two vertices x and y of a directed graph are said to be unilaterally connected if x is reachable from y or y is reachable from x. A directed graph is said to be unilaterally connected if all vertices are pairwise unilaterally connected.

$$\begin{array}{lll} x \text{ and } y \text{ are unilaterally connected} & :\Leftrightarrow & xy^T \sqsubseteq R^* \sqcup R^{*T} \\ \text{the graph is unilaterally connected} & :\Leftrightarrow & R^* \sqcap R^{*T} = E \end{array} \tag{3.23}$$

3.4.2.4 Weak connectedness

Two vertices x and y of a directed graph $(V; R)$ are said to be weakly connected if they are strongly connected in the symmetric graph $G = \left(V; R \sqcup R^T\right)$. A directed graph is said to be weakly connected if all vertices are pairwise weakly connected. Since the transitive closure of a symmetric relation is symmetric, this definition may be expressed as follows:

$$\begin{array}{lll} x \text{ and } y \text{ are weakly connected} & :\Leftrightarrow & xy^T \sqsubseteq \left(R \sqcup R^T\right)^* \\ \text{the graph is weakly connected} & :\Leftrightarrow & \left(R \sqcup R^T\right)^* = E \end{array} \tag{3.24}$$

3.4.2.5 Connectedness relations

The relation R of a directed graph $G = (V; R)$ generally contains strong, unilateral and weak connections. A relation which contains only connections of the same type is called a connectedness relation. The connectedness relations for a directed graph G are derived from the relation R and its reflexive transitive closure R^*:

$$\begin{array}{ll} \text{strong connectedness relation} & S = R^* \sqcap R^{*T} \\ \text{unilateral connectedness relation} & P = R^* \sqcup R^{*T} \\ \text{weak connectedness relation} & C = \left(R \sqcup R^T\right)^* \end{array} \tag{3.25}$$

A strongly connected vertex pair is also unilaterally connected; a unilaterally connected vertex pair is also weakly connected. Hence a strongly connected graph is also unilaterally connected, and a unilaterally connected graph is also weakly connected. For a symmetric graph, the three different kinds of connectedness coincide.

$$\begin{array}{lllllll} \text{inclusion} & : & R^* \sqcap R^{*T} & \sqsubseteq & R^* \sqcup R^{*T} & \sqsubseteq & \left(R \sqcup R^T\right)^* \\ \text{connectedness} & : & \text{strong} & \Rightarrow & \text{unilateral} & \Rightarrow & \text{weak} \end{array}$$

Two different vertices which are strongly connected lie on a cycle. A strongly connected graph is therefore cyclic. The converse is not true in the general case.

$$\text{strongly connected graph} \;\Rightarrow\; \text{cyclic graph}$$

3.4.2.6 Properties of the connectedness relations

The strong connectedness relation S is reflexive, symmetric and transitive. Reflexivity and symmetry follow directly from the definition. Transitivity follows from the following consideration. If (x, y) and (y, z) are strongly connected vertex pairs, then z is reachable from x via y and x is reachable from z via y. Hence (x, z) is also a strongly connected vertex pair.

The unilateral connectedness relation P is reflexive and symmetric, but generally not transitive. This follows from the following consideration. If (x, y) and (y, z) are unilaterally connected vertex pairs, then it is possible that x is only reachable from y and z is only reachable from y. In this case, neither is x reachable from z, nor is z reachable from x. Thus (x, z) is not a unilaterally connected vertex pair.

The weak connectedness relation C is by definition the strong connectedness relation of an associated symmetric graph. This is reflexive, symmetric and transitive.

A reflexive, symmetric and transitive relation is an equivalence relation. Hence the strong and weak connectedness relations are equivalence relations. The unilateral connectedness relation is generally not an equivalence relation.

3.4.2.7 Decomposition into connected components

The strong connectedness relation $S = \left(R \sqcup R^T\right)^*$of a directed graph $G = (V; R)$ is an equivalence relation. The graph $(V; R)$ is connected if the equivalence relation S is the all relation E. If the graph $(V; R)$ is disconnected, then it may be uniquely decomposed into connected subgraphs. The subgraphs are called the connected components of the graph. The decomposition is carried out in the following steps, independent of the kind of connectedness being considered:

1. Connectedness class: The vertex set V of the graph is partitioned into connected classes, using the relation S. A connected class $[x]$ with the vertex x as a representative contains all vertices of V which are connected with x. The class $[x]$ is a unary relation and is determined as follows:

$$[x] = Sx \tag{3.26}$$

2. Mapping: The set K of all connected classes is the quotient set V/S. Each vertex $x \in V$ is mapped to exactly one connected class, yielding a canonical mapping Φ:

$$\Phi : V \to K \qquad \text{with} \quad K = V/S \tag{3.27}$$

3. Reduced graph: The mapping Φ from the vertex set V of the directed graph $G = (V; R)$ to the set

K of connected classes induces the reduced graph $G_K = (K; R_K)$.

$$G_K = (K; R_K) \qquad \text{with} \quad R_K = \Phi^T R \Phi \tag{3.28}$$

4. Connected component: A connected component is a connected subgraph $G_k := (V_k, R_k)$ of a directed graph $G = (V; R)$. The vertex set V_k contains all vertices of a connected class K of the graph $(V; R)$. The edge set $R_k = R \sqcap (V_k \times V_k)$ contains the edges from R whose vertices belong to V_k. The union of all connected components G_k is generally a partial graph of G, since the union of all vertex sets V_k is the vertex set V and the union of all edge sets R_k is only a subset of the edge set R.

$$\bigsqcup_{k \in K} G_k \sqsubseteq G \tag{3.29}$$

3.4.2.8 Decomposition into strongly connected components

The vertex set V of a directed graph $G = (V; R)$ may be decomposed into strongly connected classes using its strong connectedness relation $S = R^* \sqcap R^{*T}$. Two different classes cannot be strongly connected in the reduced graph $G_K = (K; R_K)$, since strongly connected vertices belong to the same class. Each connected component $G_k = (V_k; R_k)$ has a symmetric transitive closure R_k^+ and is therefore a cyclic graph. The reduced graph $G_K = (K; R_K)$ has an antisymmetric transitive closure R_K^+ and is therefore an anticyclic graph.

3.4.2.9 Decomposition into weakly connected components

The vertex set V of a directed graph $G = (V; R)$ may be decomposed into weakly connected classes using its weak connectedness relation $C = \left(R \sqcup R^T\right)^*$. Two different classes cannot be weakly connected in the reduced graph $G_K = (K; R_K)$, since weakly connected vertices belong to the same class and the two vertices of an edge are at least weakly connected. Hence every directed graph is the union of its weakly connected components.

$$G = \bigsqcup_{k \in K} G_k \tag{3.30}$$

3.4.2.10 Strongly connected components example

Graph

The directed graph in Figure 3.3 will be used to demonstrate the decomposition of a directed graph into its strongly connected components.

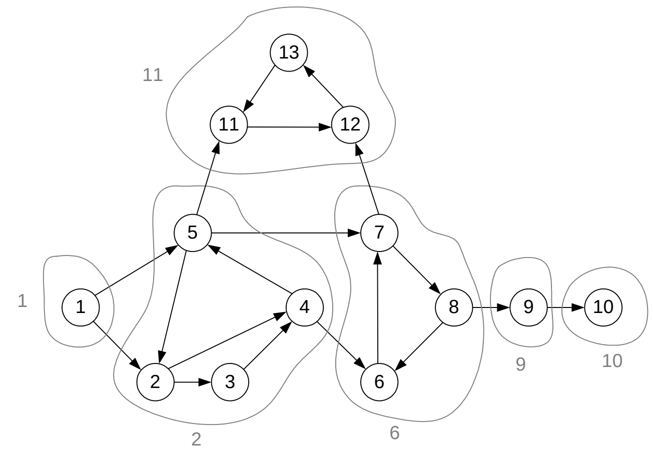
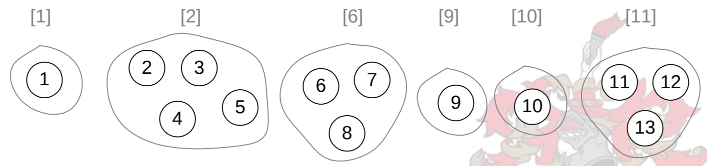


Figure 3.3: Strongly connected components graph example

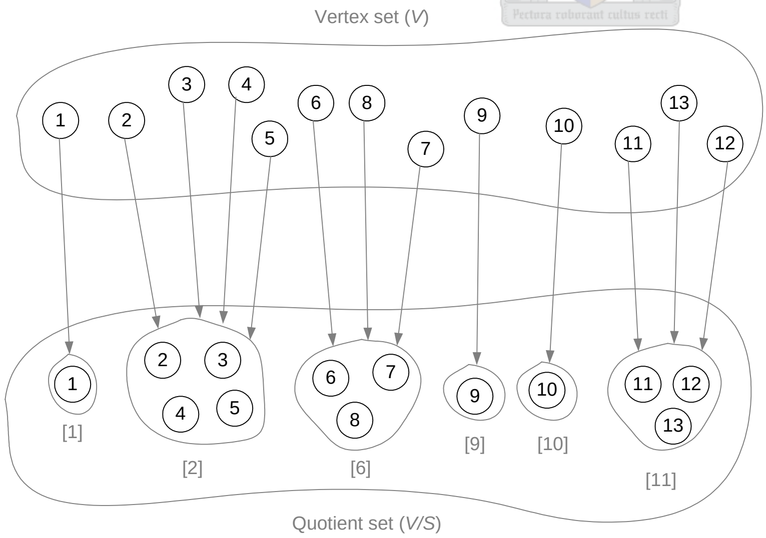
Strongly connectedness classes



$$[x] = Sx$$

$$\begin{array}{rcl}
[1] & = & \{1\} \\
[2] & = & \{2,3,4,5\} \\
[6] & = & \{6,7,8\} \\
[9] & = & \{9\} \\
[10] & = & \{10\} \\
[11] & = & \{11,12,13\}
\end{array}$$

Mapping



$$\Phi : V \to V/S$$

$$\begin{array}{rcl}
1 & \to & \{1\} \\
2 & \to & \{2,3,4,5\} \\
3 & \to & \{2,3,4,5\} \\
4 & \to & \{2,3,4,5\} \\
5 & \to & \{2,3,4,5\} \\
6 & \to & \{6,7,8\} \\
7 & \to & \{6,7,8\} \\
8 & \to & \{6,7,8\} \\
9 & \to & \{9\} \\
10 & \to & \{10\} \\
11 & \to & \{11,12,13\} \\
12 & \to & \{11,12,13\} \\
13 & \to & \{11,12,13\}
\end{array}$$

Reduced graph

$$G_K = (K; R_K)$$

$$\begin{array}{ll}
\text{Vertex set } (K) & \{[1],[2],[6],[9],[10],[11]\} \\
\text{Edge set } (R_K) & \{([1],[2]),([2],[6]),([2],[11]),([6],[9]),([9],[10])\}
\end{array}$$

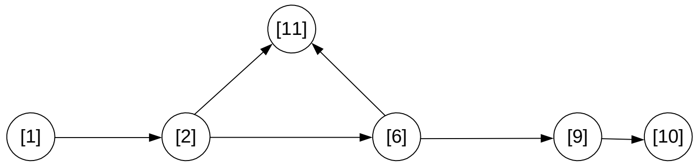


Figure 3.4: Reduced graph

Strongly connected components

$$\bigsqcup_{k \in K} G_k \sqsubseteq G$$

Table 3.2: Strongly connected components

Strongly connected component	Vertex set	Edge set
[1]	$\{1\}$	-
[2]	$\{2,3,4,5\}$	$\{(2,3),(2,4),(3,4),(4,5),(5,2)\}$
[6]	$\{6,7,8\}$	$\{(6,7),(7,8),(8,6)\}$
[9]	$\{9\}$	-
[10]	$\{10\}$	-
[11]	$\{11,12,13\}$	$\{(11,12),(12,13),(13,11)\}$

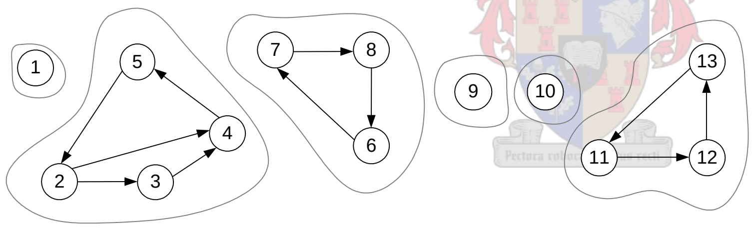


Figure 3.5: Strongly connected components

3.4.3 Acyclic graphs

3.4.3.1 Directed acyclic graph

A directed acyclic graph $G = (V; R)$ is asymmetric and does not contain cycles. Every path from a vertex x to a vertex y is elementary. The closure R^+ is asymmetric and transitive. Hence it is a strict order relation. The theoretical foundations of strict order relations may therefore be applied to directed acyclic graphs.

3.4.3.2 Rank

Every vertex x of a directed acyclic graph $G = (V; R)$ is assigned a rank $r(x)$, which is a natural number with the following properties:

1. A vertex x has the rank $r(x) = 0$ if it does not have any ancestors.

2. A vertex x has the rank $r(x) = k > 0$ if it has a k^{th} ancestor and no $(k+1)^{th}$ ancestors.

It is only possible to assign ranks if the directed graph G is acyclic. If there is a cycle through the vertex x, then for every k^{th} ancestor of x in the cycle there is a predecessor in the cycle, and hence also a $(k+1)^{th}$ ancestor of x. The directed graph must therefore be free of cycles.

If the rank $r(x)$ of a vertex x is k, then by definition the vertex x has a k^{th} ancestor but no $(k+1)^{th}$ ancestor. Thus there must be a path of length k but no path of length $k+1$ from a vertex without predecessor in G to x. Hence the rank $r(x)$ is the length k of a longest path from a vertex without predecessor in G to x.

3.4.3.3 Topological Sorting

The determination of the ranks of the vertices of a directed graph $G = (V; R)$ is called topological sorting. The vertex set $V = V_0$ is topologically sorted by iteratively reducing it to the empty vertex set $\varnothing$. In step k, the vertex set V_k is determined whose vertices $x \in V_k$ have a k^{th} ancestor in G and are therefore of rank $r(x) \geq k$. The vertex set V_k contains all predecessors of the vertices in the vertex set V_{k-1}. This iterative reduction is formulated as follows using unary relations:

$$\begin{array}{llllll} \text{initial values} & : & v_0 & = & e & \text{all relation} \\ \text{reduction} & : & v_k & = & R^T v_{k-1} & k = 1, ..., n \\ \text{termination} & : & v_n & = & \varnothing & \text{null relation} \end{array} \tag{3.31}$$

A vertex x of the vertex set V_k is of degree $r(x) = k$ if it does not belong to the vertex set V_{k+1}. The set W_k of all vertices of rank k is therefore of rank k is therefore the difference $V_k - V_{k+1}$, which is calculated as the intersection of V_k and the complement of V_{k+1}. It is called the k^{th} vertex class and is determined as a unary relation as follows:

$$w_k = v_k \sqcap \overline{v}_{k+1} \qquad k = 0, ..., n-1 \tag{3.32}$$

3.4.3.4 Order structure

Topologically sorting a directed acyclic graph $G = (V; R)$ yields a partition of the vertex set into disjoint vertex classes W_k with $k = 0, ..., n-1$. The partition has the following ordinal properties:

- The vertex class W_0 contains all vertices of the lowest rank 0. These vertices have no ancestors in G, and hence no predecessors. They are therefore minimal. Since there are no other vertices without predecessors, W_0 contains all minimal vertices.

- The vertex class W_{n-1} contains all vertices of the highest rank $n-1$. These vertices have no descendants in G, and hence no successors. They are therefore maximal. Since there may generally also be other vertices without successor, W_{n-1} generally does not contain all maximal vertices.

- Every vertex x in the vertex class W_k with $k > 0$ has at least one predecessor y in the vertex class W_{k-1}. If $x \in W_k$ did not have a predecessor $y \in W_{k-1}$, then x would not have any k^{th} ancestors, and would therefore not belong to W_k.

- A vertex has neither a predecessor nor a successor in its own vertex class. If y were a predecessor of x and hence x a successor of y, then the rank of y would have to be less than the rank of x and x, y could not belong to the same vertex class.

3.4.3.5 Basic edges and chords

A directed acyclic graph $G = (V; R)$ has basic edges and chords. An edge (x, y) in the directed graph G is called a basic edge if y is reachable from x only via this edge. If the basic edge is removed, then y is no longer reachable from x.

An edge (x, y) in the directed graph G is called a chord if the vertex y is also reachable from the vertex x via other edges. The chord (x, y) is the shortest path from x to y.

Since a directed acyclic graph does not contain cycles, an edge from x to y is a chord if and only if there is a path of length $n > 1$ from x to y.

$$\begin{array}{lll} \text{path from } x \text{ to } y \text{ with } n > 1 & \Leftrightarrow & xy^T \sqsubseteq \bigsqcup_{n>1} R^n = R \bigsqcup_{n>0} R^n = RR^+ \\ \text{chord } (x, y) & \Leftrightarrow & xy^T \sqsubseteq R \sqcap RR^+ \\ \text{basic arc } (x, y) & \Leftrightarrow & xy^T \sqsubseteq R \sqcap \overline{RR^+} \end{array} \tag{3.33}$$

3.4.3.6 Basic path

A directed acyclic graph $G = (V; R)$ does not contain cycles. If there are one or more paths from x to y, then there is at least one path of maximal length. A path of maximal length is called a basic path. A basic path contains only basic edges.

3.4.3.7 Basic graph

The graph $B = (V; Q)$ is a basic graph of a directed acyclic graph $G = (V; R)$ if Q contains only the basic edges in R. The basic graph B is constructed by removing all chords from R. The basic graph B is unique. The transitive closures $R^+$and Q^+ are equal.

$$B = (V; Q) \quad \text{with} \quad Q = R \sqcap \overline{RR^+} \tag{3.34}$$

3.4.3.8 Order diagram

In the topological sorting of a directed acyclic graph $G = (V; R)$, the rank $r(x)$ of a vertex $x \in V$ is equal to the length of a longest path from a vertex without predecessor to x. This path is a basic path consisting only of basic edges. Hence removing chords from R does not change the rank $r(x)$ of a vertex x, so that topologically sorting the graph $G = (V; R)$ and its basic graph $B = (V; Q)$ leads to the same result. The representation of the order structure of the basic graph with its vertex classes is an order diagram. This will be discussed further and demonstrated in Section 4.5.

Chapter 4

Strongly connected components and the logical sequence of tasks

4.1 Introduction

In order to determine the logical sequence of tasks (see Section 4.5), we need to sort the set of tasks topologically with the relation "has to be executed before". Only acyclic directed graphs can be sorted topologically. Since the graph of the "has to be executed before" relation in the set of tasks is not necessarily acyclic, we need to decompose it into its strongly connected components. The decomposition of a directed graph into its strongly connected components leads to the reduced graph, which is acyclic and can be sorted topologically. The algorithm for decomposing a graph into its strongly connected components described in Section 3.4.2.8 requires the calculation of the transitive closure of the graph. This is a very expensive exercise. Therefore, we need to find a more efficient way of determining strongly connected components.

A depth-first search can be used to find more information about the structure of the graph, such as the strongly connected components of a directed graph. Each vertex and edge in a directed graph is visited once during a depth-first search (see Section 4.3). This leads to a forest of rooted trees (see Section 4.2). Algorithms for finding the strongly connected components such as Kosaraju's (see Section 4.4.1), Tarjan's (see Section 4.4.2) and Gabow's (see Section 4.4.3) algorithms are based on the depth-first search algorithm. Each strongly connected component is a rooted graph.

The goal of topological sorting is to be able to process the vertices of a directed acyclic graph in such a way that each vertex is processed before all its successors. Vertices can be sorted into steps, where all the vertices in one step have to be processed before the vertices in the next step. The topological sorting algorithm described in Section 3.4.3.3 is very expensive. Therefore, we will be considering an alternative algorithm in Section 4.5.

See reference [5] for a description of the depth-first search and topological sorting algorithms. Kosaraju's, Tarjan's and Gabow's algorithms are also described in reference [5].

4.2 Rooted graphs and rooted trees

4.2.1 Introduction

A vertex of a graph from which all remaining vertices are reachable is called a root of the graph. All hierarchical structures are regarded as rooted trees. Searching for all vertices of a graph which are reachable from a given vertex leads to a search tree which corresponds to a rooted tree and forms a skeleton of the graph.

4.2.2 Root

A vertex w is called a root (root vertex) of a directed graph $G = (V; R)$ if all vertices of the graph are reachable from the vertex w. If a directed graph is not weakly connected, then it has no root. If it is strongly connected, then every vertex of the graph is a root.

$$w \text{ is a root } :\Leftrightarrow we^T \sqsubseteq R^* \tag{4.1}$$

where e is the all relation and R^* the reflexive transitive closure.

4.2.3 Rooted graph

A directed graph $G = (V; R)$ is called a rooted graph if it contains at least one root. In a rooted graph, there is a special form of connectedness between pairs of vertices, called quasi-strong connectedness. Two vertices x and y are quasi-strongly connected if there is a vertex z from which the vertices x and y are both reachable. In this case, there is a path from x to z in the dual graph G^T and a path from z to y in the graph G, so that $(x, z) \in R^{*T}$ and $(z, y) \in R^*$, and hence $(x, y) \in R^{*T}R^*$. In a rooted graph, all vertices are pairwise quasi-strongly connected via a root, so that $R^{*T}R = E$ holds.

$$\begin{array}{lll} x \text{ and } y \text{ are quasi-strongly connected} & :\Leftrightarrow & xy^T \sqsubseteq R^{*T}R^* \\ G = (V; R) \text{ is a rooted graph} & :\Leftrightarrow & R^{*T}R^* = E \end{array} \tag{4.2}$$

where E is the all relation and R^* is the reflexive transitive closure.

4.2.4 Acyclic rooted graph

A directed graph $G = (V; R)$ is acyclic if $R^+ \sqcap R^{+T} = \phi$ holds. It is a rooted graph if $R^{*T}R^* = E$ holds. An acyclic rooted graph has exactly one root. The existence of several roots would contradict the absence of cycles.

$$G = (V; R) \text{ is an acyclic rooted graph } \Leftrightarrow R^+ \sqcap R^{+T} = \phi \wedge R^{*T}R^* = E \tag{4.3}$$

where E is the all relation and R^+ is the transitive closure.

4.2.5 Rooted tree

An acyclic rooted graph $G = (V; R)$ is called a rooted tree if R is left-unique, so that $RR^T \sqsubseteq I$ holds.

$$G = (V; R) \text{ is a rooted tree } :\Leftrightarrow RR^T \sqsubseteq I \wedge R^+ \sqcap R^{+T} = \phi \wedge R^{*T}R^* = E \tag{4.4}$$

where E is the all relation, R^+ the transitive closure, R^* the reflexive transitive closure and I the identity relation.

A rooted tree with the root w has the following properties:

- The root w has no predecessor.
- Every vertex $x \neq w$ has exactly one predecessor.
- Every vertex $x \neq w$ is reachable along exactly one path from w to x.
- A rooted tree with n vertices has exactly $n-1$ edges.

4.2.6 Forest of rooted trees

A directed graph is called a forest of rooted trees if every weakly connected component is a rooted tree.

4.2.7 Search tree

Let a vertex a in a directed graph G be given. A rooted tree with root a which contains all descendants of a in G is called a search tree at the vertex a. A search tree is constructed by an iterative search, starting from the vertex a. Breadth-first search and depth-first search are distinguished.

4.3 Depth-first search

4.3.1 Trees and forests

The vertices and some of the edges of a directed graph form a depth-first search tree during the depth-first search. The depth-first search tree is a representation of the order in which the vertices had been visited. Only edges pointing to previously unvisited vertices are part of a depth-first search tree. Therefore, each depth-first search tree is a directed acyclic subgraph of the directed graph. Depth-first search trees for directed graphs are rooted trees. The number of depth-first search trees formed during a depth-first search depends on the order in which the vertices are visited, as well as the structure of the graph. If more than one depth-first search tree is formed, we have a depth-first search forest. Different depth-first searches, with different depth-first search forests can be done on the same graph, depending on the order in which vertices are visited.

4.3.2 Pre- and post-order numbering

During the depth-first search, pre- and post-order numbers are assigned to each vertex. The pre-order numbers indicate the order in which the vertices are first visited, while the post-order numbers indicate the order in which vertices are finished with in the depth-first search.

4.3.3 Classification of edges

The edges of a directed graph can be classified into four groups during a depth-first search. The classification of an edge is a property of both the structure of the graph and the dynamics of the search. Since there is more than one depth-first search forest for each graph, different classifications may be given to an edge of a graph for different depth-first searches. The pre- and post-order numbers are used to classify the edges.

Tree edges correspond to a recursive call in the depth-first search, i.e. the start vertex has been visited, but the end vertex has not been visited before. Tree edges are the edges of the depth-first search trees. The other types of edges are not part of the depth-first search tree. (Start vertex pre-order number $= -1$.)

Back edges indicate that the directed graph contains at least one cycle. The number of back edges does not necessarily correspond to the number of cycles in the directed graph. The start vertex of a back edge has been visited previously. The end vertex has also been visited previously, and is also an ancestor of the start vertex in the depth-first search tree. The removal of all the back edges results in a directed acyclic graph. (End vertex pre-order number $= -1$.)

Down edges The start vertex of a down edge points to a previously visited end vertex, which is a descendent of the start vertex in the depth-first search tree. Down edges are also known as chords (see Section 3.4.3.5) in the directed graph. (Start vertex pre-order number $>$ end vertex pre-order number.)

Cross edges The start vertex of a cross edge, points to a previously visited end vertex, which is neither an ancestor nor a descendent of the start vertex in the depth-first search tree. Cross edges connect vertices in different depth-first search trees (If it is not a tree, back or down edge.)

Example

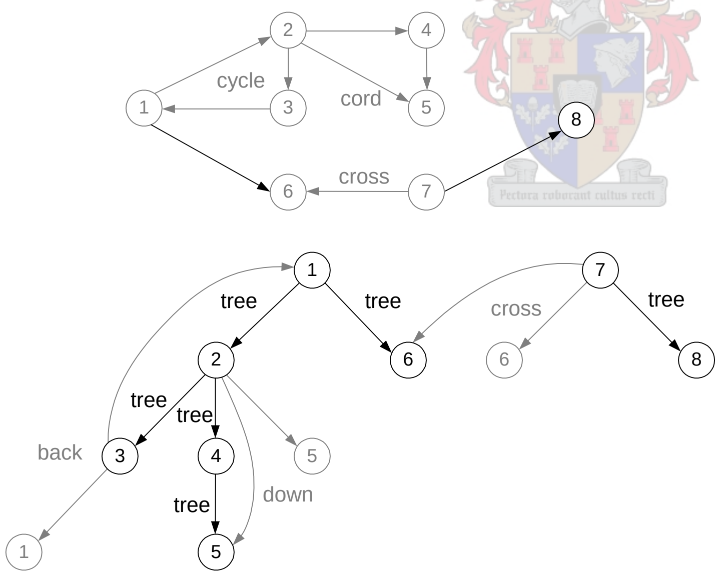


Figure 4.1: Edge classifications

The edge classifications can be seen visually in Figure 4.1. The back edge $(3, 1)$ is an indication of a cycle in the graph, in this case cycle $(1, 2), (2, 3), (3, 1)$. A down edge is an indication of a chord in the graph, in this case, if the chord $(2, 5)$ is cut, vertex 5 will still be reachable from vertex 2, via vertex 4. A cross edge points from a vertex in one depth-first search tree, vertex 7, to a vertex in another depth-first search tree, vertex 6.

4.3.4 Depth-first search algorithm

In a depth-first search, a vertex sequence F is maintained. As long as the vertex sequence F is not empty, the following steps are carried out in a loop:

- If the vertex at the end of F has a successor which has not been visited yet, such a successor is appended to the end of the sequence F.
- If the vertex at the end of F has no successor which has not been visited yet, it is removed from the sequence F.

The vertices visited and the edges used in the course of the depth-first search form the depth-first search tree. An unvisited vertex is chosen as the start vertex. If all the vertices have not been visited at the end of the process, a remaining unvisited vertex is chosen and a new vertex sequence is maintained. The process is repeated until all the vertices have been visited. A depth-first search tree is formed for each sequence. The depth-first search trees form a depth-first search forest.

4.3.5 Depth-first search example

The graph in Figure 4.2 will be used to demonstrate a depth-first search.

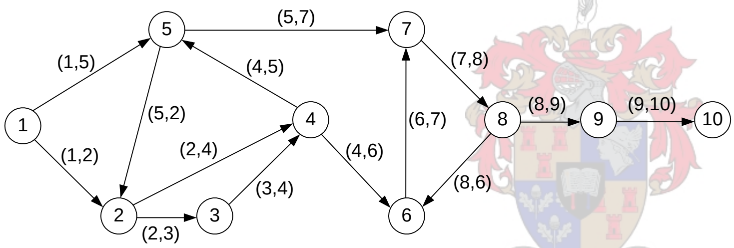


Figure 4.2: Graph example

The directed graph consists of ten vertices, labelled $1, \ldots, 10$ and 14 edges, labelled $(1,2), (1,5), (2,3), \ldots$.

Vertex 7 is chosen as the first unvisited vertex, giving it a pre-order number of 1. Vertex 7 has only one successor, vertex 8. Vertex 8 is still unvisited and is chosen as the next unvisited vertex. It is given a pre-order number of 2 and the edge $(7,8)$ is classified as a tree edge. Vertex 8 has two successors, vertices 6 and 9. Vertex 6 is randomly chosen as the next unvisited vertex and given a pre-order number of 3 and edge $(8,6)$ classified as a tree edge. Vertex 9 will be considered at a later stadium. The successors of vertex 6 will be considered first. Vertex 6 has only one successor, vertex 7, which has been visited previously. Vertex 7 still has no post-order number, which indicates it as an ancestor of vertex 6. Therefore, edge $(6,7)$ is classified as a back edge. The presence of a back edge is an indication of a cycle. Therefore, the directed graph under consideration is not a directed acyclic graph. Since vertex 6 has no other successors, we leave it, giving it a post-order number of 1. Vertex 9, the remaining successor of vertex 8, is considered next.

After vertex 7 and all its ancestors had been processed, a new random unvisited vertex, vertex 2, is chosen. Vertex 2 is the root of the second tree in the depth-first search forest. After the depth-first search has been completed, the pre-order and post-order numbers shown in Table 4.1 were given to the vertices.

Table 4.1: Pre-order and post-order numbers

	1	2	3	4	5	6	7	8	9	10
pre-order no.	10	6	7	8	9	3	1	2	4	5
post-order no.	10	9	8	7	6	1	5	4	3	2

The depth-first search forest, edge classifications, as well as the search path, can be seen in Figure 4.3.

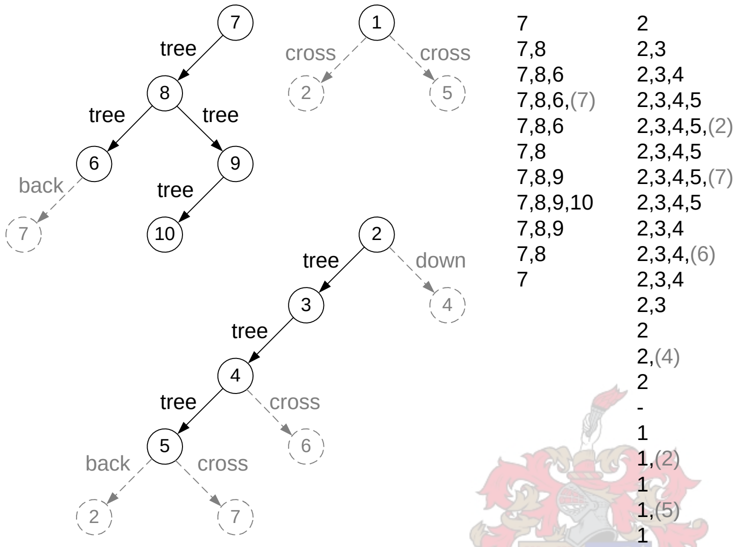


Figure 4.3: Depth-first search forest

The black vertices and edges indicate the depth-first trees. The vertices and edges in broken lines are not a part of the depth-first search trees and are only indicated to display the detection of back, down and cross edges. The first tree consists of vertices 6, 7, 8, 9 and 10. The second tree consists of vertices 2, 3, 4 and 5, while the third tree consists only of one vertex, vertex 1.

One of the many other depth-first search forests for the graph and its search path is shown in Figure 4.4. The pre-order and post-order numbers for this search shown in Table 4.2.

Table 4.2: Pre-order and post-order numbers for alternative depth-first search

	1	2	3	4	5	6	7	8	9	10
pre-order no.	6	7	8	9	10	3	1	2	4	5
post-order no.	10	9	8	7	6	1	5	4	3	2

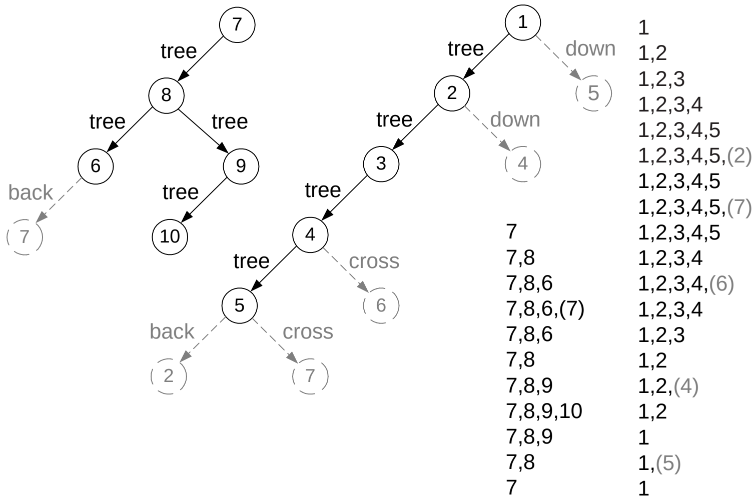


Figure 4.4: Depth-first search forest for alternative depth-first search

The first tree consists of vertices 6, 7, 8, 9 and 10, the second tree consists of vertices 1, 2, 3, 4 and 5.

4.4 Decomposition into strongly connected components

4.4.1 Kosaraju's algorithm

4.4.1.1 Description

Kosaraju's algorithm is the simplest to explain and implement. To find the strongly connected components of a directed graph, first do a depth-first search on its inverse. The inverse of a directed graph is the graph in which the directions of the edges have been reversed. Then do depth-first searches on the graph, each time starting at the next unvisited vertex with the highest post-order number.

The trees in the resulting depth-first search forest define the strongly connected components of the directed graph, since two vertices belong to the same strongly connected component if and only if they belong to the same tree in the depth-first search forest.

This is proved as follows:

If two vertices s and t are mutually reachable, they will be in the same depth-first search tree because when the first of the two is visited, the second is unvisited and is reachable from the first and so will be visited before the recursive call for the root terminates. To prove the converse, we assume that s and t are in the same tree, and let r be the root of the tree. The fact that s is reachable from r, through a directed path of tree edges, implies that there is a directed path from s to r in the inverse directed graph. Now, the key to the proof is that there must also be a path from r to s in the inverse directed graph, because r has a higher post-order number than s, since r was chosen first in the second depth-first search at a time when both were unvisited, and there is a path from s to r. If there were no path from r to s, then the path from s to r in the inverse would leave s with a higher post-order number. Therefore, there are directed paths from s to r and from r to s in the directed graph and its inverse: s and r are strongly connected. The same argument proves that t and r are strongly connected, and therefore s and t are strongly connected

4.4.1.2 Implementation

Essentially, the depth-first search implementation described in Section 4.3 is used for Kosaraju's algorithm. A few minor changes are made, however;

(1) to do a depth-first search on the inverse of the graph, instead of the graph,

(2) to use a list of sorted vertices to choose the next unvisited vertex in the depth-first searches, and

(3) to allocate strongly connected component numbers to the vertices during the depth-first searches.

4.4.1.3 Example

The graph in Figure 4.2 will now be used to demonstrate the decomposition of a directed graph into its strongly connected components.

First, a depth-first search is done on the inverse of the graph, shown in Figure 4.5. The post-order numbers in Table 4.3 were determined for the depth-first search shown in the Figure 4.6.

Table 4.3: Post-order numbers

	1	2	3	4	5	6	7	8	9	10
post-order no.	1	2	3	4	5	6	7	8	9	10

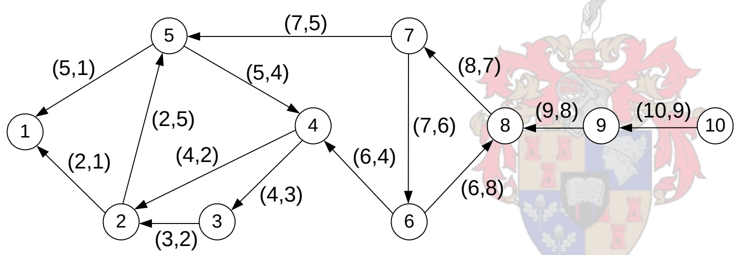


Figure 4.5: Inverse of graph example

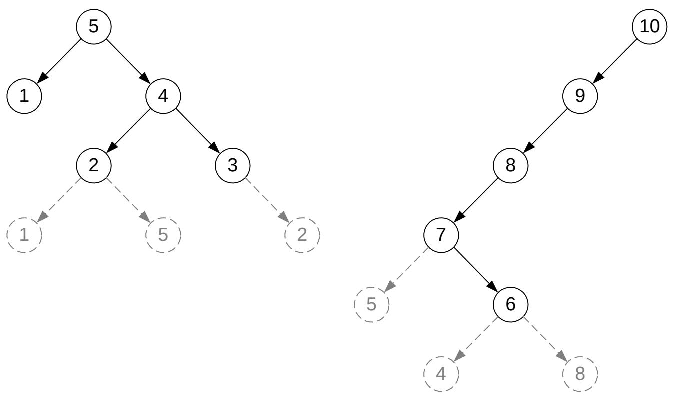


Figure 4.6: Depth-first search forest of inverse graph

Next, a depth-first search is done on the original graph, choosing the next unvisited vertices in the inverse post-order numbering, i.e. starting with vertex 10.

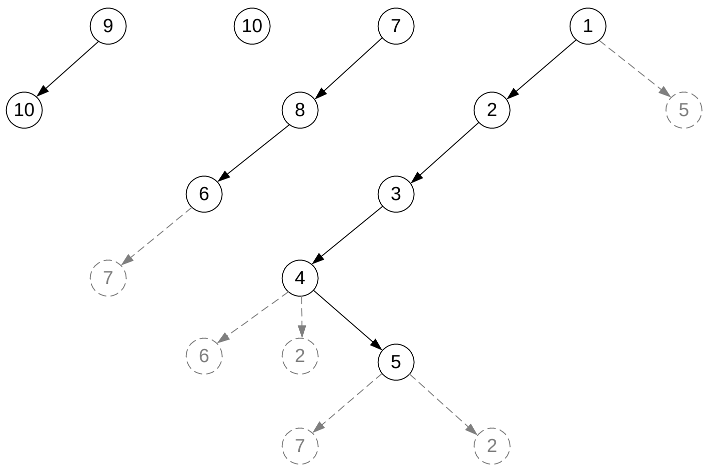


Figure 4.7: Depth-first search forest of graph

Each depth-first search tree of the depth-first search forest represents a strongly connected component. Therefore, this graph can be decomposed into five strongly connected components, three of which are single vertices.

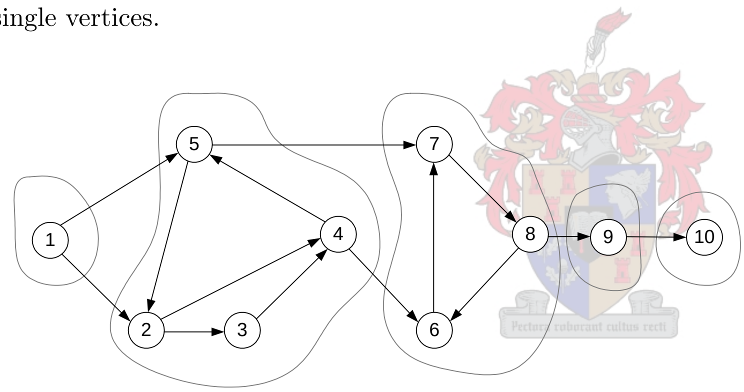


Figure 4.8: Strongly connected components

4.4.2 Tarjan's algorithm

4.4.2.1 Description

Tarjan's algorithm is based on two observations.

- First, we consider the vertices in the reverse order in which they are discovered, since we know we will not encounter any more vertices in the same strongly connected component, because all the vertices that can be reached from a given vertex has been processed already.
- Second, the back edges of the graph provide a second path from one vertex to another and bind together the strongly connected components.

An augmented depth-first search algorithm is used in the sense that a different vertex sequence is maintained. The vertices are enqueued as they are reached by tree edges. The vertices belonging to the same strongly connected component are removed and assigned a strongly connected component number after the final member of the strongly connected component has been enqueued.

The algorithm is based on our ability to identify the moment a strongly connected component has been found with a simple test. The depth-first search method finds the highest vertex reachable, via a back edge, from any descendant of each vertex. The pre-order numbers of these vertices are assigned as the low numbers for each vertex.

The pre-order and low numbers of the vertices are used to identify a strongly connected component. If a vertex's pre-order and low numbers are equal at the end of the recursive procedure, that tells us that all vertices encountered since entry (except those already assigned to a component) belong to the same strongly connected component.

4.4.2.2 Example

The graph used in Section 4.4.1 is also used to demonstrate Tarjan's algorithm. The main vertex sequence is shown in Figure 4.9.

Main vertex sequence:	Secondary vertex sequence:
-	-
9	9
9,**10**	9,**10**
9	**9**
-	-
7	7
7,8	7,8
7,8,6	7,8,6
7,8,6	**7**
-	-
1	1
1,2	1,2
1,2,3	1,2,3
1,2,3,4	1,2,3,4
1,2,3,4	1,2,3,4
1,2,3,4	1,2
1,2,3,4	1,2,5
1,2,3,4,5	1,2,5
1,2,3,4,5	1,**2**
1	**1**
-	-

Figure 4.9: Main and secondary vertex sequences

The pre-order and low numbers used to determine the strongly connected components for Tarjan's algorithm are shown in Table 4.4.

Table 4.4: Pre-order and low numbers

	1	2	3	4	5	6	7	8	9	10
pre-order number	6	7	8	9	10	5	3	4	1	2
low number	6	7	7	7	7	3	3	3	1	2
strongly connected component	5	4	4	4	4	3	3	3	1	2

4.4.3 Gabow's algorithm

4.4.3.1 Description

Gabow's algorithm enqueues the vertices in the same way as Tarjan's algorithm does, but it uses a second vertex sequence, instead of the pre-order and low numbers, to decide when to remove all the vertices in each strongly connected component from the main sequence.

The second vertex sequence contains vertices on the search path. When a back edge shows that a sequence of such vertices all belong to the same strongly connected component, we remove that vertex sequence from the secondary vertex sequence to leave only the destination vertex of the back edge, which is nearer the root of the tree than are any of the other vertices. After processing all the edges for each vertex (making recursive calls for the tree edges, removing vertices from the secondary vertex sequence for the back edges, and ignoring the down edges), we check to see whether the current vertex is at the top of the secondary vertex sequence. If the current vertex is at the top of the secondary vertex sequence, the current vertex and all the vertices above it on the main vertex sequence make a strongly connected component, and we remove them and assign the next strongly connected component number to them, as we did in Tarjan's algorithm.

4.4.3.2 Example

The graph used in Section 4.4.1 is also used to demonstrate Gabow's algorithm. The main and secondary vertex sequences is shown in Figure 4.9. The coinciding vertices on the main and secondary vertex sequences, used to determine the strongly connected components for Gabow's algorithm are shown in bold on the vertex sequences.

4.5 Logical sequence of tasks through topological sorting

4.5.1 Topologically sorting a directed acyclic graph by removing sources

The goal of topological sorting is to be able to process the vertices of a directed acyclic graph in such a way that each vertex is processed before all its successors. Vertices can be sorted into logical steps, where all the vertices in one logical step have to be processed before the vertices in the next logical step.

This topological sorting algorithm is based on the property that a directed acyclic graph has at least one source and one sink. A source is a vertex with an in-degree (see Section 3.4.1.2) of 0, i.e. with no entering edges or predecessor vertices. A sink is a vertex with an out-degree of 0, i.e. with no leaving edges or successor vertices.

There may be multiple sources, so we need to keep track of them. A set of sources will be used for this purpose. The logical steps into which the vertices are sorted can be labelled with positive integers.

4.5.1.1 Algorithm

- Find all the sources in the directed acyclic graph. All the sources are added to the sources set. These vertices are all part of the next smallest unused logical step in the topological sorting. After a source has been processed, it is marked and removed from the sources set.

- Find all the successors of the sources and process them next. Decrement their in-degree. All the successors whose predecessors have all been processed already, i.e. their in-degree is now 0, become the new sources.

- Repeat the previous steps until all vertices have been processed.

This algorithm adds each vertex to the earliest possible logical step. It is possible for the vertices to be added to a later logical step, while still respecting the requirements for a topological sorting. Therefore, there may be different topological sorts for the same directed acyclic graph.

4.5.1.2 Topological sorting example

The graph in Figure 4.10 will be used to demonstrate the topological sorting algorithm.

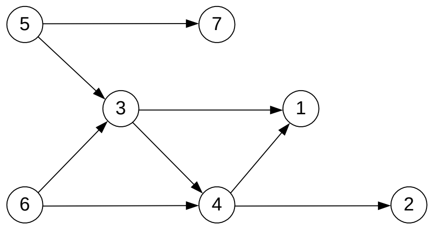


Figure 4.10: Topological sorting graph example

The graph is a directed acyclic graph and can therefore be sorted topologically. It has two sources, vertices 5 and 6, which have no entering edges or predecessors. These vertices are identified and added to the set of sources. After these vertices had been added to the first step, they are marked as processed and removed from the set of sources.

The successors of these sources are vertices 3, 4 and 7. Since vertices 5 and 6, which have already been processed, are the only predecessors of vertex 3, vertex 3 becomes a source, similarly vertex 7. However, for vertex 4, only one of its predecessors, vertex 6, has already been processed. Its other predecessor, vertex 3, has not been processed, and therefore vertex 4 is not a source yet. The new sources (vertices 3 and 7) are added to step 2, marked as processed and removed from the set of sources. This process has to be continued until all the vertices have been processed.

After doing a topological sort, four steps have been identified. These are shown in Figure 4.11, along with another possible topological sort. Vertex 7 could also have been added to step 3.

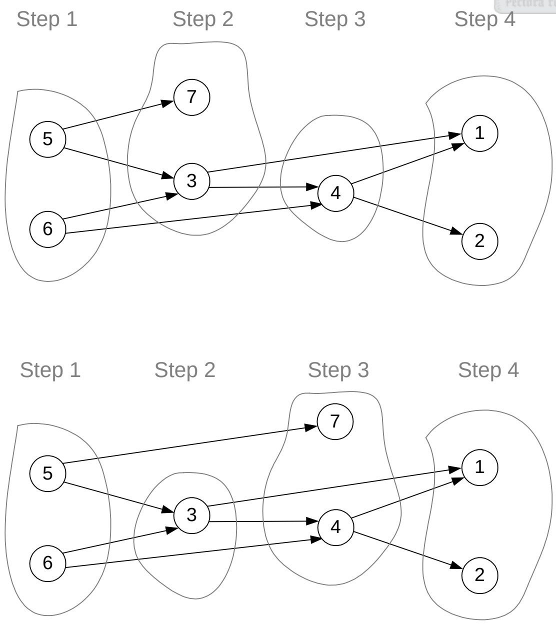


Figure 4.11: Graph, rearranged after topological sorting

4.5.2 Graphical representation of the logical sequence of tasks

The topologically sorted set of vertices of the example in Section 4.5.1.2 can be displayed graphically as shown in Figure 4.12.

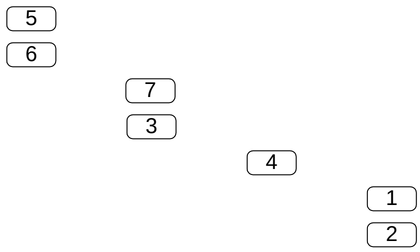


Figure 4.12: Graphical representation of the sequence of tasks

Chapter 5

Path algebras and methods of solution

5.1 Introduction

Tasks in the engineering process are dependent on other tasks. Some tasks have to be finished before other tasks can be started. The "has to be execute before" relation in the set of tasks can be used to determine task sequences. Such task sequences are paths (see Section 3.4.1.5) in the task-task graph. The question of determining all the elementary paths (see Section 3.4.1.13) in the graph leads to a literal path algebra for elementary paths (see Section 5.3.3).

Critical path analysis identifies tasks which must be completed on time for the whole project to be completed on time, and also which tasks can be delayed for a while if resources need to be reallocated to catch up on missed tasks. The logical critical path (see Section 5.4) is a longest elementary path through a directed acyclic graph.

The use of path algebras reduces the solution of path problems, such as the literal path algebra for elementary paths, to the solution of systems of equations (see Section 5.2.12). Therefore, we need methods to solve these systems of equations. The bigger the graphs, the more complex the structure becomes and the number of possible paths grows very fast. A great number of calculations has to be done during the solution of the systems of equations.

Different methods of solution (see Section 5.5) will be considered and compared to ultimately find the most efficient one. The solution of the system of equations may be determined directly, i.e. through Gauss elimination (see Section 5.5.2.5), or iteratively, through Jacobi (see Section 5.5.3.4) or Gauss-Seidel's (see Section 5.5.3.5) methods or through the forward and back substitution method (see Section 5.5.3.6). Due to the sequential structure of our graphs, we can reduce the number of calculations that has to be done. A topological sorting (see Section 3.4.3.3) of the graph can lead to the relabelling of the vertices (see Section 5.6) in order to change the adjacency matrix into a partially upper triangular matrix. Knowledge of the upper triangular part of the adjacency-matrix can be used to reduce the number of calculations for the methods of solutions of path algebras (see Section 5.5).

See reference [4] for a detailed description of the elementary path algebra, as well as the methods of solution of Gauss, Jacobi, Gauss-Seidel and the forward and back substitution method.

5.2 Path algebras

5.2.1 Network

Let a directed graph be given. Let a weight be associated with each edge of the directed graph. A directed graph with edge weights is called a weighted graph or a network. The form and meaning of the edge weights depends on the application. For example, every edge in a network may be weighted by a real number which represents a length or by a character serving as a label.

5.2.2 Path problem

The determination of paths with specific properties in networks is called a path problem. Different path problems can be formulated for different applications. A general distinction is made between structure problems and extreme value problems.

Structure problems, specific structural properties of paths between two vertices in a network are determined. Examples include determining the existence of paths, determining simple or elementary paths and determining common edges or intermediate vertices of all paths between two given vertices. In the problem under consideration, determining the logical critical path in the task-task graph is a structural problem.

Extreme value problems, minimal or maximal properties of paths between two vertices in a network are determined. An example of a path problem with maximal properties include determining the length of the longest or shortest path between vertices in a road network.

5.2.3 Path and weights

A path from i to k is an edge sequence with start vertex i and end vertex k. The path is said to be weighted if it is associated with a weight determined from the weights of its edges according to a given rule. Different path problems involve different rules for assigning weights to paths. For example, the length of a path may be determined as the sum of the lengths of its edges, while the label of a path may be determined as the concatenation of the labels of its edges.

On the basis of the algebra of sets and the literal algebra, the binary operations of union (symbol $\sqcup$) and concatenation (symbol $\circ$) are defined for the set of weights. The operations for the weights are generally different from the operations for the path sets and depend on the path problem considered.

5.2.3.1 Alphabet and words

A finite character set is called an alphabet and is designated by $\mathbb{A}$. A finite character string with zero, one or several characters is called a word. The character string without characters is called the empty word and is designated by λ. The set of all words including the empty word λ is designated by $\mathbb{A}^*$.

Two words $a, b \in \mathbb{A}^*$ are concatenated to form a single word by appending the character string of the second word to the character string of the first word. The concatenation $\circ$ is an associative operation in the set $\mathbb{A}^*$ with the empty word λ acting as the unit element.

$$\begin{array}{lrcl} \text{associative} & a \circ (b \circ c) & = & (a \circ b) \circ c \\ \text{unit element} & a \circ \lambda = a & = & \lambda \circ a \end{array} \tag{5.1}$$

5.2.3.2 Edge and path labels

Every edge of a graph is labeled by a character from an alphabet $\mathbb{A}$. The literal labeling of the edges is said to be unique if any two different edges are labeled by different characters. If the edge labels are unique, the character for an edge also serves as an edge identifier. Edge labels are assumed to be unique in the formulation of path algebras.

Every path in a graph is an edge sequence and is labeled by a character string, which is a word in the set $\mathbb{A}^*$ of words. If the edges are labeled uniquely, then the paths are also labeled uniquely. A path without edges from a vertex k to the same vertex k is labeled by the empty word λ.

5.2.4 Path set and weighted path set

5.2.4.1 Path sets

A set of paths with common start vertex i and common end vertex k is called a path set.

Let a directed graph with unique edge labels be given. A set of paths for a vertex pair (i, k) in the graph is called a complete path set and is designated by W_{ik} if it contains all paths from vertex i to vertex k. A subset a_{ik} of the complete path set W_{ik} is called a path set. The set of all possible subsets a_{ik} of W_{ik} is the power set of the complete path set and is designated by $P\,(W_{ik})$. Every path set a_{ik} is an element of the power set $P\,(W_{ik})$, that is $a_{ik} \in P\,(W_{ik})$.

The zero set, the unit set and the elementary path set are special path sets. The path set which contains no path is called the zero set and is designated by $0_W = \{\}$. The path set which contains only the empty path without edges from a vertex i to the same vertex i is called the unit set and is designated by $1_W = \{\lambda\}$. A path set a_{ik} is said to be elementary if it contains exactly one path which consists only of the edge from vertex i to vertex k.

$$\begin{array}{llcl} \text{zero set} & 0_W & = & \{\} \\ \text{unit set} & 1_W & = & \{\lambda\} \\ \text{elementary path set} & a_{ik} & = & \{< i, k >\} \end{array} \tag{5.2}$$

The path set is said to be weighted if it is associated with a weight determined from the weights of its paths according to a given rule. Different path problems involve different rules for assigning weights to path sets. For example, in a minimum length problem the length of the shortest path in the path set is the weight of the path set.

5.2.4.2 Weighted path set

Every path set $a_{ik} \in P\,(W_{ik})$ is assigned a unique weight $z_{ik} \in Z$ from a weight set Z. The zero set 0_W is assigned the zero element 0_Z, and the unit set 1_W is assigned the unit element 1_Z. Associating the path set a_{ik} with the weight z_{ik} defines a mapping as shown below.

$$\begin{array}{llcl} \text{mapping} & f\,(a_{ik}) & = & z_{ik} \in Z \\ \text{zero element} & f\,(0_W) & = & 0_Z \in Z \\ \text{unit element} & f\,(1_W) & = & 1_Z \in Z \end{array} \tag{5.3}$$

As for path sets, the binary operations $\sqcup$ and $\circ$ are defined for the weights. The operations for weights and the operations for path sets are generally different and depends on the path problem considered. The mapping f is said to be homomorphic if the following statements hold:

$$\begin{aligned} f\left(a_{ik} \sqcup b_{ik}\right) &= f\left(a_{ik}\right) \sqcup f\left(b_{ik}\right) \\ f\left(a_{ik} \circ b_{km}\right) &= f\left(a_{ik}\right) \circ \left(b_{km}\right) \end{aligned} \tag{5.4}$$

If the mapping f is homomorphic, the weights of path sets may be determined without explicitly determining the path sets, since the following implications hold for $x_{ij} = f\left(a_{ij}\right)$, $y_{ij} = f\left(b_{ij}\right)$ and $z_{ij} = f\left(c_{ij}\right)$:

$$\begin{aligned} c_{ik} &= a_{ik} \sqcup b_{ik} &\Rightarrow\quad z_{ik} &= x_{ik} \sqcup y_{ik} \\ c_{im} &= a_{ik} \circ b_{km} &\Rightarrow\quad z_{im} &= x_{ik} \circ y_{km} \end{aligned} \tag{5.5}$$

Using these implications, the properties of path sets with the operations $\sqcup$ and $\circ$ may be transferred to the properties of weights with the operations $\sqcup$ and $\circ$. The homomorphism condition is therefore of fundamental importance for a path algebra.

5.2.5 Elementary path set matrix

Let a directed graph with n vertices be given. The path sets for all vertex pairs of the graph are arranged in an $n \times n$ matrix. An $n \times n$ matrix is called a complete path set matrix and is designated by $\mathbf{W}$ if it contains the complete path set W_{ik} for every vertex pair (i, k) of the graph. An $n \times n$ matrix is called a path set matrix $\mathbf{A}$ with $\mathbf{A} \subseteq \mathbf{W}$ if it contains a path set $a_{ik} \subseteq W_{ik}$ for every vertex pair (i, k) in the graph. The set of all possible path set matrices $\mathbf{A} \subseteq \mathbf{W}$ is called the power set of the complete path set matrix and is designated by $P\left(\mathbf{W}\right)$. A path set matrix $\mathbf{A}$ is an element of the power set $P\left(\mathbf{W}\right)$, that is $\mathbf{A} \in P\left(\mathbf{W}\right)$.

The zero matrix, the identity matrix and the elementary path set matrix are special path set matrices. A path set matrix is called a zero matrix and is designated by $\mathbf{0}_W$ if it contains the zero set 0_W for every vertex pair (i, k). A path set matrix is called an identity matrix and is designated by $\mathbf{I}_W$ if it contains the unit set 1_W for every vertex pair (k, k) and the zero set 0_W for all remaining vertex pairs. A path set matrix is said to be elementary if it contains the elementary path set a_{ik} for every vertex pair (i, k) with an edge from vertex i to vertex k and the zero set 0_W for all remaining vertex pairs. A directed graph with unique edge labels is uniquely described by the elementary path set matrix.

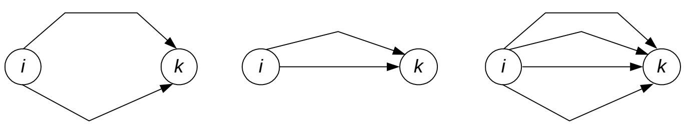
5.2.6 Elementary path weight matrix

Let a directed graph with n vertices, a weight set Z and a homomorphic mapping f be given. Then every path set matrix $\mathbf{A}$ may be mapped homomorphically to a weight matrix $\mathbf{Z}$. Every path set a_{ik} of $\mathbf{A}$ is mapped to the weight $z_{ik} = f\left(a_{ik}\right) \in Z$ of $\mathbf{Z}$. As in the case of path set matrices, the zero matrix $\mathbf{0}_Z$, the identity matrix $\mathbf{I}_Z$ and the elementary weight matrix are special weight matrices.

5.2.7 Operations in the path set

Let the path sets $a_{ik} \in P\left(W_{ik}\right)$ and $b_{ik} \in P\left(W_{ik}\right)$ be given. The path set $c_{ik} \in P\left(W_{ik}\right)$ which contains all paths which are contained in a_{ik} or in b_{ik} is called the union of a_{ik} and b_{ik}.

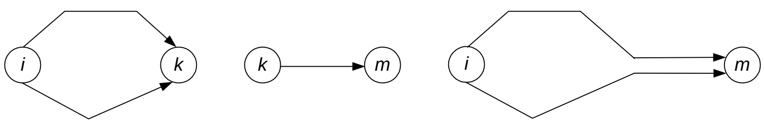
$$\text{union } \sqcup \qquad c_{ik} = a_{ik} \sqcup b_{ik} := \{x \mid x \in a_{ik} \vee x \in b_{ik}\} \tag{5.6}$$



a_{ik} $\qquad$ b_{ik} $\qquad$ $c_{ik} = a_{ik} \sqcup b_{ik}$

Let the path sets $a_{ik} \in P\left(W_{ik}\right)$ and $b_{km} \in P\left(W_{km}\right)$ be given. The path set $c_{im} \in P\left(W_{im}\right)$ which contains the paths which are formed by concatenating a path $x \in a_{ik}$ and a path $y \in b_{km}$ is called the concatenation of a_{ik} and b_{km}.

$$\text{concatenation } \circ \qquad c_{im} = a_{ik} \circ b_{km} := \{x \circ y \mid x \in a_{ik} \wedge y \in b_{km}\} \tag{5.7}$$



a_{ik} $\qquad$ b_{km} $\qquad$ $c_{im} = a_{ik} \circ b_{km}$

5.2.8 Algebraic structure

5.2.8.1 Path sets

The operations on path sets have the following properties shown in Table 5.1:

Table 5.1: Algebraic structure of path sets

Property	Union $\sqcup$			Concatenation $\circ$		
idempotent	$a_{ik} \sqcup a_{ik}$	$=$	a_{ik}			
associative	$a_{ik} \sqcup (b_{ik} \sqcup c_{ik})$	$=$	$(a_{ik} \sqcup b_{ik}) \sqcup c_{ik}$	$a_{ik} \circ (b_{km} \circ c_{mj})$	$=$	$(a_{ik} \circ b_{km}) \circ c_{mj}$
commutative	$a_{ik} \sqcup b_{ik}$	$=$	$b_{ik} \sqcup a_{ik}$			
distributive	$a_{ik} \circ (b_{km} \sqcup c_{km})$	$=$	$(a_{ik} \circ b_{km}) \sqcup (a_{ik} \circ c_{km})$	$(a_{ik} \sqcup b_{ik}) \circ c_{km}$	$=$	$(a_{ik} \circ c_{km}) \sqcup (b_{ik} \circ c_{km})$
zero element	$0_W \sqcup a_{ik}$	$=$	$a_{ik} = a_{ik} \sqcup 0_W$	$0_W \circ a_{ik}$	$=$	$0_W = a_{ik} \circ 0_W$
unit element				$1_W \circ a_{ik}$	$=$	$a_{ik} = a_{ik} \circ 1_W$

5.2.8.2 Weighted path sets

Let the path sets of a directed graph be homomorphically mapped to weights. The the domain $(Z; \sqcup, \circ)$ with the weight set Z and the binary operations $\sqcup$ and $\circ$ is a path algebra. It has the properties shown in table 5.2 for the elements $x, y, z \in Z$:

Table 5.2: Algebraic structure of weighted path sets

Property	Union $\sqcup$	Concatenation $\circ$
idempotent	$x \sqcup x = x$	
associative	$(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$	$(x \circ y) \circ z = x \circ (y \circ z)$
distributive	$x \circ (y \sqcup z) = (x \circ y) \sqcup (x \circ z)$	$(x \sqcup y) \circ z = (x \circ y) \sqcup (y \circ z)$
commutative	$x \sqcup y = y \sqcup x$	
zero element	$0_Z \sqcup x = x = x \sqcup 0_Z$	$0_Z \circ x = 0_Z = x \circ 0_Z$
unit element		$1_Z \circ x = x = x \circ 1_Z$

5.2.9 Operations

5.2.9.1 Path set matrices

Let the path set matrices $\mathbf{A}, \mathbf{B} \in P(\mathbf{W})$ for a directed graph with n vertices be given. In analogy with the algebra of relations, the binary operations of union $\sqcup$ and concatenation $\circ$ are defined. The operations $\sqcup$ and $\circ$ already defined for path sets are used for the matrix elements.

$$\begin{array}{llcll} \text{union } \sqcup & \mathbf{C} & = & \mathbf{A} \sqcup \mathbf{B} & := [a_{ik} \sqcup b_{ik}] \\ \text{concatenation } \circ & \mathbf{C} & = & \mathbf{A} \circ \mathbf{B} & := \left[\bigsqcup_{m=1}^{n} (a_{im} \circ b_{mk}) \right] \end{array} \tag{5.8}$$

For every vertex pair (i, k), the path set matrix $\mathbf{A} \sqcup \mathbf{B}$ contains the paths which are contained in the path set a_{ik} or in the path set b_{ik}. For every vertex pair (i, k), the path set matrix $\mathbf{A} \circ \mathbf{B}$ contains the paths formed by concatenating all paths in a_{im} with all paths b_{mk} for all vertices m. The concatenation of two path set matrices is also called their product.

5.2.9.2 Weight matrices

Let weight matrices $\mathbf{X}$, $\mathbf{Y}$ for the path set matrices $\mathbf{A}$, $\mathbf{B}$ of a directed graph be given. As in the case of path set matrices, the binary operations of $\sqcup$ and $\circ$ are defined by applying the operations $\sqcup$ and $\circ$ defined for weights to the matrix elements.

$$\begin{array}{llcll} \text{union } \sqcup & \mathbf{Z} & = & \mathbf{X} \sqcup \mathbf{Y} & := [x_{ik} \sqcup y_{ik}] \\ \text{concatenation } \circ & \mathbf{Z} & = & \mathbf{X} \circ \mathbf{Y} & := \left[\bigsqcup_{m=1}^{n} (x_{im} \circ y_{mk}) \right] \end{array} \tag{5.9}$$

5.2.10 Algebraic structure

5.2.10.1 Path set matrices

The algebraic structure of path sets is directly transferred to path set matrices. The domain $(P(\mathbf{W}); \sqcup, \circ)$ with the power set $P(\mathbf{W})$ of the complete path set matrix and the binary operations $\sqcup$ and $\circ$ is called a path algebra. The properties are shown in Table 5.3 for the path set matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in P(\mathbf{W})$:

Table 5.3: Algebraic structure of path set matrices

Property	Union $\sqcup$	Concatenation $\circ$
idempotent	$\mathbf{A} \sqcup \mathbf{A} = \mathbf{A}$	
associative	$(\mathbf{A} \sqcup \mathbf{B}) \sqcup \mathbf{C} = \mathbf{A} \sqcup (\mathbf{B} \sqcup \mathbf{C})$	$(\mathbf{A} \circ \mathbf{B}) \circ \mathbf{C} = \mathbf{A} \circ (\mathbf{B} \circ \mathbf{C})$
distributive	$\mathbf{A} \circ (\mathbf{B} \sqcup \mathbf{C}) = (\mathbf{A} \circ \mathbf{B}) \sqcup (\mathbf{A} \circ \mathbf{C})$	$(\mathbf{A} \sqcup \mathbf{B}) \circ \mathbf{C} = (\mathbf{A} \circ \mathbf{B}) \sqcup (\mathbf{B} \circ \mathbf{C})$
commutative	$\mathbf{A} \sqcup \mathbf{B} = \mathbf{B} \sqcup \mathbf{A}$	
zero element	$\mathbf{0}_W \sqcup \mathbf{A} = \mathbf{A} = \mathbf{A} \sqcup \mathbf{0}_W$	$\mathbf{0}_W \circ \mathbf{A} = \mathbf{0}_W = \mathbf{A} \circ \mathbf{0}_W$
unit element		$\mathbf{1}_W \circ \mathbf{A} = \mathbf{A} = \mathbf{A} \circ \mathbf{1}_W$

5.2.10.2 Weight matrix

Since path set matrices are homomorphically mapped to weight matrices, the algebraic structures of path set matrices and weight matrices and their operations $\sqcup$ and $\circ$ are compatible. In the set of all possible weight matrices for a directed graph, the zero matrix $\mathbf{0}_Z$ is the identity element for the union and the identity matrix $\mathbf{I}_Z$ is the identity element for the concatenation.

5.2.11 Closure

5.2.11.1 Elementary path set matrix

Let an elementary path set matrix $\mathbf{A}$ for a directed graph with n vertices be given. In analogy with the algebra of relations, the closure $\mathbf{A}^*$ is defined as the union of the powers $\mathbf{A}^m$ with $m \geq 0$. For every vertex pair (i,k), the power $\mathbf{A}^m$ contains all paths which lead from vertex i to vertex k and consist of exactly m edges. The power $\mathbf{A}^0$ is the identity matrix $\mathbf{I}_W$. For every vertex pair (i,k), the closure $\mathbf{A}^*$ contains all paths which lead from vertex i to vertex k. It therefore coincides with the complete path set matrix $\mathbf{W}$.

$$\mathbf{A}^* := \mathbf{I}_W \sqcup \mathbf{A} \sqcup \mathbf{A}^2 \sqcup \mathbf{A}^3 \sqcup \ldots = \mathbf{W} \tag{5.10}$$

If the power expression for the closure $\mathbf{A}^*$ does not change beyond a certain finite exponent q, the path set matrix $\mathbf{A}$ is said to be stable and the exponent q is called its stability index. For every vertex pair (i,k), the closure $\mathbf{A}^*$ of a stable path set matrix $\mathbf{A}$ with stability index q contains all paths which lead from vertex i to vertex k and consist of at most q edges. The elementary path set matrix for an acyclic graph with n vertices is stable with a stability index $q < n$, since a path in this graph consists of at most $n-1$ edges. The elementary path set matrix of a graph containing a cycle is not stable, since a path in this graph can traverse the cycle an arbitrary number of times and may hence consist of an arbitrary number of edges.

5.2.11.2 Elementary weight matrix

Let an elementary path set matrix $\mathbf{A}$ for a directed graph with n vertices be given. The elementary path set matrix $\mathbf{A}$ is assigned the elementary weight matrix $\mathbf{Z}$, which contains the weights of the edges of the graph. Since the weighting is a homomorphic mapping, the closure $\mathbf{Z}^*$ of the weight matrix may be determined directly from the union of the powers of $\mathbf{Z}$ without explicitly calculating $\mathbf{A}^*$. For every vertex pair (i,k), the closure $\mathbf{Z}^*$ contains the weight for the set of all paths which lead from vertex i to vertex k.

$$\mathbf{Z}^* = \mathbf{I}_Z \sqcup \mathbf{Z} \sqcup \mathbf{Z}^2 \sqcup \mathbf{Z}^3 \sqcup \ldots \tag{5.11}$$

If the power expression of the closure $\mathbf{Z}^*$ does not change beyond a certain exponent s, then the weight matrix $\mathbf{Z}$ is said to be stable and the exponent s is called the stability index. The weight matrix $\mathbf{Z}$ may be stable even if the path set matrix $\mathbf{A}$ is not stable.

5.2.12 Path algebra

The union $\sqcup$ and the concatenation $\circ$ are defined as binary operations for path sets and their weights. The rules for the union and concatenation of weighted path sets are formulated such that the weights are determined directly without explicitly constructing the path sets. This leads to path algebras for networks. A path algebra is said to be either boolean, real or literal if the weights of the path sets are respectively boolean, real or literal.

The path algebras for the different path problems may be generalized by abstraction. They are conveniently formulated in matrix and vector notation. Using path algebras reduces the solution of path problems to the solution of systems of equations.

5.2.12.1 System of equations for path sets

For a given vertex k, the path sets whose paths lead from each of the vertices $i = 1, \ldots, n$ to k may be read off in column k of the closure $\mathbf{A}^*$. The k^{th} column of the closure $\mathbf{A}^*$ is designated by $\mathbf{x}$, the unit vector with the unit set 1_W in row k by $\mathbf{e_k}$. If the closure $\mathbf{A}^*$ is known, then $\mathbf{x}$ is calculated as follows:

$$\mathbf{x} = \mathbf{A}^* \circ \mathbf{e_k}$$

By substituting the calculational rule for the closure $\mathbf{A}^*$, the following relationship between the elementary path set matrix $\mathbf{A}$ and the vector $\mathbf{x}$ is obtained:

$$\begin{aligned}
\mathbf{A}^* &= \mathbf{I_W} \sqcup \mathbf{A} \sqcup \mathbf{A^2} \sqcup \ldots \quad \Rightarrow \quad \mathbf{A}^* = \mathbf{A} \circ \mathbf{A}^* \sqcup \mathbf{I}_W \\
\mathbf{x} &= (\mathbf{A} \circ \mathbf{A}^* \sqcup \mathbf{I}_W) \circ \mathbf{e}_k \quad = \quad \mathbf{A} \circ (\mathbf{A}^* \circ \mathbf{e}_k) \sqcup (\mathbf{I}_W \circ \mathbf{e}_k)
\end{aligned}$$

$$\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{e}_k \tag{5.12}$$

For a given vertex i, the path sets whose paths lead from i to each of the vertices $k = 1, \ldots, n$ may be read off in row i of the closure $\mathbf{A}^*$. The transpose of row i of the closure $\mathbf{A}^*$ is designated by $\mathbf{y}$, the unit vector with the unit set 1_W in row i by $\mathbf{e}_i$. In analogy with the result for column k of $\mathbf{A}^*$, row i satisfies the following equation:

$$\mathbf{y} = \mathbf{A}^T \circ \mathbf{y} \sqcup \mathbf{e}_i \tag{5.13}$$

5.2.12.2 System of equations for weights

For a given vertex k, the weights of the path sets whose paths lead from each of the vertices $i = 1, \ldots, n$ to k may be read off in column k of the closure $\mathbf{Z}^*$. Column k of the closure $\mathbf{Z}^*$ is designated by $\mathbf{x}$, the unit vector with the unit element 1_Z in row k by $\mathbf{e}_k$. If the closure $\mathbf{Z}^*$ is known, then $\mathbf{x}$ is calculated as follows:

$$\mathbf{x} = \mathbf{Z}^* \circ \mathbf{e}_k$$

By analogy with the equations for path sets, the vector $\mathbf{x}$ is the solution of the following system of equations:

$$\mathbf{x} = \mathbf{Z} \circ \mathbf{x} \sqcup \mathbf{e}_k \tag{5.14}$$

For a given vertex i, the path sets whose paths lead from i to each of the vertices $k = 1\ldots, n$ may be read off in row i of the closure $\mathbf{Z}^*$ is designated by $\mathbf{y}$, the unit vector with the unit element 1_Z in row i by $\mathbf{e}_i$. By analogy with the result for column k of $\mathbf{Z}^*$, row i satisfies:

$$\mathbf{y} = \mathbf{Z}^T \circ \mathbf{y} \sqcup \mathbf{e}_i \tag{5.15}$$

General methods for the solution of systems of equations in a path algebra are treated later.

5.3 Literal path algebra

5.3.1 Introduction

The literal labelling of graphs is treated above. It forms the basis for literal path algebras. Literal path algebras for different path problems differ in the definition of the literal weight set and the definitions of the operations. The literal path algebras are particularly important for structure problems in graph theory, such as:

- determination of the simple paths and cycles
- determination of the elementary paths and cycles
- determination of the separating edges and vertices
- determination of the shortest or the longest paths and cycles

We are interested in finding the elementary paths in a graph. This literal path algebra will therefore be treated in detail in the next secion. Literal vertex labels are necessary in this case.

5.3.2 Literal vertex labels

Let every vertex of a directed graph be labeled by a character from an alphabet $\mathbb{A}$. Let any two vertices be labeled by different characters, so that the vertex labels are unique. Every edge of the directed graph is labeled by the characters of the start and end vertex

5.3.3 Elementary paths

5.3.3.1 Problem

Every path in the directed graph with literal vertex labels is associated with a word. In this word, the characters occur in the order in which the associated vertices occur in the path. An elementary path does not contain any vertex more than once and is therefore designated by a simple word. Let two paths in the directed graph be labeled by the words a and b. The word a can be concatenated with the word b to form a word $c = a \circ b$ only if the last character of a and the first character of b coincide. The concatenated word c is formed by appending the word b without its first character to the word a. Elementary cycles through a vertex k cannot be determined using this path algebra, since the word for such a cycle contains the character for the vertex k at the beginning and at the end and is therefore not simple. If the set of simple words is extended to include words with identical first and last characters, elementary paths including elementary cycles may be determined using this extended set of words.

5.3.3.2 Weights

Let the path set a_{ik} containing paths from vertex i to vertex k be given. Let the path set a_{ik} containing paths from vertex i to vertex k be given. The set of simple words for the elementary paths contained in a_{ik} is chosen as the weight z_{ik} of the path set a_{ik}. If the path set a_{ik} is the zero set 0_W, then $z_{ik} = 0_Z = \{\}$. If the path set a_{ik} is the unit set 1_W, then $z_{ik} = 1_W = \{\lambda\}$ with the empty word λ. If the path set a_{ik} is neither the zero set 0_W nor the unit set 1_W, then z_{ik} is a set of simple words. Let the set of all simple words over the alphabet $\mathbb{A}$ including the empty word λ be $\mathbb{S}$. Then z_{ik} is a subset of $\mathbb{S}$, and hence an element of the power set $P(\mathbb{S})$. Thus the weight mapping is defined as follows:

$$\begin{array}{lclcll} f(0_W) & = & 0_Z & = & \{\} & \\ f(1_W) & = & 1_Z & = & \{\lambda\} & \\ f(a_{ik}) & = & z_{ik} & \in & P(\mathbb{S}) & \quad\text{for}\quad a_{ik} \notin \{0_W, 1_W\} \end{array} \tag{5.16}$$

5.3.3.3 Operations

The operations $\sqcup$ and $\circ$ are defined for the weight set $Z = P(\mathbb{S})$. Let the path sets a_{ik}, b_{ik} be weighted with sets x_{ik}, y_{ik} of simple words. The weight $x_{ik} \sqcup y_{ik}$ of the union $a_{ik} \sqcup b_{ik}$ is the union $x_{ik} \cup y_{ik}$ or the two sets of simple words. The weight $x_{ik} \circ y_{km}$ of the concatenation $a_{ik} \circ b_{km}$ is the set of all simple words formed by concatenating a simple word from x_{ik} with a simple word from y_{km}.

$$\begin{array}{lccl} \text{union} & x_{ik} \sqcup y_{ik} & := & x_{ik} \cup y_{ik} \\ \text{concatenation} & x_{ik} \circ y_{km} & := & \{x \circ y \in \mathbb{S} \mid x \in x_{ik} \wedge y \in y_{km}\} \end{array} \tag{5.17}$$

The domain $(P(\mathbb{S}); \sqcup; \circ)$ is a literal path algebra with the zero element $0_Z = \{\}$ and the unit element $1_Z = \{\lambda\}$. The operations have the required properties.

5.3.3.4 Weight matrices

Let a directed graph be given. If the graph contains an edge from vertex i to vertex $k \neq i$, then the element z_{ik} of the elementary weight matrix $\mathbf{Z}$ is a one-element set containing the simple word with the characters of the vertices i and k. Otherwise, z_{ik} is the zero element. The matrix $\mathbf{Z}$ is stable. If the graph contains paths from vertex i to vertex $k \neq i$, then the element z^*_{ik} of the closure $\mathbf{Z}^*$ is equal to the set of words for the elementary paths from i to k. Otherwise, z^*_{ik} is the zero element. If the graph contains cycles through the vertex k, then the element z^*_{kk} of the closure $\mathbf{Z}^*$ contains the empty word λ as well as all words for the elementary cycles through k. Otherwise z^*_{kk} is the unit element.

5.3.4 Extreme elementary paths

The path algebra for the shortest or longest elementary paths is defined on the basis of the path algebra for elementary paths.

5.3.4.1 Problem

Let the vertices of a directed graph be uniquely labeled by the characters of an alphabet $\mathbb{A}$. A simple path from vertex i to vertex k does not contain any vertex more than once. It is called a shortest path from vertex i to vertex k if it does not contain more vertices than any other path from vertex i to vertex k. It is called a longest path from vertex i to vertex k if it does not contain fewer vertices than any other path from vertex i to vertex k. The words for all shortest or all longest paths from vertex i to vertex k are to be determined.

5.3.4.2 Weights

Let the path set a_{ik} containing paths from vertex i to vertex k be given. The set of all extreme simple words for the shortest or longest paths contained in a_{ik} is chosen as the weight z_{ik} of the path set a_{ik}. The weight mapping has the same form as for simple paths:

$$\begin{array}{lclcll} f(0_W) & = & 0_Z & = & \{\} & \\ f(1_Z) & = & 1_Z & = & \{\lambda\} & \\ f(z_{ik}) & = & z_{ik} & \in & P(\mathbb{S}) & \text{for} \quad a_{ik} \notin \{0_W, 1_W\} \end{array} \tag{5.18}$$

5.3.4.3 Operations

The operations $\sqcup$ and $\circ$ are defined for the weight set $P(\mathbb{S})$. Let the path sets a_{ik}, b_{ik} be weighted by the sets x_{ik}, y_{ik} of extreme simple words. The weight $x_{ik} \sqcup y_{ik}$ of the union $a_{ik} \sqcup b_{ik}$ is the reduction $\operatorname{extr}(x_{ik} \cup y_{ik})$ of the union $x_{ik} \cup y_{ik}$ to the set of extreme simple words. The concatenation $\circ$ is defined as for simple paths:

$$\begin{array}{lll} \text{union} & x_{ik} \sqcup y_{ik} := & \operatorname{extr}(x_{ik} \cup y_{ik}) \\ \text{concatenation} & x_{ik} \circ y_{km} := & \{x \circ y \in \mathbb{S} \mid x \in x_{ik} \wedge y \in y_{km}\} \end{array} \tag{5.19}$$

As in the case of simple paths, the domain $(P(\mathbb{S}); \sqcup, \circ)$ is a literal path algebra with the zero element $0_Z = \{\}$ and the unit element $1_Z = \{\lambda\}$.

5.3.4.4 Weight matrices

The elementary weight matrices $\mathbf{Z}$ of the literal path algebra for simple paths and for extreme simple paths coincide. The matrix $\mathbf{Z}$ is stable both for shortest and for longest simple paths. If the graph contains paths from vertex i to vertex k, then the element z^*_{ik} of the closure $\mathbf{Z}^*$ is equal to the set of words for the extreme simple paths from i to k. Otherwise z^*_{ik} is the zero element. If the graph contains cycles through the vertex k, then the element z^*_{kk} of the closure $\mathbf{Z}^*$ contains all words for the extreme simple cycles through k. Otherwise z^*_{kk} is the unit element. Since the shortest cycle through a vertex k is always the empty path λ, z^*_{kk} is always the unit element in the case of shortest simple paths.

5.3.5 Properties of elementary path algebra

5.3.5.1 Powers of an element

The 0^{th} power x^0 of an element $x \in Z$ of the weight set Z is defined to be the unit element 1_Z. The m^{th} power x^m is defined as the concatenation of in Section , x^{m-1} and x.

$$x^0 := 1_Z \qquad x^m := x^{m-1} \circ x \tag{5.20}$$

An element x is said to be idempotent if $x^2 = x$. Every power x^m of an idempotent element is equal to x for $m \geq 1$. An element x is said to be nilpotent of degree q if $x^q = 0_Z$. Every power x^m of a nilpotent element is equal to 0_Z for $m \geq q$. Every power x^m of a nilpotent element x is equal to 0_Z for $m \geq q$.

$$\begin{array}{ll} \text{idempotent} & x^2 = x \\ \text{nilpotent} & x^q = 0_Z \end{array} \tag{5.21}$$

5.3.5.2 Closure of an element

The reflexive transitive closure $\hat{x}$ of an element $x \in Z$ of a weight set Z is calculated as the union of the powers of x. If the union does not change beyond a certain power x^p, then the element x is stable and the closure $\hat{x}$ exists. The positive integer p is called the stability index of the element.

$$\hat{x} = 1_Z \sqcup x \sqcup x^2 \sqcup \ldots = \bigsqcup_{m \geq 0} x^m = \bigsqcup_{m=0}^{p} x^m \tag{5.22}$$

If an element is nilpotent, idempotent or subunitary, then it is stable.

$$\begin{array}{lcll} \text{nilpotent} & \hat{x} & = & 1_Z \sqcup x \sqcup x^2 \sqcup \ldots \sqcup x^{q-1} \\ \text{idempotent} & \hat{x} & = & 1_Z \sqcup x \\ \text{subunitary} & \hat{x} & = & 1_Z \end{array} \tag{5.23}$$

5.3.5.3 Stability

Path algebras are classified with respect to stability. A path algebra is said to be conditionally stable if at least one element of the weight set is not stable. It is said to be unconditionally stable if every element of the weight set is stable. It is said to be unitarily stable if the reflexive transitive closure of every element of the weight set is the unit element. The elementary path algebra is unconditionally stable with a closure $\hat{x} = 1_Z$.

5.3.6 Elementary paths example

The graph shown in Figure 5.1 will be used as an example to show all the elementary paths in a graph to a given vertex.

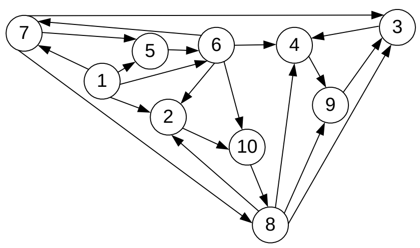


Figure 5.1: Elementary paths example graph

After the graph has been decomposed into its strongly connected components, reduced to an acyclic directed graph and sorted topologically, the sequence of tasks shown in Figure 5.2 is found. All the possible elementary paths to vertex 4 is listed in Appendix A. It is also shown on the sequence of tasks in Figure 5.2. However, since some paths are extensions of others, each individiual path is not that distinct on the sequence of tasks. Different filters can be used to display only a subset of the total set of calculated paths.

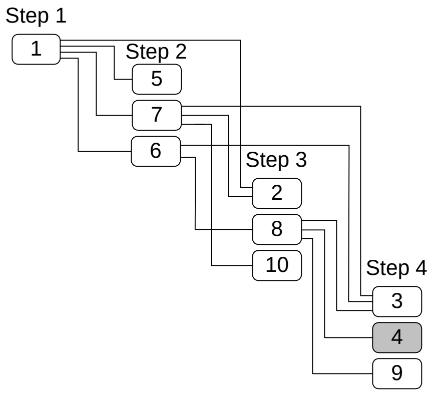


Figure 5.2: Elementary paths example sequence of tasks

5.4 Logical critical path

A logical critical path is a longest elementary path, from a source vertex to a sink vertex (see Section 4.5.1), through a directed acyclic graph. Such paths correspond to the longest time to perform an ordered sequence of tasks. Tasks that lie along the logical critical path cannot be delayed without delaying the finish time for the entire project. Therefore, it is critical for these tasks to be identified in order to avoid costly delays to the whole project. A project can have several, parallel logical critical paths. Similarly, to accelerate a project, it is necessary to reduce the total time required for the tasks on the logical critical path.

5.5 Systems of equations

5.5.1 Solution of systems of equations

5.5.1.1 Introduction

Let a directed graph with n vertices be given. The edge weights of the graph are arranged in the elementary weight matrix $\mathbf{A}$. A path algebra for a path problem in this graph leads to a system of n equations with the solution vector $\mathbf{x}$ depending on the vector $\mathbf{b}$ on the right-hand side. The system of n equations with n variables is formulated as follows:

$$\begin{aligned} \mathbf{x} &= \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b} \\ x_i &= a_{i1} \circ x_1 \sqcup a_{i2} \circ x_2 \sqcup ... \sqcup a_{in} \circ x_n \sqcup b_i \qquad i = 1, ..., n \end{aligned} \tag{5.24}$$

5.5.1.2 Solutions

Let the matrix $\mathbf{A}$ of a system of equations $\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b}$ be stable. Then the system of equations has a solution $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ in which $\mathbf{A}^*$ is the closure of $\mathbf{A}$. If the matrix $\mathbf{A}$ is nilpotent, then the solution $\mathbf{x}$ is unique. If the matrix $\mathbf{A}$ is not nilpotent, several solutions may exist. If several solutions exist, then $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ is the least solution.

5.5.1.3 Staggered system of equations

A system of equations $\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b}$ is said to be staggered if the matrix $\mathbf{A}$ is a lower triangular matrix with zero elements on and above the diagonal or an upper triangular matrix with zero elements on and below the diagonal. The solution $\mathbf{x}$ of a staggered system of equations is unique.

5.5.1.4 Equivalent systems of equations

Two systems of equations $\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b}$ and $\mathbf{x} = \mathbf{C} \circ \mathbf{x} \sqcup \mathbf{d}$ with stable matrices $\mathbf{A}$ and $\mathbf{C}$ are said to be equivalent if their least solutions $\mathbf{A}^* \circ \mathbf{b}$ and $\mathbf{C}^* \circ \mathbf{d}$ are identical.

5.5.2 Direct methods of solution

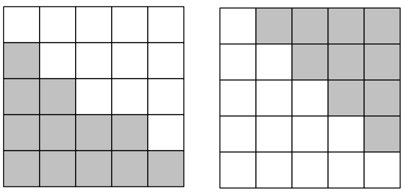
5.5.2.1 Introduction

The least solution of a system of equations may be determined directly if the system of equations is staggered. The solution of the staggered system of equations is determined by forward or back substitution. If the system of equations is not staggered, it is transformed into an equivalent staggered system of equations by elimination. The best-known elimination method is the one due to Gauss.

5.5.2.2 Forward substitution

Let the matrix $\mathbf{A}$ of the system of equations be a staggered lower triangular matrix: it contains only zero elements on and above the diagonal, as seen in Figure 5.3. The system of equations is solved by forward substiution. The variables are calculated as follows:

$$x_1 = b_1 \qquad x_k = \bigsqcup_{j=1}^{k-1} a_{kj} \circ x_j \sqcup b_k \qquad k = 2, ..., n \tag{5.25}$$



(a) lower (b) upper

Figure 5.3: Triangular matrices

5.5.2.3 Back substitution

Let the matrix $\mathbf{A}$ of the system of equations be a staggered upper triangular matrix: it contains only zero elements on and below the diagonal, as seen in Figure 5.3. The system of equations is solved by back substitution. The variables are calculated as follows:

$$x_n = b_n \qquad x_k = \bigsqcup_{j=k+1}^{n} a_{kj} \circ x_j \sqcup b_k \qquad k = n-1, ..., 1 \tag{5.26}$$

5.5.2.4 Elimination

In order to eliminate a variable x_k from the system $\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b}$, the k^{th} equation is first solved for x_k. Then x_k is eliminated in the other equations by substitution. To solve the k^{th} equation x_k, the terms which do not involve x_k are combined into a value c_k.

$$\begin{aligned} x_k &= \bigsqcup_j a_{kj} \circ x_j \sqcup b_k \\ x_k &= a_{kk} \circ x_k \sqcup c_k \qquad \text{with } c_k = \bigsqcup_{j \neq k} a_{kj} \circ x_j \sqcup b_k \end{aligned} \tag{5.27}$$

The equation $x_k = a_{kk} \circ x_k \sqcup c_k$ has a least solution if the element a_{kk} is stable, so that the closure $\hat{a}_{kk}$ exists. The least solution is:

$$\begin{aligned} x_k &= \hat{a}_{kk} \circ c_k \\ x_k &= \bigsqcup_{j \neq k} \hat{a}_{kk} \circ a_{kj} \circ x_j \sqcup \hat{a}_{kk} \circ b_k \end{aligned} \tag{5.28}$$

To eliminate the variable x_k in the i^{th} equation, the terms which do not involve x_k are combined into a value c_i:

$$\begin{aligned} x_i &= \bigsqcup_j a_{ik} \circ x_j \sqcup b_i \\ x_i &= a_{ik} \circ x_k \sqcup c_i \qquad \text{with } c_i = \bigsqcup_{j \neq k} a_{ik} \circ x_j \sqcup b_i \end{aligned} \tag{5.29}$$

The solution for x_k is substituted into the i^{th} equation $x_i = a_{ik} \circ x_k \sqcup c_i$. This substitution eliminates x_k in the i-th equation:

$$\begin{aligned} x_i &= a_{ik} \circ \hat{a}_{kk} \circ c_k \sqcup c_i \\ x_i &= \bigsqcup_{j \neq k} (a_{ij} \sqcup a_{ik} \circ \hat{a}_{kk} \circ a_{kj}) \circ x_j \sqcup (b_i \sqcup a_{ik} \circ \hat{a}_{kk} \circ b_k) \end{aligned} \tag{5.30}$$

In performing the elimination, it is assumed that the element a_{kk} is stable, so that the closure $\hat{a}_{kk}$ exists. If this is not the case, the elimination cannot be performed. The closure $\hat{a}_{kk}$ of the element a_{kk} is calculated as a union of powers of a_{kk} according to Section 5.3.5.2. For various path algebras, the closure $\hat{a}_{kk}$is known a priori and need not be calculated explicitly.

5.5.2.5 Gaussian elimination method

Let a system $\mathbf{x} = \mathbf{A_0} \circ \mathbf{x} \sqcup \mathbf{b_0}$ with n variables be given. It is transformed into a staggered system of equations with an upper triangular matrix in n consecutive steps.

$$\mathbf{x} = \mathbf{A_k} \circ \mathbf{x} \sqcup \mathbf{b_k} \qquad k = 1, ...n \tag{5.31}$$

In every step $k = 1, ..., n$, the variable x_k is eliminated in the equations $i = k, ..., n$ of the system $\mathbf{x} = \mathbf{A_{k-1}} \circ \mathbf{x} \sqcup \mathbf{b_{k-1}}$. The formulas for the elements of the matrix $\mathbf{A}_k$ and the vector $\mathbf{b}_k$ are compiled below.

$$\begin{array}{lclclrcl} \tilde{a}_{kj} &=& \hat{a}_{kk} \circ a_{kj} & & & j &=& k+1, ..., n \\ \tilde{a}_{ij} &=& a_{ij} \sqcup a_{ik} \circ \hat{a}_{kk} \circ a_{kj} &=& a_{ij} \sqcup a_{ik} \circ \tilde{a}_{kj} & i, j &=& k+1, ..., n \\ \tilde{b}_k &=& \hat{a}_{kk} \circ b_k & & & & & \\ \tilde{b}_i &=& b_i \sqcup a_{ik} \circ \hat{a}_{kk} \circ b_k &=& b_i \sqcup a_{ik} \circ \tilde{b}_k & i &=& k+1, ..., n \end{array} \tag{5.32}$$

The matrices $\mathbf{A}_k$ and the vectors $\mathbf{b}_k$ in the steps $k = 1, ..., n$ are not explicitly constructed in the algorithms. Instead, the matrix and the vector of the original system of equations are repeatedly overwritten. In the k^{th} step, the elements are overwritten as follows:

$$\begin{array}{rclrcl} a_{kj} & \leftarrow & \hat{a}_{kk} \circ a_{kj} & j & = & k+1, ..., n \\ a_{ij} & \leftarrow & a_{ij} \sqcup a_{ik} \circ a_{kj} & i, j & = & k+1, ..., n \\ b_k & \leftarrow & \hat{a}_{kk} \circ b_k & & & \\ b_i & \leftarrow & b_i \sqcup a_{ik} \circ b_k & i & = & k+1, ..., n \end{array} \tag{5.33}$$

The Gaussian elimination method assumes that in each step the diagonal element a_{kk} is stable, so that the closure $\hat{a}_{kk}$ exists. If this is not the case, the elimination proccess fails. Upon successful completion of the elimination process, the system of equations is staggered, and the variables may be determined by back substitution. The solution reached by Gaussian elimination is always the least solution.

5.5.3 Iterative Methods of Solution

5.5.3.1 Introduction

Various iterative methods have been developed for solving systems of equations. Such methods form the basis for powerful algorithms in graph theory. In formulating these methods, it is assumed that the matrix of the system of equations contains zero elements on the diagonal. The simplest iterative methods are the Jacobi method, the Gauss-Seidel method and the forward and back substitution method. They form a class of methods and are treated in the following in generalized form.

5.5.3.2 General iteration

The general iteration for solving a system of equations $\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b}$ consists of the following steps:

$$\begin{array}{llcll} \text{initial values} & \mathbf{x}_0 & = & \mathbf{b} & \\ \text{iteration} & \mathbf{x}_{k+1} & = & \mathbf{M} \circ \mathbf{x}_k \sqcup \mathbf{N} \circ \mathbf{b} & k = 0, 1, ... \\ \text{termination} & \mathbf{x}_{k+1} & = & \mathbf{x}_k & \end{array} \tag{5.34}$$

The vector $\mathbf{b}$ is conveniently chosen as the initial vector $\mathbf{x}_0$ for the iteration, since every solution $\mathbf{x}$ of the system of equations $\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b}$ contains the vector $\mathbf{b}$. In each iteration $k = 0, 1, ...$ an iterated vector $\mathbf{x}_{k+1}$ is calculated from the vector $\mathbf{x}_k$ and the vector $\mathbf{b}$ using the matrices $\mathbf{M}$ and $\mathbf{N}$. The iteration is terminated if two consecutive iterated vectors $\mathbf{x}_{k+1}$ and $\mathbf{x}_k$ coincide. The matrices $\mathbf{M}$ and $\mathbf{N}$ of the iteration procedure must be chosen such that the iteration yields the least solution $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ of the system of equations. The relevant conditions are derived in the following section.

5.5.3.3 Conditions

The iteration with the general rule defined above yields iterated vectors of the following form:

$$\begin{array}{lclcl} \mathbf{x}_0 & = & \mathbf{b} & & \\ \mathbf{x}_1 & = & \mathbf{M} \circ \mathbf{x}_0 \sqcup \mathbf{N} \circ \mathbf{b} & = & \mathbf{M} \circ \mathbf{b} \sqcup \mathbf{N} \circ \mathbf{b} \\ \mathbf{x}_2 & = & \mathbf{M} \circ \mathbf{x}_1 \sqcup \mathbf{N} \circ \mathbf{b} & = & \mathbf{M}^2 \circ \mathbf{b} \sqcup (\mathbf{I} \sqcup \mathbf{M}) \circ \mathbf{N} \circ \mathbf{b} \\ \mathbf{x}_{k+1} & = & \mathbf{M} \circ \mathbf{x}_k \sqcup \mathbf{N} \circ \mathbf{b} & = & \mathbf{M}^{k+1} \circ \mathbf{b} \sqcup \left(\mathbf{I} \sqcup \mathbf{M} \sqcup \mathbf{M}^2 \sqcup ... \sqcup \mathbf{M}^k\right) \circ \mathbf{N} \circ \mathbf{b} \end{array} \tag{5.35}$$

The iteration can only yield a solution if the matrix $\mathbf{M}$ is stable. If the stability index of the matrix $\mathbf{M}$ is p, the vector $\mathbf{x}_{p+1}$ is obtained as:

$$\mathbf{x}_{p+1} = \mathbf{M}^{p+1}\mathbf{b} \sqcup \mathbf{M}^* \circ \mathbf{N} \circ \mathbf{b} \tag{5.36}$$

The vector $\mathbf{x}_{p+1}$ contains the least solution $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ of the system of equations if the product $\mathbf{M}^* \circ \mathbf{N}$ is equal to the closure $\mathbf{A}^*$.

$$\mathbf{x}_{p+1} = \mathbf{M}^{p+1} \circ \mathbf{b} \sqcup \mathbf{A}^* \circ \mathbf{b} \qquad \text{with} \qquad \mathbf{M}^* \circ \mathbf{N} = \mathbf{A}^* \tag{5.37}$$

The vector $\mathbf{x}_{p+1}$ is the least solution $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ of the system of equations only if $\mathbf{M}^{p+1} \circ \mathbf{b} \sqsubseteq \mathbf{M}^{p+1} \circ \mathbf{A}^* \circ \mathbf{b} = \mathbf{M}^{p+1} \circ \mathbf{M}^* \circ \mathbf{N} \circ \mathbf{b} \sqsubseteq \mathbf{M}^* \circ \mathbf{N} \circ \mathbf{b} = \mathbf{A}^* \circ \mathbf{b}$

Hence the general iteration procedure yields the least solution $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ of the system of equations if the matrix $\mathbf{M}$ is stable and the product $\mathbf{M}^* \circ \mathbf{N}$ is identical with the closure $\mathbf{A}^*$. If the stability index of the matrix $\mathbf{M}$ is p, then $p + 1$ iterations are required to determine the least solution.

5.5.3.4 Jacobi method

The Jacobi method is the simplest method for solving a system of equations. The iteration is carried out according to the following rule:

$$\text{iteration} \quad \mathbf{x}_{k+1} \quad = \quad \mathbf{A} \circ \mathbf{x}_k \sqcup \mathbf{b} \tag{5.38}$$

The iteration procedure is a special case of the general iteration procedure and satisfies the conditions for the least solution of the system of equations.

$$\begin{array}{llll} \text{matrices} & \mathbf{M} = \mathbf{A} & & \mathbf{N} = \mathbf{I} \\ \text{condition} & \mathbf{M}^* \circ \mathbf{N} = & \mathbf{A}^* \circ \mathbf{I} & = \mathbf{A}^* \end{array} \tag{5.39}$$

5.5.3.5 Gauss-Seidel method

In the Gauss-Seidel method, the matrix $\mathbf{A}$ of the system of equations is represented as the union of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{R}$. The lower triangular matrix $\mathbf{L}$ contains zero elements on and above the diagonal. The upper triangular matrix $\mathbf{R}$ contains zero elements on and below the diagonal. The system of equations to be solved may thus be formulated as follows:

$$\mathbf{x} = \mathbf{A} \circ \mathbf{x} \sqcup \mathbf{b} \quad \Leftrightarrow \quad \mathbf{x} = (\mathbf{L} \sqcup \mathbf{R}) \circ \mathbf{x} \sqcup \mathbf{b} \quad \Leftrightarrow \quad \mathbf{x} = \mathbf{L} \circ \mathbf{x} \sqcup \mathbf{R} \circ \mathbf{x} \sqcup \mathbf{b} \tag{5.40}$$

The Gauss-Seidel iteration is carried out according to the following rule:

$$\text{iteration} \qquad \mathbf{x}_{k+1} = \mathbf{L} \circ x_{k+1} \sqcup \mathbf{R} \circ \mathbf{x}_k \sqcup \mathbf{b} \tag{5.41}$$

This iteration procedure corresponds to a staggered system of equations with the matrix $\mathbf{L}$ and the solution vector $\mathbf{x}_{k+1}$. To reduce it to the general iteration procedure, the solution vector $\mathbf{x}_{k+1}$ is written as a function of $\mathbf{x}_k$ and $\mathbf{b}$ using the closure $\mathbf{L}^*$. With the rules in Section 5.5.3.2 the iteration procedure is shown to satisfy the conditions for the least solution of the system of equations:

$$\begin{array}{ll} \text{iteration} & \mathbf{x}_{k+1} = \mathbf{L}^* \circ (\mathbf{R} \circ \mathbf{x}_k \sqcup \mathbf{b}) = \mathbf{L}^* \circ \mathbf{R} \circ \mathbf{x}_k \sqcup \mathbf{L}^* \circ \mathbf{b} \\ \text{matrices} & \mathbf{M} = \mathbf{L}^* \circ \mathbf{R} \quad \mathbf{N} = \mathbf{L}^* \\ \text{condition} & \mathbf{M}^* \circ \mathbf{N} = (\mathbf{L}^* \circ \mathbf{R})^* \circ \mathbf{L}^* = (\mathbf{L} \sqcup \mathbf{R})^* = \mathbf{A}^* \end{array} \tag{5.42}$$

5.5.3.6 Forward and back substitution method

Like the Gauss-Seidel Method, this method uses a decomposition of the matrix $\mathbf{A}$ of the system of equations into a union of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{R}$. The iteration is carried out according to the following rules:

$$\begin{array}{lrcl} \text{iteration} & \mathbf{y}_{k+1} & = & \mathbf{R} \circ \mathbf{y}_{k+1} \sqcup \mathbf{x}_k \sqcup \mathbf{b} \\ & \mathbf{x}_{k+1} & = & \mathbf{L} \circ \mathbf{x}_{k+1} \sqcup \mathbf{y}_{k+1} \end{array} \tag{5.43}$$

The first equation corresponds to a system of equations with the matrix $\mathbf{R}$ and the solution vector $\mathbf{y}_{k+1}$, which is solved by back substitution. The second equation corresponds to a system of equations with the matrix $\mathbf{L}$ and the solution vector $\mathbf{x}_{k+1}$, which is solved by forward substitution. In order to reduce the iteration procedure to the general iteration procedure, the solution vectors $\mathbf{y}_{k+1}$ and $\mathbf{x}_{k+1}$ are specified using the closures $\mathbf{R}^*$ and $\mathbf{L}^*$, and the first equation is substituted into the second equation. By the rules for closures the iteration procedure satisfies the required condition.

$$\begin{array}{lrcllcl} \text{iteration} & \mathbf{y}_{k+1} & = & \mathbf{R}^* \circ (\mathbf{x}_k \sqcup \mathbf{b}) & & & \\ & \mathbf{x}_{k+1} & = & \mathbf{L}^* \circ \mathbf{y}_{k+1} & & & \\ & \mathbf{x}_{k+1} & = & \mathbf{L}^* \circ \mathbf{R}^* \circ (\mathbf{x}_k \sqcup \mathbf{b}) & = & \mathbf{L}^* \circ \mathbf{R}^* \circ \mathbf{x}_k \sqcup \mathbf{L}^* \circ \mathbf{R}^* \circ \mathbf{b} \\ \text{matrices} & \mathbf{M} & = & \mathbf{N} = \mathbf{L}^* \circ \mathbf{R}^* & & & \\ \text{condition} & \mathbf{M}^* \circ \mathbf{N} & = & (\mathbf{L}^* \circ \mathbf{R}^*)^* \circ (\mathbf{L}^* \circ \mathbf{R}^*) & = & (\mathbf{L}^* \circ \mathbf{R}^*)^* = (\mathbf{L} \sqcup \mathbf{R})^* = \mathbf{A}^* \end{array} \tag{5.44}$$

5.5.3.7 Number of iterations

Every iterative method yields the least solution $\mathbf{x} = \mathbf{A}^* \circ \mathbf{b}$ of the system of equations after at most $p+1$ iterations, where p is the stability index of the matrix $\mathbf{M}$. An upper bound for the stability index p of $\mathbf{M}$ is given by the stability index q of the matrix $\mathbf{A}$. The quadratic matrix $\mathbf{A}$ with n rows and columns has a stability index $q < n$ if the path algebra is stable. In this case, the iterative methods require at most n iterations.

A stronger upper bound may be derived for the matrix $\mathbf{M}$ of the forward and back substitution method assuming a unitarily stable path algebra. The derivation leads to a stability index $p \leq q/2 + 1$. This method thus requires roughly half as many iterations as the Jacobi method and the Gauss-Seidel method do in the worst case. Since the calculational cost per iteration is the same for all iterative methods, the calculational cost of this method is roughly half that of the Jacobi and Gauss-Seidel methods in the worst case.

Knowledge of an upper bound on the number of iterations in the case of stable matrices is of fundamental importance for algorithms. If the upper bound is exceeded in the course of the iteration process, then the matrix $\mathbf{A}$ of the system of linear equations is not stable, and the iteration is aborted without a result.

5.6 Relabelling of vertices

After a graph has been decomposed into its strongly connected components we can construct the reduced graph of super vertices and super edges. The reduced graph can be sorted topologically, since it is a directed acyclic graph. The topological sorting can be used to relabel the vertices of a graph, such that each edge points from a lower numbered vertex to a higher numbered vertex. Such relabelling results in an upper triangular adjacency-matrix for a directed acyclic graph.

The vertices of cyclic graphs can also be relabelled, considering the strongly connected components in topological order. The vertices in a strongly connected component are relabelled as long as none of their successors have been relabelled. In this case, the vertex is stored and processed after the other vertices. The adjacency-matrix up to this point is now an upper triangular matrix. All the unprocessed vertices are labelled at the end. The adjacency-matrix from this point is not an upper triangular matrix. Therefore, the adjacency-matrix will be of the form shown in Figure 5.4.

Knowledge of the upper triangular part of the adjacency-matrix can be used to reduce the number of calculations for the methods of solutions of path algebras (see Section 5.5).

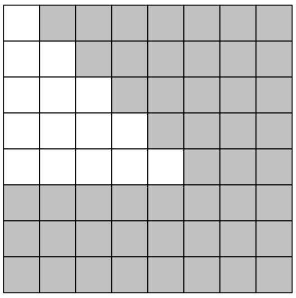


Figure 5.4: Adjacency-matrix after relabelling

5.6.1 Relabelling example

The graph shown in Figure 5.5 will be used to demonstrate the advantages of relabelling the vertices of the graph.

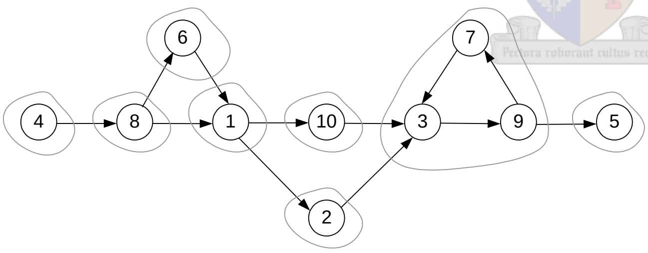


Figure 5.5: Graph before relabelling

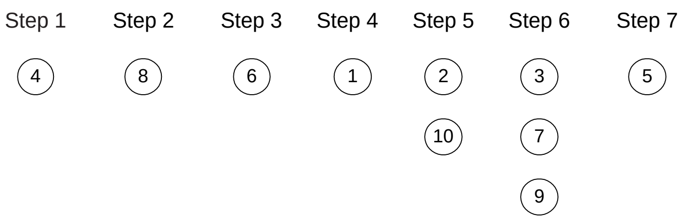


Figure 5.6: Topologically sorted graph

The strongly connected components are shown in Figure 5.5 and the topologically sorted graph in Figure 5.6. The relabelled graph is shown in Figure 5.7

Weight matrix before relabelling:

$$\mathbf{Z} \;=\; \begin{matrix}
\{\} & \{(1,2)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(1,10)\} \\
\{\} & \{\} & \{(2,3)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(3,9)\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(4,8)\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{(6,1)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{(7,3)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{(8,1)\} & \{\} & \{\} & \{\} & \{\} & \{(8,6)\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{(9,5)\} & \{\} & \{(9,7)\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{(10,3)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\}
\end{matrix}$$

Weight matrix after relabelling:

$$\mathbf{Z} \;=\; \begin{matrix}
\{\} & \{(1,0)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{(2,3)\} & \{(2,4)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{(3,4)\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{(4,5)\} & \{(4,6)\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(5,10)\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(6,10)\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(7,8)\} & \{(7,9)\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(8,10)\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\
\{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{(10,7)\} & \{\} & \{\} & \{\}
\end{matrix}$$

The partially staggered form can be seen in the weight matrix after relabelling.

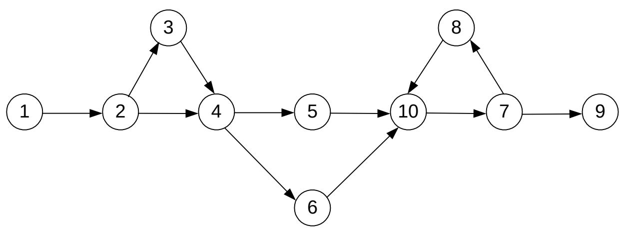


Figure 5.7: Graph after relabelling

Chapter 6

Implementation of computer model for graphs and performance testing

6.1 Introduction

It has been shown that the "has to be executed before" relation in the set of tasks of an engineering process model can be described by a directed graph. Different algorithms were devised to determine certain structural characteristics of these graphs. The Java programming language was used to implement these concepts.

First, we need a computer model for the graphs. Different possible models, including the model that was implemented are discussed in Section 6.2. An algorithm to generate random test graphs is discussed in Section 6.2.2. The Unified Modelling Language (UML) (see Section 6.3) will be used to outline the implementation of the computer model. See reference [3] for a more detailed description of the unified modelling language.

Different methods of solution of a system of equations were implemented. Random test graphs were used to compare the performance of these methods (see Section 6.5) in order to choose the best method. See the attached CD for the source code of the implementation, as well as a complete documented example.

6.2 Computer models for graphs

6.2.1 Data structures

6.2.1.1 Adjacency-matrix representation

The amount of space used to store a graph using an adjacency-matrix is proportional to the square of the number of vertices (V^2). If the number of edges is relatively small compared to the number of vertices, the adjacency-matrix uses an unneccessary amount of storage space, since most of the entries are false.

6.2.1.2 Adjacency-lists representation

In the adjacency-lists representation, a list of successors is associated with each vertex. The amount of space used to store a graph using adjacency-lists is proportional to the number of vertices and the number of edges ($V + E$), since only existing edges are stored. This is the primary advantage of the adjacency-lists representation over the adjacency-matrix representation.

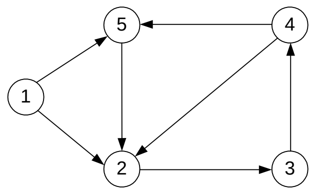
6.2.1.3 Alternative representation

The graph representation used in this implementation is similar to the adjacency-lists representation. A set of vertices and a set of edges are stored as part of the graph. A set of successors and a set of predecessors is associated with each vertex. The set of predecessors can be used to quickly find the reverse of a graph. When an edge (x, y) is added to a graph, the vertex x is added to the set of successors of vertex y and vertex y is added to the set of successors of x.

The amount of space used to store a graph using this representation is proportional to the number of vertices and five times the number of edges ($V + 5E$). Each vertex is stored once in the vertex set (V). Each edge is stored once in the edge set (E). For each edge two vertices, the start vertex and end vertex, are stored ($2E$), as well as two vertices, one predecessor and one successor ($2E$). Although this representation is more expensive in storage space than the adjacency lists representation, it saves on the number of calculations for graph algorithms.

6.2.1.4 Example

The graph in Figure 6.1 will be used to illustrate the different graph representations.



(a) Graph

Adjacency matrix:

	1	2	3	4	5
1	0	1	0	0	1
2	0	0	1	0	0
3	0	0	0	1	0
4	0	1	0	0	1
5	0	1	0	0	0

Adjacency lists:

1	→	2,5
2	→	3
3	→	4
4	→	5
5	→	1

(b) Adjacency-matrix and adjacency-lists

Alternative representation:

Vertex set: {1,2,3,4,5}
Edge set: {(1,2),(1,5),(2,3),(3,4),(4,2),(4,5),(5,2)}

Vertex	Set of successors	Set of predecessors
1	{2,5}	{}
2	{3}	{1,4,5}
3	{4}	{2}
4	{2,5}	{3}
5	{2}	{1,4}

Edge	Start vertex	End vertex
(1,2)	1	2
(1,5)	1	5
(2,3)	2	3
(3,4)	3	4
(4,2)	4	2
(4,5)	4	5
(5,2)	5	2

(c) Alternative representation

Figure 6.1: Graph representation

6.2.2 Graph generator

Graphs modelling an engineering process are sequential in nature. Therefore, we need to generate graphs with this property. Graphs will be generated randomly with the following specifications and restraints:

Specifications:

- number of vertices of the graph (n)
- number of steps in the sequence of tasks (s)
- maximum number of vertices in a step (t)

Restraints:

- The number of vertices should be equal to or greater than the number of steps, since there should be at least one vertex in a step.
- The maximum number of vertices in a step times the number of steps should be equal to or greater than the number of vertices, since there should be enough placements for each vertex.

$$(s \leq n \leq s * t) \tag{6.1}$$

.

- Each vertex should have a vertex in the previous step pointing to it. This is necessary to justify the placement of the vertex in the step.

Other considerations:

- Random edges spanning over two steps are added to add more complexity.
- Edges creating random strongly connected components are added.

6.2.2.1 Algorithm

- Create n new vertices.
- Create s new steps.
- Assign each vertex to a step.
- Create necessary edges in three steps:
  1. edges pointing to vertices from the previous step,
  2. edges spanning over two steps and
  3. edges creating strongly connected components.

6.2.2.2 Example

Generate a graph with $n = 10$ vertices in $s = 3$ steps, with a maximum number of $t = 4$ vertices per step. The generated elements are shown in Figure 6.2.

Vertex	Step
1	2
2	1
3	1
4	1
5	2
6	2
7	2
8	3
9	3
10	3

(a) Vertices and steps

Step 1: 2, 3, 4

Step 2: 1, 5, 6, 7

Step 3: 8, 9, 10

(b) Sequence of tasks

Edge	Type
(7,10)	1
(7,9)	1
(6,8)	1
(4,6)	1
(2,1)	1
(3,5)	1
(4,7)	1
(3,8)	2
(2,4)	3
(4,3)	3
(3,2)	3

(c) Edges

Figure 6.2: Elements

The resulting graph is shown in Figure 6.3.

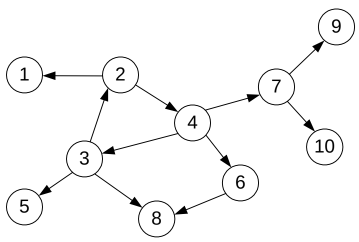


Figure 6.3: Random graph

6.3 Unified Modelling Language view of implementation

6.3.1 Introduction

The Unified Modeling Language (UML) is a graphical notation for drawing diagrams of software concepts. It can be used for drawing diagrams of a problem domain, a proposed software design, or an already completed software implementation. These three levels can be described as Conceptual, Specification and Implementation.

Implementation level diagrams have a strong connection to source code, since it is the intent of these diagrams to describe existing source code. As such there are rules and semantics that these diagrams must follow, in order to have very little ambiguity, and a great deal of formality.

Static UML diagrams, which describe the unchanging logical structure of software elements by depicting classes, objects, and data structures and the relationships that exist between them, will be used in this document. UML diagrams are not particularly good for communicating algorithmic detail.

6.3.2 Class diagrams

UML class diagrams allow us to denote the static contents of and relationships between classes. In a class diagram we can show the variables and methods of a class. We can also show whether one class holds a reference to another. In short, we can depict all the source code dependencies between classes.

This can be valuable. It can be much easier to evaluate the dependency structure of a system from a diagram than from source code. Diagrams make certain dependency structures visible.

A class diagram shows the major classes and relationships in the program. A class is represented by a rectangle, which can be subdivided into compartments. The top compartment is for the name of the class, the second is for the variables of the class and the third is for the methods of the class. The basic structure of a class diagram is shown in Figure 6.4.

Figure 6.5 shows the relationships between classes. A relationship is represented by an arrow. Associations between classes most often represent instance variables that hold references to other objects. The name on an association maps to the name of the variable that holds the reference. A number next to an arrowhead typically shows the number of instances held by the relationship. If some kind of a container, such as an array, is used, the symbol $*$ implies many.

An empty arrow head as in Figure 6.6 shows an inheritance relationship. In UML arrows heads point in the direction of source code dependency.

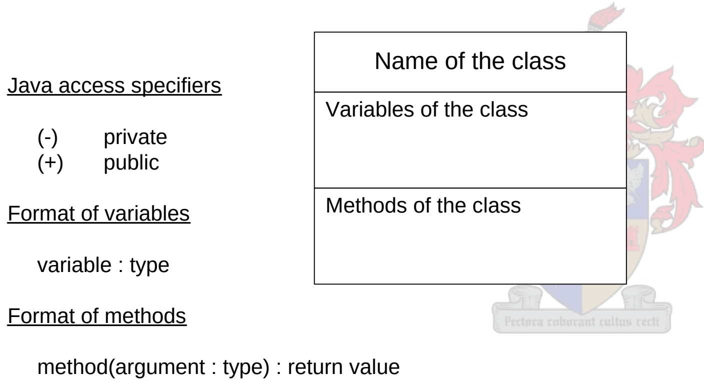


Figure 6.4: Class diagram basics

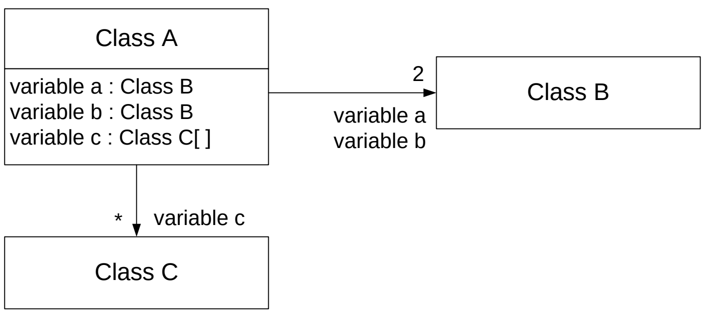
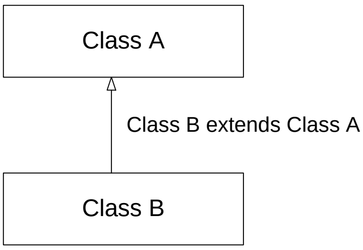


Figure 6.5: Class diagram associations



test

Figure 6.6: Class diagram inheritance

6.4 UML view of graph model

6.4.1 Basic graph implementation

An implementation of the basic graph model is shown in Figure 6.7. This model is extended as more functionality is added to the classes. These extensions are shown in detail in Appendix B.

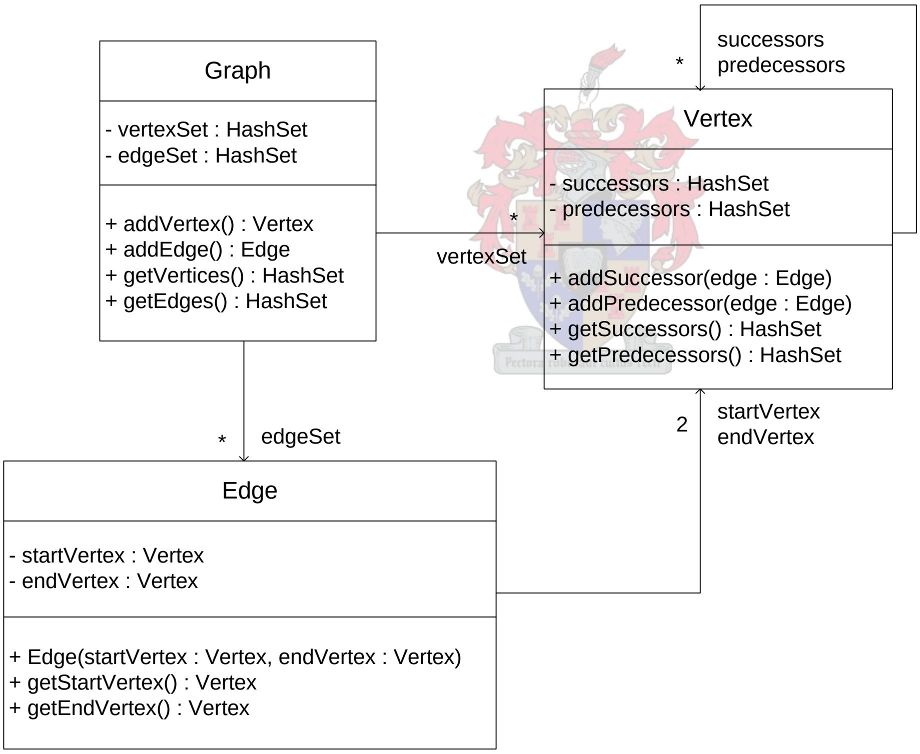


Figure 6.7: UML diagram of basic Graph implementation

6.5 Performance testing of solution methods

The performance of the solution methods of Jacobi and Gauss-Seidel, as well as the forward and back substitution method discussed in Section 5.5 is tested using random test data. Different methods of doing the same task have to be compared in order to choose the method best suited to the task.

During the solution of a system of equations a number of concatenation and union operations are performed. The number of calculations for each solution method is used to compare the performance of the different algorithms. The number of calculations is a function of the number of vertices in the graph. In the case of the iterative methods it is also a function of the number of iterations that has to be completed. Knowledge of the structure of the graph can be used to reduce the number of calculations per iteration for the iterative methods, using the method of relabelling discussed in Section 5.6. The execution times are also used as a comparison between the different methods.

6.5.1 Gauss elimination calculations

A graph with a vertex set of size = 5 will be used to demonstrate and derive equations to determine the number of calculations for each method.

6.5.1.1 Gauss elimination

The number of calculations performed during the solution of the equations in Section 5.5.2.5 can be determined as follows:

$k = 1$:

$$
\begin{array}{lllll}
a_{12} & \leftarrow & \hat{a}_{11} & \circ & a_{12} \\
a_{13} & \leftarrow & \hat{a}_{11} & \circ & a_{13} \\
a_{14} & \leftarrow & \hat{a}_{11} & \circ & a_{14} \\
a_{15} & \leftarrow & \hat{a}_{11} & \circ & a_{15}
\end{array}
\quad \text{4 calculations}
$$

$$
\begin{array}{lllllll}
a_{22} & \leftarrow & a_{22} & \sqcup & a_{21} & \circ & a_{12} \\
a_{23} & \leftarrow & a_{23} & \sqcup & a_{31} & \circ & a_{13} \\
a_{24} & \leftarrow & a_{24} & \sqcup & a_{41} & \circ & a_{14} \\
a_{25} & \leftarrow & a_{25} & \sqcup & a_{51} & \circ & a_{15} \\
a_{42} & \leftarrow & a_{42} & \sqcup & a_{21} & \circ & a_{12} \\
a_{43} & \leftarrow & a_{43} & \sqcup & a_{31} & \circ & a_{13} \\
a_{44} & \leftarrow & a_{44} & \sqcup & a_{41} & \circ & a_{14} \\
a_{45} & \leftarrow & a_{45} & \sqcup & a_{51} & \circ & a_{15}
\end{array}
\qquad
\begin{array}{lllllll}
a_{32} & \leftarrow & a_{32} & \sqcup & a_{21} & \circ & a_{12} \\
a_{33} & \leftarrow & a_{33} & \sqcup & a_{31} & \circ & a_{13} \\
a_{34} & \leftarrow & a_{34} & \sqcup & a_{41} & \circ & a_{14} \\
a_{35} & \leftarrow & a_{35} & \sqcup & a_{51} & \circ & a_{15} \\
a_{52} & \leftarrow & a_{52} & \sqcup & a_{21} & \circ & a_{12} \\
a_{53} & \leftarrow & a_{53} & \sqcup & a_{31} & \circ & a_{13} \\
a_{54} & \leftarrow & a_{54} & \sqcup & a_{41} & \circ & a_{14} \\
a_{55} & \leftarrow & a_{55} & \sqcup & a_{51} & \circ & a_{15}
\end{array}
\quad \text{32 calculations}
$$

$$
\begin{array}{lllllll}
b_1 & \leftarrow & \hat{a}_{11} & \circ & b_1 & & \\
b_2 & \leftarrow & b_2 & \sqcup & a_{21} & \circ & b_1 \\
b_3 & \leftarrow & b_3 & \sqcup & a_{31} & \circ & b_1 \\
b_4 & \leftarrow & b_4 & \sqcup & a_{41} & \circ & b_1 \\
b_5 & \leftarrow & b_5 & \sqcup & a_{51} & \circ & b_1
\end{array}
\quad 1 + 8 \text{ calculations}
$$

The total number of calculations for $k = 1$ is $4 + 32 + 1 + 8 = 45$ calculations. Similarly, for $k = 2$, $3 + 18 + 1 + 6 = 28$, $k = 3$, $2 + 8 + 1 + 4 = 15$, $k = 4$, $1 + 2 + 1 + 2 = 6$ and $k = 5$, 1. The total number of calculations for $k = 1, \ldots, 5$ is $45 + 28 + 15 + 6 + 1 = 95$ (excluding the number of calculations for determining the element closures).

6.5.1.2 Back substitution

The Gauss elimination is followed by a back substitution as discussed in Section 5.5.2.3.

$$
\begin{array}{lll}
x_5 & = & b_5 \\
x_4 & = & a_{45} \circ x_5 \sqcup b_4 \\
x_3 & = & a_{34} \circ x_4 \sqcup a_{35} \circ x_5 \sqcup b_3 \\
x_2 & = & a_{23} \circ x_3 \sqcup a_{24} \circ x_4 \sqcup a_{25} \circ x_5 \sqcup b_2 \\
x_1 & = & a_{12} \circ x_2 \sqcup a_{13} \circ x_3 \sqcup a_{14} \circ x_4 \sqcup a_{15} \circ x_5 \sqcup b_1
\end{array}
\quad \text{20 calculations}
$$

6.5.1.3 Generalized

The number of calculations for $k = 1, \ldots, 5$ can be rewritten as follows:

$$\begin{array}{lccccccc}
k=1 & : & 4 & + & 32 & + & 1 & + & 8 \\
k=2 & : & 3 & + & 18 & + & 1 & + & 6 \\
k=3 & : & 2 & + & 8 & + & 1 & + & 4 \\
k=4 & : & 1 & + & 2 & + & 1 & + & 2 \\
k=5 & : & 0 & + & 0 & + & 1 & + & 0
\end{array}$$

Considering each column separately, we begin to see a pattern forming:

$$\begin{array}{lccccccccccl}
\text{column 1:} & 0 & + & 1 & + & 2 & + & 3 & + & 4 & \\
\text{column 2:} & 0 & + & 2 & + & 8 & + & 18 & + & 32 & = 2\,(1+4+9+16) = 2\left(1^2+2^2+3^2+4^2\right) \\
\text{column 3:} & 1 & + & 1 & + & 1 & + & 1 & + & 1 & \\
\text{column 4:} & 0 & + & 2 & + & 4 & + & 6 & + & 8 & = 2\,(1+2+3+4)
\end{array}$$

The algebraic equations in Appendix C can be used to rewrite these sums in terms of the number of vertices n:

$$1+2+3+\ldots+(n-1) = \tfrac{(n-1)^2+(n-1)}{2} = \tfrac{n^2-2n+1+n-1}{2} = \tfrac{n^2-n}{2}$$

$$1^2+2^2+3^2+\ldots+(n-1)^2 = \tfrac{(n-1)(n-1+1)(2(n-1)+1)}{6} = \tfrac{(n-1)\times n\times(2n-1)}{6} = \tfrac{n(n-1)(2n-1)}{6}$$

$$\begin{aligned}
& 3\tfrac{(n^2-n)}{2} + 2\tfrac{n(n-1)(2n-1)}{6} + n \\
= \;& \left(\tfrac{3}{2}n^2 - \tfrac{3}{2}n\right) + \left(\tfrac{2}{3}n^3 - \tfrac{1}{3}n^2 - \tfrac{2}{3}n^2 + \tfrac{1}{3}n\right) + n \\
= \;& \tfrac{2}{3}n^3 + \tfrac{1}{2}n^2 - \tfrac{1}{6}n \\
= \;& n\left(\tfrac{2}{3}n^2 + \tfrac{1}{2}n - \tfrac{1}{6}\right)
\end{aligned}$$

The total number of calculations for Gauss elimination has now been determined as:

$n\left(\tfrac{2}{3}n^2 + \tfrac{1}{2}n - \tfrac{1}{6}\right) + \left(n^2 - n\right) = n\left(\tfrac{2}{3}n^2 + \tfrac{3}{2}n - \tfrac{7}{6}\right)$

and for back substitution:

$n^2 - n$

Element closures: We can determine the number of calculations in determining the element closures in terms of the stability index p of the element as follows:

$$\hat{a}_{kk} = 1_Z \sqcup a_{kk} \sqcup a_{kk} \circ a_{kk} \sqcup a_{kk} \circ a_{kk} \circ a_{kk} \sqcup \ldots$$

$1+2+3+\ldots+(p-1)$ concatenation ($\circ$) calculations and p union ($\sqcup$) calculations $= 1+2+3+\ldots+p = \frac{(p^2+p)}{2}$ calculations, $\frac{(p^2+p)}{2}$ calculations.

Total: Total number of calculations for the Gauss elimination method, with a back substitution:

$n\left(\tfrac{2}{3}n^2 + \tfrac{3}{2}n - \tfrac{7}{6}\right) + \frac{(p^2+p)}{2}$

where n is the number of vertices and p the stability index of the element.

6.5.2 Jacobi calculations

The number of calculations performed during the solution of the equations in Section 5.5.3.4 can be determined as follows:

$$\begin{aligned}
&\mathbf{y} = \mathbf{A} \circ \mathbf{x}_k \sqcup \mathbf{b} \text{ and } \mathbf{x}_{k+1} = \mathbf{y} \\
&y_i = \bigsqcup_{j=1}^{n} a_{ij} \circ x_j \sqcup b_i
\end{aligned} \tag{6.2}$$

$$\begin{array}{rcl} y_1 & = & a_{11} \circ x_1 \sqcup a_{12} \circ x_2 \sqcup a_{13} \circ x_3 \sqcup a_{14} \circ x_4 \sqcup a_{15} \circ x_5 \sqcup b_1 \\ y_2 & = & a_{21} \circ x_1 \sqcup a_{22} \circ x_2 \sqcup a_{23} \circ x_3 \sqcup a_{24} \circ x_4 \sqcup a_{25} \circ x_5 \sqcup b_2 \\ y_3 & = & a_{31} \circ x_1 \sqcup a_{32} \circ x_2 \sqcup a_{33} \circ x_3 \sqcup a_{34} \circ x_4 \sqcup a_{35} \circ x_5 \sqcup b_3 \quad \text{50 calculations} \\ y_4 & = & a_{41} \circ x_1 \sqcup a_{42} \circ x_2 \sqcup a_{43} \circ x_3 \sqcup a_{44} \circ x_4 \sqcup a_{45} \circ x_5 \sqcup b_4 \\ y_5 & = & a_{51} \circ x_1 \sqcup a_{52} \circ x_2 \sqcup a_{53} \circ x_3 \sqcup a_{54} \circ x_4 \sqcup a_{55} \circ x_5 \sqcup b_5 \end{array}$$

6.5.2.1 Generalized

Total number of $n \times 2n = 2n^2$ calculations, where n is the number of vertices.

6.5.3 Jacobi calculations after sorting

$$\begin{array}{rcl} y_1 & = & a_{12} \circ x_2 \sqcup a_{13} \circ x_3 \sqcup a_{14} \circ x_4 \sqcup a_{15} \circ x_5 \sqcup b_1 \\ y_2 & = & a_{23} \circ x_3 \sqcup a_{24} \circ x_4 \sqcup a_{25} \circ x_5 \sqcup b_2 \qquad \text{16 calculations} \\ y_3 & = & a_{34} \circ x_4 \sqcup a_{35} \circ x_5 \sqcup b_3 \end{array}$$

(division due to staggered form)

$$\begin{array}{rcl} y_4 & = & a_{41} \circ x_1 \sqcup a_{42} \circ x_2 \sqcup a_{43} \circ x_3 \sqcup a_{44} \circ x_4 \sqcup a_{45} \circ x_5 \sqcup b_4 \\ y_5 & = & a_{51} \circ x_1 \sqcup a_{52} \circ x_2 \sqcup a_{53} \circ x_3 \sqcup a_{54} \circ x_4 \sqcup a_{55} \circ x_5 \sqcup b_5 \end{array} \quad \text{20 calculations}$$

Total calculations $= 16 + 20 = 36$

6.5.3.1 Generalized

Upper part:

$$\left(n^2 - n\right) - \left((n-r)^2 - (n-r)\right)$$

Lower part:

$$2n\,(n-r)$$

Total:

$$\left(n^2 - n\right) - \left((n-r)^2 - (n-r)\right) + 2n\,(n-r) = 2n^2 - r^2 - r$$

where n is the number of vertices and r is the number or rows in staggered form.

6.5.4 Gauss-Seidel calculations

The number of calculations performed during the solution of the equations in Section 5.5.3.5 can be determined as follows:

$$\begin{array}{ll} \mathbf{x}_{k+1} = \mathbf{L} \circ \mathbf{x}_{k+1} \sqcup \mathbf{R} \circ \mathbf{x}_k \sqcup \mathbf{b} & \\ x_i \leftarrow \bigsqcup_{j=1}^{n} a_{ij} \circ x_j \sqcup b_i & i = 1, \ldots, n \end{array} \tag{6.3}$$

$$\begin{array}{rcl} x_1 & \leftarrow & a_{11} \circ x_1 \sqcup a_{12} \circ x_2 \sqcup a_{13} \circ x_3 \sqcup a_{14} \circ x_4 \sqcup a_{15} \circ x_5 \sqcup b_1 \\ x_2 & \leftarrow & a_{21} \circ x_1 \sqcup a_{22} \circ x_2 \sqcup a_{23} \circ x_3 \sqcup a_{24} \circ x_4 \sqcup a_{25} \circ x_5 \sqcup b_2 \\ x_3 & \leftarrow & a_{31} \circ x_1 \sqcup a_{32} \circ x_2 \sqcup a_{33} \circ x_3 \sqcup a_{34} \circ x_4 \sqcup a_{35} \circ x_5 \sqcup b_3 \quad \text{50 calculations} \\ x_4 & \leftarrow & a_{41} \circ x_1 \sqcup a_{42} \circ x_2 \sqcup a_{43} \circ x_3 \sqcup a_{44} \circ x_4 \sqcup a_{45} \circ x_5 \sqcup b_4 \\ x_5 & \leftarrow & a_{51} \circ x_1 \sqcup a_{52} \circ x_2 \sqcup a_{53} \circ x_3 \sqcup a_{54} \circ x_4 \sqcup a_{55} \circ x_5 \sqcup b_5 \end{array}$$

6.5.4.1 Generalized

$n \times 2n = 2n^2$, where n is the number of vertices.

6.5.5 Gauss-Seidel calculations after sorting

$$\begin{aligned} x_1 &\leftarrow a_{12} \circ x_2 \sqcup a_{13} \circ x_3 \sqcup a_{14} \circ x_4 \sqcup a_{15} \circ x_5 \sqcup b_1 \\ x_2 &\leftarrow a_{23} \circ x_3 \sqcup a_{24} \circ x_4 \sqcup a_{25} \circ x_5 \sqcup b_2 \qquad \text{16 calculations} \\ x_3 &\leftarrow a_{34} \circ x_4 \sqcup a_{35} \circ x_5 \sqcup b_3 \end{aligned}$$

(division due to staggered form)

$$\begin{aligned} x_4 &\leftarrow a_{41} \circ x_1 \sqcup a_{42} \circ x_2 \sqcup a_{43} \circ x_3 \sqcup a_{44} \circ x_4 \sqcup a_{45} \circ x_5 \sqcup b_4 \qquad \text{20 calculations} \\ x_5 &\leftarrow a_{51} \circ x_1 \sqcup a_{52} \circ x_2 \sqcup a_{53} \circ x_3 \sqcup a_{54} \circ x_4 \sqcup a_{55} \circ x_5 \sqcup b_5 \end{aligned}$$

Total calculations $= 16 + 20 = 36$

6.5.5.1 Generalized

$(n^2 - n) - \left((n-r)^2 - (n-r)\right) + 2n(n-r) = 2n^2 - r^2 - r$
where n is the number of vertices and r is the number or rows in staggered form.

6.5.6 Forward and back substitution calculations

The number of calculations performed during the solution of the equations in Section 5.5.3.6 can be determined as follows:

$$\begin{aligned} &\mathbf{y}_{k+1} = \mathbf{R} \circ y_{k+1} \sqcup \mathbf{x}_k \sqcup \mathbf{b} \text{ and } \mathbf{x}_{k+1} = \mathbf{L} \circ \mathbf{x}_{k+1} \sqcup \mathbf{y}_{k+1} \\ &y_n = x_n \sqcup b_n \\ &y_i = \bigsqcup_{j=i+1}^{n} a_{ij} \circ y_j \sqcup x_i \sqcup b_i \qquad i = n-1, \ldots, 1 \\ &x_1 = y_1 \\ &x_i = \bigsqcup_{j=1}^{i-1} a_{ij} \circ x_j \sqcup y_i \qquad i = 2, \ldots, n \end{aligned} \tag{6.4}$$

$$\begin{aligned} y_5 &= x_5 \sqcup b_5 \\ y_4 &= a_{45} \circ y_5 \sqcup x_4 \sqcup b_4 \\ y_3 &= a_{34} \circ y_4 \sqcup a_{35} \circ y_5 \sqcup x_3 \sqcup b_3 \qquad 1 + 24 \text{ calculations} \\ y_2 &= a_{23} \circ y_3 \sqcup a_{24} \circ y_4 \sqcup a_{25} \circ y_5 \sqcup x_2 \sqcup b_2 \\ y_1 &= a_{12} \circ y_2 \sqcup a_{13} \circ y_3 \sqcup a_{14} \circ y_4 \sqcup a_{15} \circ y_5 \sqcup x_1 \sqcup b_1 \end{aligned}$$

$1 + 3 + 5 + 7 + 9 = 25 = n^2$

$$\begin{aligned} x_1 &= y_1 \\ x_2 &= a_{21} \circ x_1 \sqcup y_2 \\ x_3 &= a_{31} \circ x_1 \sqcup a_{32} \circ x_2 \sqcup y_3 \qquad \text{20 calculations} \\ x_4 &= a_{41} \circ x_1 \sqcup a_{42} \circ x_2 \sqcup a_{43} \circ x_3 \sqcup y_4 \\ x_5 &= a_{51} \circ x_1 \sqcup a_{52} \circ x_2 \sqcup a_{53} \circ x_3 \sqcup a_{54} \circ x_4 \sqcup y_5 \end{aligned}$$

$2 + 4 + 6 + 8 = 2(1 + 2 + 3 + 4) = 20 = 2 \times \frac{(n^2 - n)}{2} = n^2 - n$

6.5.6.1 Generalized

Total calculations $= 1 + 24 + 20 = 45 = n^2 + n^2 - n = 2n^2 - n$, where n is the number of vertices.

6.5.7 Forward and back substitution calculations after sorting

$$\begin{aligned}
y_5 &= x_5 \sqcup b_5 \\
y_4 &= a_{45} \circ y_5 \sqcup x_4 \sqcup b_4 \\
y_3 &= a_{34} \circ y_4 \sqcup a_{35} \circ y_5 \sqcup x_3 \sqcup b_3 \qquad\qquad 1 + 24 \text{ calculations} \\
y_2 &= a_{23} \circ y_3 \sqcup a_{24} \circ y_4 \sqcup a_{25} \circ y_5 \sqcup x_2 \sqcup b_2 \\
y_1 &= a_{12} \circ y_2 \sqcup a_{13} \circ y_3 \sqcup a_{14} \circ y_4 \sqcup a_{15} \circ y_5 \sqcup x_1 \sqcup b_1
\end{aligned}$$

n^2 calculations

$$\begin{aligned}
x_1 &= y_1 \\
x_2 &= y_2 \\
x_3 &= y_3
\end{aligned}$$

——————————————————— (division due to staggered form)

$$\begin{aligned}
x_4 &= a_{41} \circ x_1 \sqcup a_{42} \circ x_2 \sqcup a_{43} \circ x_3 \sqcup y_4 \qquad\qquad 14 \text{ calculations} \\
x_5 &= a_{51} \circ x_1 \sqcup a_{52} \circ x_2 \sqcup a_{53} \circ x_3 \sqcup a_{54} \circ x_4 \sqcup y_5
\end{aligned}$$

$\left(n^2 - n\right) - \left(r^2 - r\right)$ calculations

6.5.7.1 Generalized

Total calculations $= 1 + 24 + 14 = 39 = n^2 + \left(n^2 - n\right) - \left(r^2 - r\right) = 2n^2 - r^2 - n + r$
where n is the number of vertices and r is the number or rows in staggered form.

6.5.8 Interpretation of results

General equations for determining the number of calculations for the solution methods of Gauss, Jacobi and Gauss-Seidel, as well as the forward and back substitution method were determined in Sections 6.5.1 to 6.5.7. The number of calculations were determined per iteration for the iterative methods. Therefore, the number of iterations for an iterative method to reach a solution determines the total amount of calculations for the method. The test data in Appendix D was used to compare the different methods.

6.5.8.1 Number of iterations

The data in Appendix D.1 was divided into groups of graphs of similar size. The average number of iterations for the Jacobi, Gauss-Seidel and forward and back substitution methods were determined for each group. The results are shown in Figures 6.8 and 6.9 for the unsorted and sorted cases, respectively. It is obvious from these graphs that the Jacobi method required the most iterations to reach a solution for the test graphs, while the forward and back substitution method required the least.

6.5.8.2 Influence of sorting on the number of iterations

The grouped and averaged data of Appendix D.1 was also used to plot the difference in the number of iterations before and after sorting, difference $=$ number of iterations$_{\text{after}}$ $-$ number of iterations$_{\text{before}}$, for the Jacobi, Gauss-Seidel and forward and back substitution methods. The results are shown in Figure 6.10. A positive result means that more iterations were required after sorting than before sorting and a negative result means that less iterations were required after sorting than before sorting.

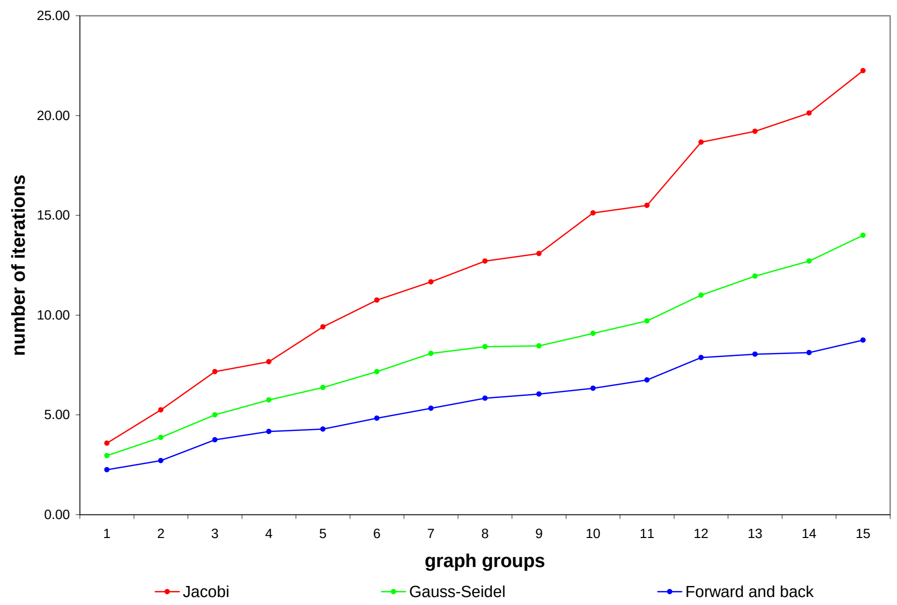


Figure 6.8: Number of iterations for unsorted graphs

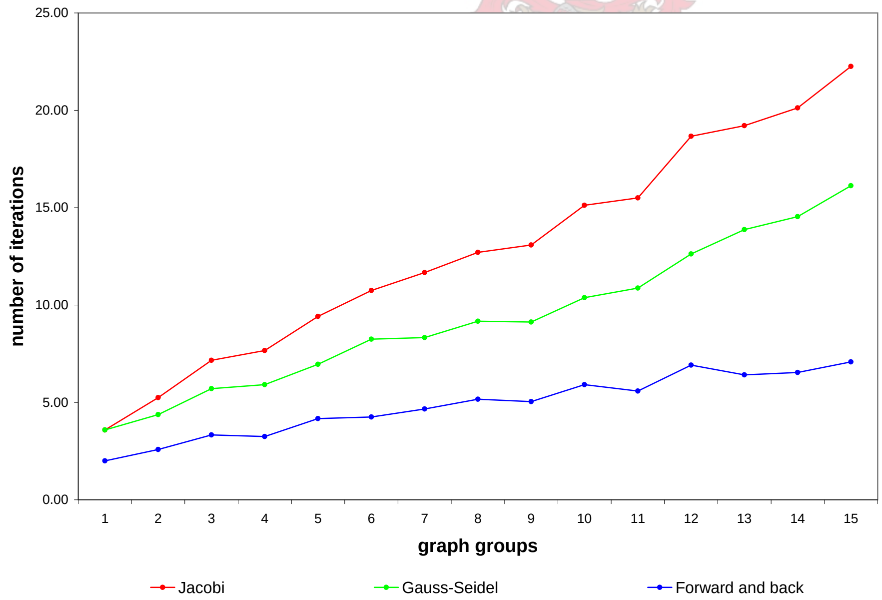


Figure 6.9: Number of iterations for sorted graphs

Figure 6.10: Difference in number of iterations before and after sorting

It is very clear from the figure that the number of iterations for the Jacobi method is independent of whether the graph has been sorted and relabelled or not. The number of iterations for the Gauss-Seidel and forward and back substitution methods can either increase or decrease after sorting. It is obvious from the average values in the figures that the required number of iterations for the Gauss-Seidel method increases in most cases, while the required number of iterations for the forward and back substitution method decreases in most cases.

6.5.8.3 Number of calculations

The results in Sections 6.5.2 to 6.5.7 show that the Jacobi and Gauss-Seidel algorithms require an equal amount of calculations per iteration, while the forward and back substitution algorithm requires a little less. However, the total number of required calculations for the iterative methods is dependent on the number of iterations necessary for each algorithm. The direct Gauss algorithm requires a constant amount of calculations for a graph of given size.

The data in Appendix D.2 was grouped and the average number of calculations for the Jacobi, Gauss-Seidel and forward and back substitution methods were plotted for each group. The results are shown in Figures 6.11 and 6.12 for the unsorted and sorted cases, respectively. It is obvious that the Jacobi method requires the most calculations in most cases, while the forward and back substitution method requires the least. This can be expected from the results in Section 6.5.8.1.

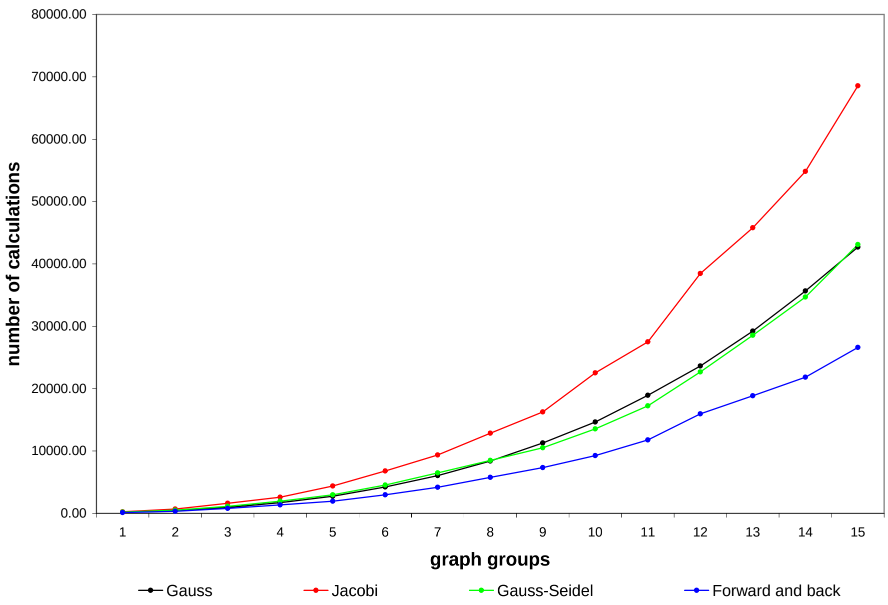


Figure 6.11: Number of calculations for unsorted graphs

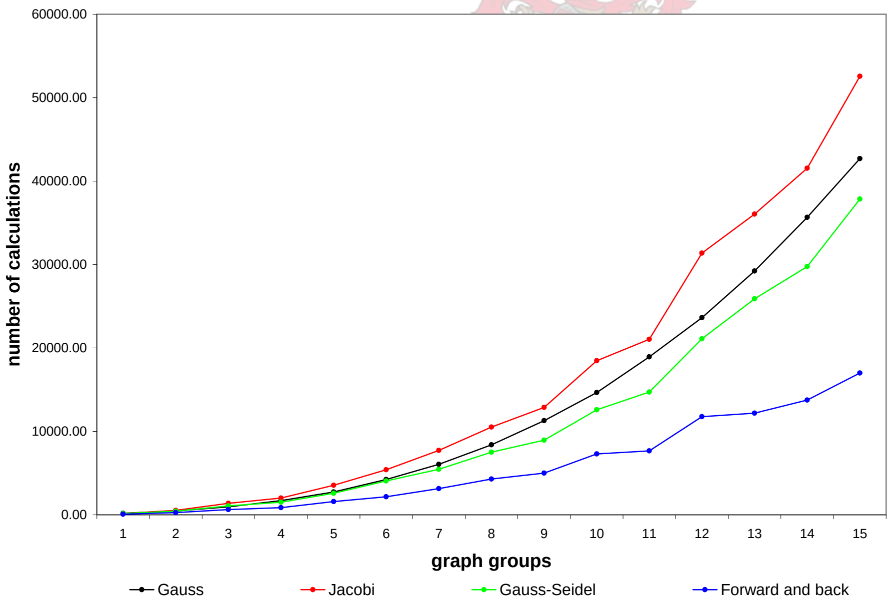


Figure 6.12: Number of calculations for sorted graphs

6.5.8.4 Influence of sorting on the number of calculations

The grouped and averaged data of Appendix D.2 was also used to plot the difference in the number of calculations before and after sorting, difference $=$ number of calculations$_{\text{after}}$ $-$ number of calculations$_{\text{before}}$, for the Jacobi, Gauss-Seidel and forward and back substitution methods. The results are shown in Figure 6.13. A positive result means that more calculations were done after sorting than before sorting and a negative result means that less calculations were done after sorting than before sorting.

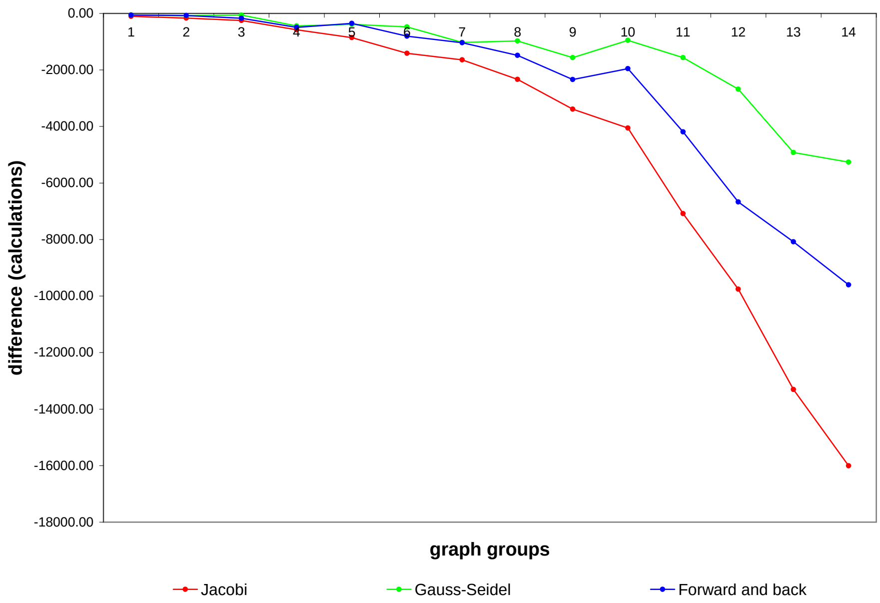


Figure 6.13: Difference in number of calculations before and after sorting

An obvious reduction in the number of calculations due to sorting is obvious for all the methods. This is most pronounced for the Jacobi method and least for the Gauss-Seidel method. This can be ascribed to the fact that the number of iterations remains constant before and after sorting for the Jacobi method. Therefore, the reduction in the number of calculations per iteration after sorting leads to a total reduction of calculations for the Jacobi method. In the case of the Gauss-Seidel, and to a minor degree the forward and back substitution method, there may be an increase in the number of iterations required after sorting, which counters the reduction in the number of calculations per iteration.

6.5.8.5 Duration

The extreme values of the data in Appendix D.3 were removed, and the remaining values grouped according to similar graph sizes. The average values for the duration of the grouped graphs were plotted for the Jacobi, Gauss-Seidel and forward and back substitution methods. The results are shown in Figures 6.14 and 6.15, for the unsorted and sorted cases, respectively.

It is obvious from these figures that the implementation of the direct Gauss elimination method is faster than that of the iterative methods for both the unsorted and sorted test graphs. The Jacobi method is the slowest of the iterative methods in general, while the forward and back substitution method is the fastest.

There are pronounced jumps in the duration values on the graphs for both the unsorted and sorted test graphs. Although the graphs have been sorted according to size, the size of a graph is only an indication of the possibilities of paths. The number of resulting paths, as well as their lengths, depend on the structure of the graph and the choice of vertex to which the paths have to be calculated. A smaller graph may have more results than a larger graph and the number of results for graphs of the same size may vary considerably. Therefore, the duration of some smaller graphs may be longer than for larger graphs and the duration may vary considerably for graphs of the same size.

The differences, difference $= \text{duration}_{\text{after sorting}} - \text{duration}_{\text{before sorting}}$, between the sorted and unsorted durations, for the Jacobi, Gaualsoss-Seidel and the forward and back substitution methods were calculated. The extreme values were removed and the long tail of entering values cropped. The remaining values were grouped and averaged. These values were plotted and the results are shown in Figure 6.16. A positive result means a longer duration after sorting than before sorting and a negative result means a shorter duration after sorting than before sorting.

The Jacobi method has the most pronounced reduction in execution time. This is a result of the constant number of iterations before and after sorting, which means the number of calculations will either remain the same or be reduced, depending on the renumbering. There are cases for which the duration is longer after sorting than before sorting, most notably for the Gauss-Seidel method. This can be ascribed to the fact that this method has shown a greater increase in the necessary number of iterations after sorting.

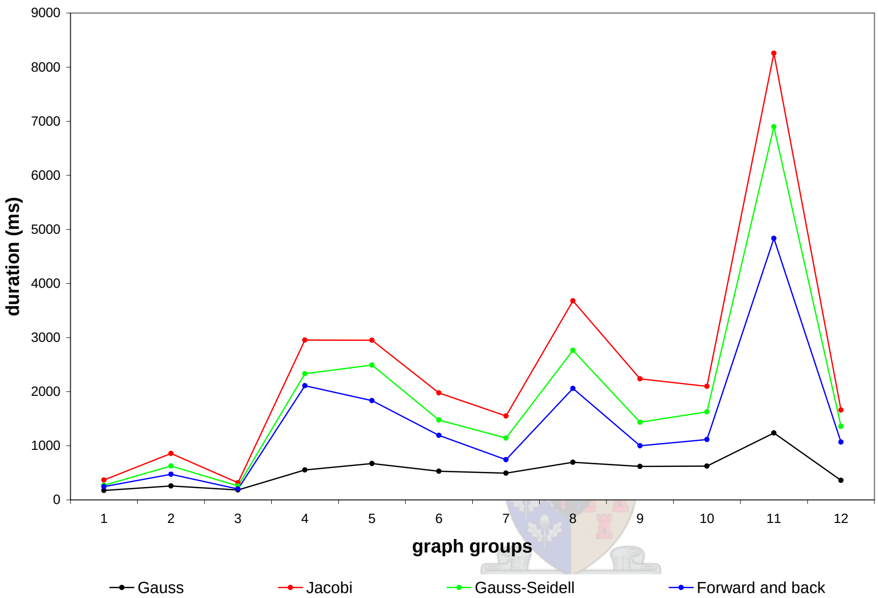


Figure 6.14: Unsorted durations

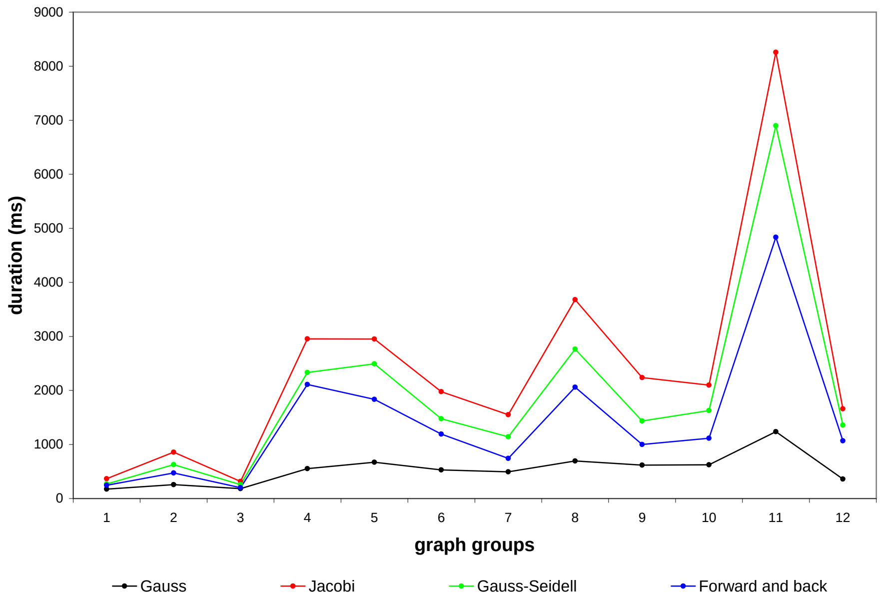


Figure 6.15: Sorted durations

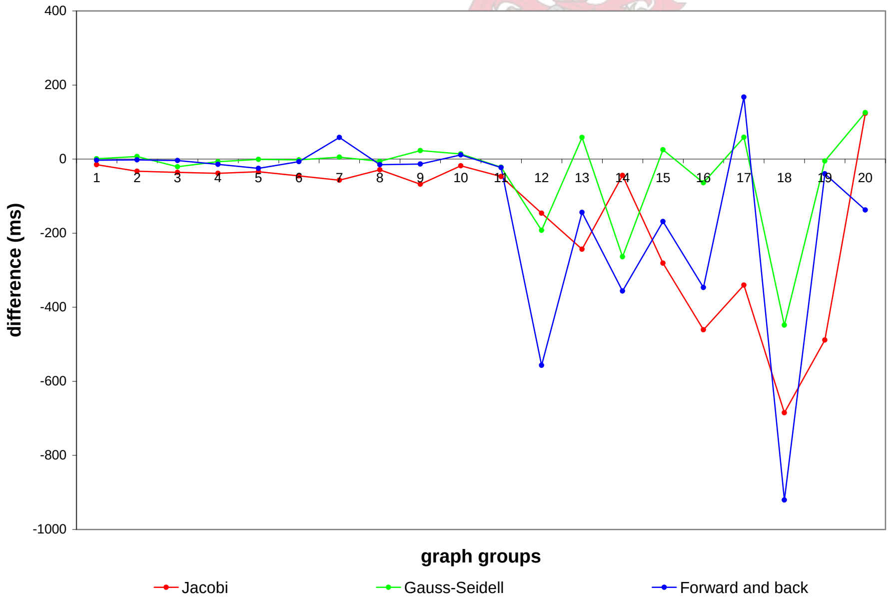


Figure 6.16: Differences in duration between unsorted and sorted

Chapter 7

Conclusions

The relation "has to be executed before" in the set of tasks of an engineering process model was successfully described by a directed graph. Therefore, there exists a powerful mathematical basis in graph theory to determine the dependencies of tasks in a process model upon each other. The logical sequence of tasks, as well as the logical critical path can be determined algebraically.

The logical sequence of tasks is important to ensure that tasks in an engineering process are executed in the correct order to avoid costly delays. The logical sequence of tasks was determined by a topological sorting of the directed graph. However, this could only be done if the graph was acyclic. Therefore, the detection of cycles in the graph became very important. The cycles was detected indirectly by finding the strongly connected components of the graph. Different methods of determining the strongly connected components, the methods of Kosaraju, Tarjan and Gabow, were discussed and implemented. This led to the reduced graph of super vertices, which is a directed acyclic graph. This graph could successfully be sorted topologically to find the logical sequence of tasks.

The logical critical path is the longest elementary path through a graph. The importance of this path lies in the fact that the tasks on this path has a direct influence on the total duration of a project. The tasks on this path need to be executed on time to avoid delays. The elementary path algebra was discussed and implemented to find all the elementary paths to or from a given task in a graph. In addition to the logical critical path, the elementary path algebra is useful to determine the dependencies of tasks on other tasks in an engineering process.

The elementary path algebra was reduced to a system of equations. Different methods of solution were discussed and implemented, the direct Gauss elimination method, followed by a back substitution, as well as the iterative methods of Jacobi and Gauss-Seidel and the forward and back substitution method. The performance of these methods was tested, using random test data, and compared to each other. The number of calculations required to execute each method was used, as well as the duration of the execution.

Relabelling of vertices of a graph after topological sorting was used in an attempt to improve the performance of the methods. Such a relabelling leads to a partially staggered coefficient matrix. Knowledge of this structure could be used to reduce the number of calculations per iteration for the iterative methods.

On average, the Jacobi method required the most iterations to reach a solution, while the forward and back substitution method required the least. The number of iterations for the Jacobi method remained constant after sorting, while an unexpected change in the number of iterations required by the Gauss-Seidel and the forward and back substitution methods was encountered. Therefore, relabelling of the vertices always led to a reduction of calculations for the method of Jacobi, while the gain in the reduction of calculations per iteration was sometimes countered by an increase in the necessary number of iterations for the Gauss-Seidel and forward and back substitution methods. Despite the increase in number of iterations for the Gauss-Seidel and forward and back substitution methods in some cases, the number of calculations decreased on average.

The direct Gauss elimination method was found to be the fastest on average with the current implementation of the methods. The Jacobi method was the slowest of the iterative methods, while the forward and back substitution method was the fastest. Sorting and relabelling of the graphs led to a general decrease in duration of the iterative methods.

Bibliography

[1] W Huhnt. Progress measurement in planning processes on the base of process models. In *Xth International Conference on Computing in Civil and Building Engineering*, Weimar, Germany, June 02-04 2004. 1

[2] W Huhnt and J Lawrence. Modelling of engineering planning process. In *25th Annual Symposium on Information Technology in Engineering*, Stellenbosch, 2003. South African Institution of Civil Engineering Division of Information Technology. 1

[3] Robert Cecil Martin. *UML for Java Programmers*. Prentice Hall, 2002. 65

[4] PJ Pahl and R Damrath. *Mathematical Foundations of Computational Engineering: A Handbook*. Springer Verlag, 1 edition, 2001. 5, 17, 45

[5] Robert Sedgewick. *Algorithms in Java, Part 5: Graph Algorithms*. Addison Wesley, 3 edition, 2003. 32

Appendix A

Elementary paths - example from Section 5.3.6

All elementary paths to vertex 4

Paths from vertex 1

[<1,7,5,6,2,10,8,3,4> ,
<1,6,7,8,9,3,4> ,
<1,7,5,6,4> ,
<1,6,7,3,4> ,
<1,6,7,8,4> ,
<1,5,6,2,10,8,9,3,4> ,
<1,6,7,8,3,4> ,
<1,7,8,3,4> ,
<1,2,10,8,9,3,4> ,
<1,7,8,4> ,
<1,5,6,2,10,8,3,4> ,
<1,7,5,6,10,8,3,4> ,
<1,6,10,8,4> ,
<1,5,6,10,8,4> ,
<1,2,10,8,4> ,
<1,7,5,6,10,8,9,3,4> ,
<1,6,2,10,8,3,4> ,
<1,7,3,4> ,
<1,6,4> ,
<1,2,10,8,3,4> ,
<1,6,10,8,9,3,4> ,
<1,5,6,2,10,8,4> ,
<1,5,6,7,3,4> ,
<1,5,6,10,8,9,3,4> ,
<1,6,2,10,8,9,3,4> ,
<1,5,6,10,8,3,4> ,
<1,5,6,4> ,
<1,5,6,7,8,4> ,
<1,7,5,6,2,10,8,4> ,
<1,7,8,9,3,4> ,
<1,6,10,8,3,4> ,
<1,7,5,6,2,10,8,9,3,4> ,
<1,6,2,10,8,4> ,
<1,5,6,7,8,9,3,4> ,
<1,7,5,6,10,8,4> ,
<1,5,6,7,8,3,4>]

Paths from vertex 5

[<5,6,2,10,8,3,4> ,
<5,6,2,10,8,4> ,
<5,6,7,8,4> ,
<5,6,7,8,3,4> ,
<5,6,7,8,9,3,4> ,
<5,6,10,8,9,3,4> ,
<5,6,10,8,4> ,
<5,6,4> ,
<5,6,10,8,3,4>]
<5,6,7,3,4> ,
<5,6,2,10,8,9,3,4> ,

Paths from vertex 6

[<6,10,8,3,4> ,
<6,10,8,9,3,4> ,
<6,4> ,
<6,2,10,8,9,3,4> ,
<6,2,10,8,3,4> ,
<6,7,8,9,3,4> ,
<6,70,8,4> ,
<6,10,8,4> ,
<6,7,8,3,4>]
<6,2,10,8,4> ,
<6,7,3,4> ,

Paths from vertex 7

[<7,8,3,4> ,
<7,5,6,10,8,4> ,
<7,5,6,4> ,
<7,8,9,3,4> ,
<7,5,6,2,10,8,9,3,4> ,
<7,5,6,2,10,8,4> ,
<7,3,4> ,
<7,5,6,10,8,3,4> ,
<7,8,4>]
<7,5,6,10,8,9,3,4> ,
<7,5,6,2,10,8,3,4> ,

Paths from vertex 2

[<2,10,8,9,3,4> , <2,10,8,4> , <2,10,8,3,4>]

Paths from vertex 8

[<8,3,4> , <8,9,3,4> , <8,4>]

Paths from vertex 10

[<10,8,4> , <10,8,9,3,4> , <10,8,3,4>]

Paths from vertex 3

[<3,4>]

Paths from vertex 4

[<4> , <%>]

Paths from vertex 9

[<9,3,4>]

Appendix B

UML implementation of graph model

B.1 Graph

Graph	
+ NEUTRAL : int + KOSARAJU : int + TARJAN : int + GABOW : int + TO_VERTEX : int + FROM_VERTEX : int - vertexSet : HashSet - edgeSet : HashSet - superVertexSet: HashSet - superEdgeSet: HashSet	- SCCs : HashMap - steps : ArrayList - equation : Equation - pathCode : int - toVertexNumber : int - fromVertexNumber : int
+ Graph() + addVertex() : Vertex + addEdge(startVertex : Vertex, endVertex : Vertex) : Edge + addSuperVertex() : SuperVertex + addSuperEdge(startVertex : SuperVertex, endVertex : SuperVertex) : SuperEdge + getVertexSet() : HashSet + getEdgeSet() : HashSet + getSuperVertexSet() : HashSet + getSuperEdgeSet() : HashSet + generateRandomGraph(n : int, t : int, s: int)	

Graph (continued)
+ dfs() + Kosaraju() + Tarjan() + Gabow() + reducedGraph() + topologicalSortingReducedGraph() + reLabel() + isSorted() : boolean + getSCCs() : HashMap + setPathCode(code : int) + setToVertex(vertex : Vertex) + setFromVertex(vertex : Vertex) + getEquation() : Equation

B.2 Vertex

Vertex
- successors : HashMap - predecessors : HashMap - label : int - SCCno : int - step : int - superVertex : SuperVertex - weight : ElementaryPathSet
+ Vertex() + addSuccessor(vertex : Vertex, edge : Edge) + addPredecessor(vertex : Vertex, edge : Edge) + getSuccessors() : HashSet + getPredecessors() : HashSet + setLabel(label : int) + setSCCno(SCCno : int) + setStep(step : int) + getLabel() : int + getSCCno() : int + getStep() : int + setWeight(weight : ElementaryPathSet) + getWeight() : ElementaryPathSet

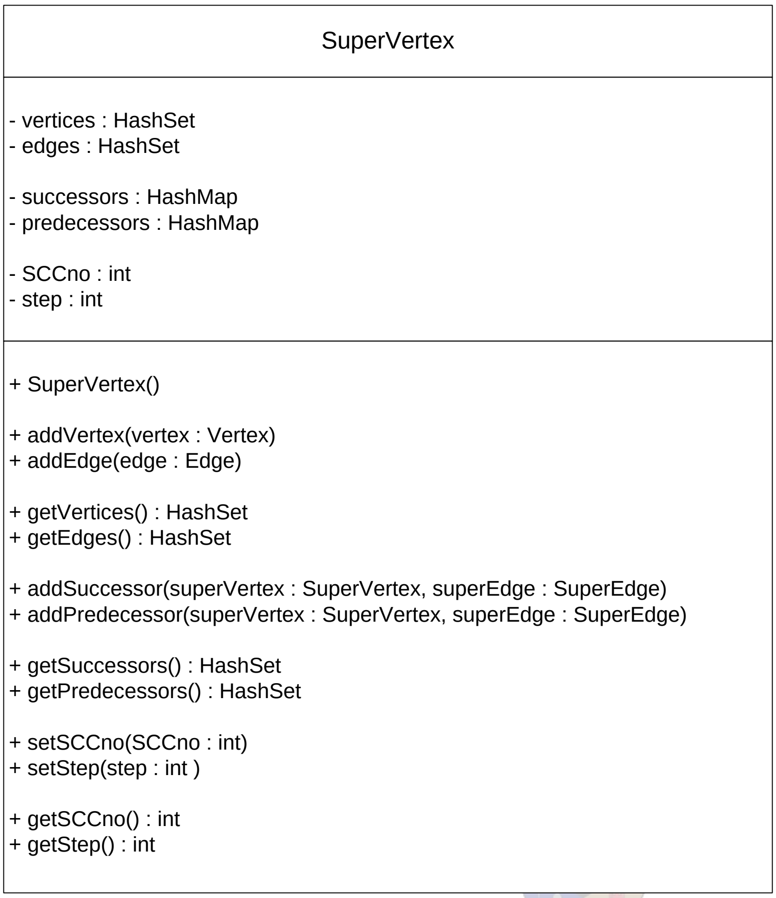
The edges entering and leaving a vertex is mapped to the successor and predecessor vertices to which it points. Vertices has the same strongly connected component numbers and step numbers in the logical sequence of tasks as the super vertices to which it is mapped. All the paths, to or from a vertex , which has been calculated is stored as the weight of the vertex.

B.3 Edge

Edge	
+ TREE : int + BACK : int + DOWN : int + CROSS : int + INTERNAL : int + TRANSITION : int - startVertex : Vertex - endVertex : Vertex	- superVertex : SuperVertex - superEdge : SuperEdge - typeOfEdgeDFS : int - typeOfEdgeReduced : int - label : String - weight : ElementaryPath
+ Edge (startVertex : Vertex, endVertex : Vertex) + getStartVertex() : Vertex + getEndVertex() : Vertex + setLabel(label : String) + getLabel() : String + setTypeOfEdgeDFS(type : int) + setTypeOfEdgeReduced(type : int) + getTypeOfEdgeDFS() : int + getTypeOfEdgeReduced() : int + setSuperVertex(superVertex : SuperVertex) + setSuperEdge(superEdge : SuperEdge) + getSuperVertex() : SuperVertex + getSuperEdge() : SuperEdge	

An edge is a relationship between two vertices and is represented by an ordered vertex pair. Edges can can classified as either, tree, back, down or cross edges during a depth-first search (DFS). Edges can also be classified as either an internal or an transition edge, depending on whether it is mapped to a super vertex or a super edge in the reduced graph (see Section 3.4.2.8).

B.4 SuperVertex



A graph can be decomposed into its strongly connected components (see the strongly connected component example in Section 3.4.2.10). Each vertex in the reduced graph represents a strongly connected component in the graph.

Super vertices are used to store the strongly connected components . A super vertex corresponds to a vertex in the reduced graph. The set of vertices of a strongly connected component, as well as the set of edges connecting these vertices are stored as part of the super vertex. The successors and predecessors of the super vertex are the adjacent super vertices in the reduced graph. As with the vertices of the graph the super edges connecting the super vertex with its successors and predecessors are mapped to these super vertices. Since the reduced graph is acyclic it can be sorted topologically (see Section 3.4.3.3) into steps.

B.5 SuperEdge

```
SuperEdge
--------------------------------------------------
- startVertex : SuperVertex
- endVertex : SuperEdge

- edges : HashSet
--------------------------------------------------
+ SuperEdge(startVertex : SuperVertex, endVertex : SuperVertex)

+ getStartVertex() : SuperVertex
+ getEndVertex() : SuperVertex

+ addEdge(edge : Edge)

+ getEdges() : HashSet
```

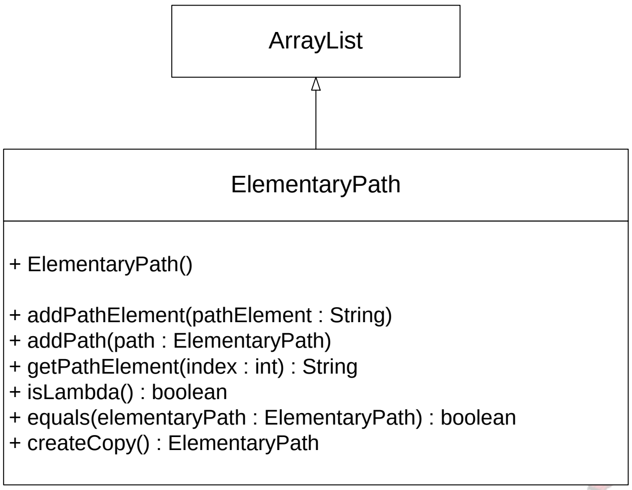
Super edges are edges connecting vertices in the reduced graph. Super edges connect super vertices. The set of edges connecting vertices in one strongly connected component to vertices in another strongly connected component, and which is therefore not part of any of the strongly connected components, is stored as part the super edge connecting the two super vertices.

B.6 Equation

```
Equation
--------------------------------------------------
+ GAUSS : int
+ JACOBI : int
+ GAUSS_SEIDEL : int
+ FORWARD_BACK : int

- algebra : ElementaryPathAlgebra

- matrix : ElementaryPathSet[ ][ ]
- x : ElementaryPathSet[ ]
- b : ElementaryPathSet[ ]

- mapping : Vertex[ ]
--------------------------------------------------
+ analyze(methodOfSolution : int)

- setVertexIndices()
- setSystemMatrix()
- transposeSystemMatrix()
- setSystemVector(unitNumber : int)

- eliminateGauss()
- substitute()

- eliminateJacobi()
- eliminateGaussSeidel()
- eliminateForwardBack()
```

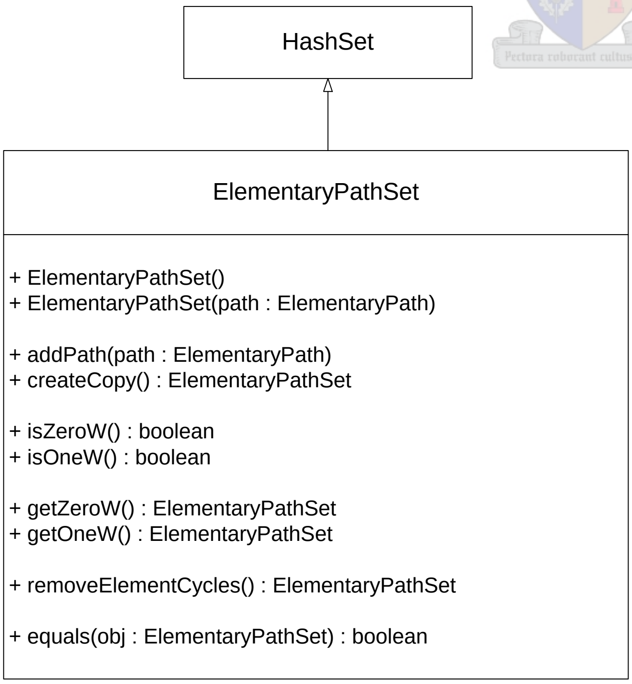
The system of equations of the literal path algebra for elementary paths, consisting of the elementary weight matrix, the solution vector and the unit vector, is represented by the class Equation. Vertices are mapped to their indices in the elementary weight matrix. The system of equations can be solved by any of the methods of solution mentioned in Section 5.5.

B.7 ElementaryPath



The class ElementaryPath is used to store elementary paths (see Section 3.4.1.13) in the graph. An empty path is represented by $\{\lambda\}$.

B.8 ElementaryPathSet



The class ElementaryPathSet is used to store a set of elementary paths. The zero set is equal to $\{\}$ and the one set is equal to $\{\lambda\}$. Elementary cycles are not stored as part of the elementary path set.

B.9 ElementaryPathAlgebra

ElementaryPathAlgebra
+ LAMBDA : String
+ zero() : ElementaryPathSet + one() : ElementaryPathSet + union(a : ElementaryPathSet, b : ElementaryPathSet) : ElementaryPathSet + concatenate(a : ElementaryPathSet, b : ElementaryPathSet) : ElementaryPathSet + elementClosure(element : ElementaryPathSet) : ElementaryPathSet

The elementary path algebra operations are implemented in the Class ElementaryPathAlgebra.

Appendix C

Useful algebraic equations

$$1 + 2 + 3 + \ldots + n = \frac{n^2 + n}{2} \tag{C.1}$$

$$1 + 3 + 5 + \ldots = n^2, \text{ where } n = \text{number of odd numbers} \tag{C.2}$$

$$1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n\,(n+1)\,(2n+1)}{6} \tag{C.3}$$

Appendix D

Test graph data

D.1 Iterations

	Vertices	Edges	Jacobi			Gauss-Seidel			Forward and back		
			unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference
1	5	4	3	3	0	3	3	0	2	2	0
2	5	5	3	3	0	2	3	1	2	2	0
3	5	5	3	3	0	3	3	0	2	2	0
4	5	4	3	3	0	3	3	0	2	2	0
5	5	5	3	3	0	3	3	0	2	2	0
6	5	3	3	3	0	3	3	0	2	2	0
7	5	4	3	3	0	2	3	1	2	2	0
8	5	5	3	3	0	2	3	1	2	2	0
9	5	4	3	3	0	2	3	1	2	2	0
10	5	5	3	3	0	2	3	1	2	2	0
11	6	5	4	4	0	3	4	1	2	2	0
12	6	7	4	4	0	3	4	1	2	2	0
13	6	6	4	4	0	4	4	0	3	2	-1
14	6	5	4	4	0	3	4	1	2	2	0
15	6	7	4	4	0	4	4	0	2	2	0
16	6	6	4	4	0	3	4	1	3	2	-1
17	6	7	4	4	0	3	4	1	3	2	-1
18	6	5	4	4	0	4	4	0	3	2	-1
19	6	7	4	4	0	3	4	1	3	2	-1
20	6	6	4	4	0	3	4	1	2	2	0
21	7	8	4	4	0	3	4	1	3	2	-1
22	7	8	4	4	0	3	4	1	2	2	0
23	7	8	4	4	0	4	4	0	2	2	0
24	7	7	4	4	0	3	4	1	2	2	0
25	7	10	6	6	0	5	4	-1	3	3	0
26	7	9	6	6	0	5	5	0	3	3	0
27	7	7	4	4	0	4	4	0	2	2	0
28	7	8	4	4	0	3	4	1	2	2	0
29	7	7	4	4	0	3	4	1	2	2	0
30	7	8	4	4	0	4	4	0	2	2	0
31	8	9	6	6	0	4	6	2	3	3	0
32	8	11	6	6	0	4	4	0	3	3	0
33	8	10	4	4	0	4	4	0	2	2	0
34	8	6	4	4	0	4	4	0	3	2	-1
35	8	7	4	4	0	3	4	1	2	2	0

	Vertices	Edges	Jacobi			Gauss-Seidel			Forward and back		
			unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference
36	8	9	4	4	0	3	4	1	2	2	0
37	8	11	6	6	0	4	5	1	3	3	0
38	8	11	6	6	0	3	3	0	3	3	0
39	8	9	4	4	0	2	4	2	2	2	0
40	8	9	6	6	0	3	4	1	3	3	0
41	9	13	6	6	0	4	4	0	3	3	0
42	9	13	6	6	0	4	3	-1	3	3	0
43	9	14	6	6	0	4	4	0	3	4	1
44	9	9	4	4	0	4	4	0	3	2	-1
45	9	14	8	8	0	6	7	1	4	3	-1
46	9	14	6	6	0	4	6	2	3	3	0
47	9	10	6	6	0	4	4	0	3	3	0
48	9	10	6	6	0	5	6	1	3	2	-1
49	9	8	4	4	0	3	4	1	3	2	-1
50	9	6	4	4	0	3	4	1	3	2	-1
51	10	12	6	6	0	4	4	0	4	3	-1
52	10	16	8	8	0	6	6	0	4	4	0
53	10	13	6	6	0	5	4	-1	3	3	0
54	10	17	8	8	0	7	7	0	4	3	-1
55	10	15	6	6	0	5	6	1	3	3	0
56	10	12	6	6	0	4	5	1	4	3	-1
57	10	17	8	8	0	5	5	0	4	4	0
58	10	17	8	8	0	5	5	0	4	4	0
59	10	13	6	6	0	4	6	2	3	3	0
60	10	11	6	6	0	4	4	0	3	4	1
61	11	16	7	7	0	4	6	2	3	3	0
62	11	16	9	9	0	7	7	0	5	4	-1
63	11	18	9	9	0	5	6	1	5	5	0
64	11	15	7	7	0	5	6	1	4	3	-1
65	11	17	7	7	0	4	7	3	4	2	-2
66	11	17	9	9	0	6	7	1	4	4	0
67	11	20	9	9	0	7	7	0	5	5	0
68	11	17	9	9	0	6	6	0	4	4	0
69	11	15	5	5	0	4	5	1	2	2	0
70	11	9	5	5	0	3	5	2	2	2	0
71	12	23	11	11	0	8	7	-1	5	5	0
72	12	19	9	9	0	6	8	2	5	3	-2
73	12	15	7	7	0	4	5	1	4	4	0
74	12	17	7	7	0	5	5	0	4	3	-1
75	12	13	5	5	0	3	5	2	3	2	-1
76	12	19	7	7	0	5	6	1	4	3	-1
77	12	15	7	7	0	5	5	0	4	3	-1
78	12	12	5	5	0	4	5	1	3	2	-1
79	12	14	7	7	0	5	5	0	4	3	-1
80	12	21	9	9	0	7	7	0	5	5	0
81	13	19	7	7	0	5	5	0	4	4	0
82	13	12	5	5	0	3	5	2	3	2	-1
83	13	24	11	11	0	9	7	-2	6	5	-1
84	13	19	9	9	0	7	7	0	4	4	0
85	13	14	5	5	0	4	5	1	3	2	-1
86	13	17	7	7	0	6	7	1	4	2	-2
87	13	19	9	9	0	6	5	-1	4	5	1
88	13	20	9	9	0	7	5	-2	5	3	-2
89	13	18	7	7	0	5	6	1	4	3	-1
90	13	26	13	13	0	10	8	-2	6	5	-1
91	14	18	8	8	0	6	6	0	4	3	-1
92	14	23	10	10	0	7	8	1	5	4	-1
93	14	20	8	8	0	6	6	0	5	3	-2
94	14	15	6	6	0	6	6	0	3	2	-1

	Vertices	Edges	Jacobi			Gauss-Seidel			Forward and back		
			unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference
95	14	20	10	10	0	7	7	0	6	4	-2
96	14	14	6	6	0	6	6	0	3	2	-1
97	14	17	6	6	0	5	6	1	4	2	-2
98	14	20	8	8	0	6	6	0	3	4	1
99	14	24	10	10	0	7	8	1	4	4	0
100	14	26	12	12	0	8	7	-1	5	6	1
101	15	19	8	8	0	5	6	1	3	3	0
102	15	20	8	8	0	5	6	1	4	4	0
103	15	19	8	8	0	5	6	1	4	3	-1
104	15	29	12	12	0	8	8	0	5	6	1
105	15	21	10	10	0	6	8	2	4	4	0
106	15	23	10	10	0	7	5	-2	5	4	-1
107	15	20	8	8	0	5	6	1	4	4	0
108	15	22	10	10	0	7	8	1	4	4	0
109	15	25	10	10	0	8	8	0	5	3	-2
110	15	28	10	10	0	7	7	0	5	4	-1
111	16	28	12	12	0	8	8	0	5	6	1
112	16	24	8	8	0	5	7	2	4	4	0
113	16	25	10	10	0	7	7	0	5	4	-1
114	16	22	8	8	0	5	7	2	3	3	0
115	16	21	8	8	0	4	6	2	4	4	0
116	16	22	8	8	0	6	7	1	4	3	-1
117	16	23	10	10	0	8	7	-1	4	5	1
118	16	29	12	12	0	8	9	1	6	6	0
119	16	28	12	12	0	7	7	0	5	6	1
120	16	20	8	8	0	6	7	1	4	4	0
121	17	24	9	9	0	5	8	3	4	3	-1
122	17	34	13	13	0	8	8	0	6	6	0
123	17	28	11	11	0	7	8	1	5	5	0
124	17	22	9	9	0	6	9	3	3	3	0
125	17	28	11	11	0	8	9	1	5	4	-1
126	17	28	11	11	0	7	9	2	6	5	-1
127	17	23	11	11	0	8	8	0	5	5	0
128	17	22	9	9	0	5	8	3	4	3	-1
129	17	22	7	7	0	6	7	1	3	2	-1
130	17	28	11	11	0	6	8	2	5	6	1
131	18	28	11	11	0	8	8	0	4	5	1
132	18	34	15	15	0	10	11	1	6	6	0
133	18	23	9	9	0	6	9	3	5	3	-2
134	18	25	9	9	0	5	8	3	4	3	-1
135	18	27	13	13	0	9	8	-1	7	6	-1
136	18	26	11	11	0	7	7	0	6	4	-2
137	18	18	7	7	0	5	7	2	3	2	-1
138	18	34	13	13	0	10	9	-1	6	5	-1
139	18	27	9	9	0	6	7	1	3	3	0
140	18	30	13	13	0	10	8	-2	6	6	0
141	19	32	13	13	0	10	11	1	6	5	-1
142	19	29	9	9	0	4	6	2	4	3	-1
143	19	31	13	13	0	9	7	-2	5	5	0
144	19	27	11	11	0	7	10	3	5	4	-1
145	19	28	11	11	0	8	9	1	5	4	-1
146	19	36	15	15	0	11	10	-1	7	6	-1
147	19	35	13	13	0	8	8	0	6	5	-1
148	19	29	13	13	0	9	10	1	5	5	0
149	19	31	13	13	0	9	7	-2	6	5	-1
150	19	29	13	13	0	9	9	0	6	3	-3
151	20	32	11	11	0	8	7	-1	5	5	0
152	20	39	13	13	0	9	10	1	6	6	0
153	20	28	9	9	0	6	7	1	5	3	-2

	Vertices	Edges	Jacobi			Gauss-Seidel			Forward and back		
			unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference
154	20	28	11	11	0	7	9	2	5	5	0
155	20	37	12	12	0	8	6	-2	5	5	0
156	20	34	14	14	0	10	9	-1	5	7	2
157	20	39	17	17	0	11	11	0	8	8	0
158	20	33	9	9	0	6	8	2	4	3	-1
159	20	28	11	11	0	8	9	1	5	4	-1
160	20	30	11	11	0	9	8	-1	5	5	0
161	21	28	9	9	0	7	6	-1	4	2	-2
162	21	35	13	13	0	8	9	1	6	6	0
163	21	26	9	9	0	6	8	2	5	3	-2
164	21	37	14	14	0	9	8	-1	6	7	1
165	21	27	7	7	0	5	7	2	3	2	-1
166	21	33	11	11	0	7	9	2	6	5	-1
167	21	29	9	9	0	8	7	-1	4	3	-1
168	21	32	12	12	0	8	9	1	6	5	-1
169	21	31	11	11	0	8	8	0	5	5	0
170	21	35	15	15	0	10	11	1	7	7	0
171	22	37	13	13	0	10	9	-1	6	5	-1
172	22	40	13	13	0	9	10	1	7	4	-3
173	22	43	19	19	0	11	12	1	9	9	0
174	22	34	13	13	0	9	7	-2	5	6	1
175	22	38	13	13	0	10	10	0	6	6	0
176	22	41	16	16	0	9	10	1	6	7	1
177	22	31	11	11	0	8	11	3	5	4	-1
178	22	33	10	10	0	7	7	0	5	5	0
179	22	33	10	10	0	7	8	1	4	5	1
180	22	39	12	12	0	8	8	0	6	6	0
181	23	37	13	13	0	8	9	1	5	4	-1
182	23	29	10	10	0	7	8	1	5	3	-2
183	23	41	16	16	0	11	10	-1	7	6	-1
184	23	33	10	10	0	6	8	2	5	4	-1
185	23	39	16	16	0	10	10	0	7	6	-1
186	23	29	10	10	0	6	9	3	4	3	-1
187	23	35	12	12	0	8	10	2	6	5	-1
188	23	38	14	14	0	10	9	-1	7	6	-1
189	23	45	16	16	0	10	12	2	7	6	-1
190	23	29	10	10	0	7	8	1	5	4	-1
191	24	35	10	10	0	7	8	1	5	4	-1
192	24	34	12	12	0	6	8	2	6	4	-2
193	24	42	13	13	0	8	10	2	6	6	0
194	24	40	14	14	0	10	9	-1	7	5	-2
195	24	34	12	12	0	7	8	1	5	4	-1
196	24	41	13	13	0	9	8	-1	7	6	-1
197	24	33	11	11	0	7	8	1	6	4	-2
198	24	42	14	14	0	9	9	0	6	6	0
199	24	34	12	12	0	8	9	1	6	4	-2
200	24	41	14	14	0	10	8	-2	6	6	0
201	25	47	17	17	0	10	11	1	9	7	-2
202	25	45	18	18	0	12	12	0	8	8	0
203	25	37	11	11	0	8	8	0	5	5	0
204	25	40	13	13	0	7	9	2	6	6	0
205	25	39	12	12	0	7	10	3	6	5	-1
206	25	42	14	14	0	9	10	1	6	5	-1
207	25	43	15	15	0	10	12	2	7	6	-1
208	25	36	10	10	0	6	9	3	4	3	-1
209	25	35	10	10	0	6	8	2	5	3	-2
210	25	39	14	14	0	8	9	1	7	6	-1
211	26	38	13	13	0	8	8	0	7	5	-2
212	26	40	11	11	0	7	9	2	4	3	-1

	Vertices	Edges	Jacobi unsorted	Jacobi sorted	Jacobi difference	Gauss-Seidel unsorted	Gauss-Seidel sorted	Gauss-Seidel difference	Forward and back unsorted	Forward and back sorted	Forward and back difference
213	26	42	15	15	0	10	11	1	6	6	0
214	26	36	12	12	0	9	9	0	5	4	-1
215	26	47	13	13	0	10	8	-2	5	4	-1
216	26	41	13	13	0	8	7	-1	6	4	-2
217	26	37	13	13	0	8	10	2	6	6	0
218	26	44	16	16	0	8	9	1	6	7	1
219	26	41	11	11	0	6	10	4	5	3	-2
220	26	44	14	14	0	9	9	0	5	5	0
221	27	49	19	19	0	10	13	3	8	9	1
222	27	43	12	12	0	8	9	1	5	5	0
223	27	56	20	20	0	12	14	2	7	8	1
224	27	44	15	15	0	10	9	-1	7	5	-2
225	27	37	13	13	0	7	8	1	6	5	-1
226	27	43	15	15	0	9	10	1	6	5	-1
227	27	44	15	15	0	8	10	2	7	6	-1
228	27	46	12	12	0	7	9	2	4	4	0
229	27	51	17	17	0	11	13	2	8	6	-2
230	27	44	15	15	0	10	10	0	7	5	-2
231	28	48	14	14	0	8	9	1	6	5	-1
232	28	36	11	11	0	8	10	2	5	3	-2
233	28	52	15	15	0	10	11	1	6	5	-1
234	28	51	17	17	0	11	11	0	8	8	0
235	28	50	16	16	0	10	11	1	7	7	0
236	28	52	17	17	0	10	11	1	7	8	1
237	28	50	17	17	0	9	11	2	7	8	1
238	28	46	15	15	0	9	9	0	5	6	1
239	28	40	13	13	0	8	10	2	5	5	0
240	28	56	21	21	0	12	13	1	9	8	-1
241	29	51	16	16	0	11	10	-1	8	6	-2
242	29	45	14	14	0	10	11	1	5	4	-1
243	29	44	16	16	0	10	11	1	7	6	-1
244	29	43	15	15	0	11	9	-2	7	5	-2
245	29	47	15	15	0	9	12	3	8	6	-2
246	29	48	16	16	0	10	11	1	6	6	0
247	29	42	14	14	0	8	9	1	6	5	-1
248	29	49	15	15	0	8	9	1	7	6	-1
249	29	46	13	13	0	8	10	2	5	4	-1
250	29	49	14	14	0	8	11	3	6	6	0
251	30	55	20	20	0	13	13	0	8	8	0
252	30	50	16	16	0	9	13	4	7	6	-1
253	30	48	16	16	0	9	10	1	7	6	-1
254	30	49	18	18	0	12	12	0	8	6	-2
255	30	49	13	13	0	8	10	2	7	5	-2
256	30	53	16	16	0	8	9	1	6	6	0
257	30	55	18	18	0	12	13	1	8	6	-2
258	30	43	12	12	0	7	10	3	6	4	-2
259	30	43	14	14	0	10	10	0	5	3	-2
260	30	49	18	18	0	10	12	2	8	8	0
261	31	55	17	17	0	11	11	0	7	7	0
262	31	44	13	13	0	9	11	2	7	4	-3
263	31	48	18	18	0	12	14	2	8	6	-2
264	31	45	15	15	0	10	10	0	5	5	0
265	31	47	17	17	0	9	12	3	7	5	-2
266	31	45	15	15	0	10	11	1	7	4	-3
267	31	58	23	23	0	15	12	-3	10	9	-1
268	31	55	19	19	0	10	12	2	10	7	-3
269	31	54	19	19	0	10	12	2	7	8	1
270	31	57	23	23	0	13	15	2	11	8	-3
271	32	57	20	20	0	14	13	-1	8	8	0

	Vertices	Edges	Jacobi unsorted	Jacobi sorted	Jacobi difference	Gauss-Seidel unsorted	Gauss-Seidel sorted	Gauss-Seidel difference	Forward and back unsorted	Forward and back sorted	Forward and back difference
272	32	55	19	19	0	10	13	3	8	7	-1
273	32	55	19	19	0	13	12	-1	8	7	-1
274	32	62	21	21	0	12	12	0	9	10	1
275	32	49	15	15	0	8	12	4	6	5	-1
276	32	59	16	16	0	8	12	4	8	8	0
277	32	57	21	21	0	13	15	2	8	8	0
278	32	55	16	16	0	9	11	2	7	6	-1
279	32	50	15	15	0	9	11	2	5	4	-1
280	32	49	16	16	0	10	11	1	6	7	1
281	33	49	16	16	0	9	13	4	6	5	-1
282	33	57	22	22	0	14	15	1	9	8	-1
283	33	60	21	21	0	10	13	3	9	10	1
284	33	51	16	16	0	9	12	3	7	4	-3
285	33	53	17	17	0	11	13	2	7	5	-2
286	33	62	22	22	0	14	15	1	9	8	-1
287	33	61	22	22	0	14	12	-2	9	9	0
288	33	55	18	18	0	10	14	4	8	6	-2
289	33	55	16	16	0	10	12	2	8	7	-1
290	33	54	17	17	0	10	15	5	8	6	-2
291	34	54	16	16	0	9	13	4	6	5	-1
292	34	62	21	21	0	14	12	-2	10	8	-2
293	34	65	19	19	0	11	14	3	9	8	-1
294	34	58	19	19	0	10	13	3	8	6	-2
295	34	62	22	22	0	14	15	1	9	8	-1
296	34	59	20	20	0	10	16	6	9	6	-3
297	34	49	18	18	0	11	12	1	8	5	-3
298	34	52	20	20	0	12	14	2	7	7	0
299	34	58	20	20	0	13	14	1	9	6	-3
300	34	55	17	17	0	11	11	0	7	6	-1
301	35	59	20	20	0	13	15	2	8	8	0
302	35	57	21	21	0	12	16	4	9	7	-2
303	35	58	19	19	0	11	12	1	8	5	-3
304	35	58	17	17	0	12	14	2	6	4	-2
305	35	64	23	23	0	13	14	1	9	8	-1
306	35	57	19	19	0	12	15	3	7	5	-2
307	35	62	23	23	0	16	17	1	10	7	-3
308	35	59	23	23	0	14	15	1	10	9	-1
309	35	51	16	16	0	12	14	2	6	5	-1
310	35	53	17	17	0	11	13	2	8	5	-3
311	36	58	15	15	0	12	13	1	6	3	-3
312	36	64	23	23	0	14	14	0	8	10	2
313	36	62	23	23	0	12	14	2	11	8	-3
314	36	60	19	19	0	11	14	3	8	7	-1
315	36	58	20	20	0	13	12	-1	9	7	-2
316	36	58	21	21	0	14	17	3	8	6	-2
317	36	57	18	18	0	10	14	4	8	5	-3
318	36	59	22	22	0	11	13	2	8	8	0
319	36	57	19	19	0	14	15	1	6	6	0
320	36	68	23	23	0	14	15	1	9	9	0
321	37	62	23	23	0	14	15	1	9	9	0
322	37	63	20	20	0	11	14	3	9	7	-2
323	37	61	19	19	0	13	15	2	8	7	-1
324	37	59	22	22	0	14	16	2	9	7	-2
325	37	55	17	17	0	10	14	4	6	5	-1
326	37	57	18	18	0	13	16	3	9	3	-6
327	37	57	18	18	0	13	14	1	5	4	-1
328	37	57	18	18	0	10	14	4	7	4	-3
329	37	70	26	26	0	16	16	0	10	11	1
330	37	50	18	18	0	13	13	0	7	6	-1

	Vertices	Edges	Jacobi			Gauss-Seidel			Forward and back		
			unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference
331	38	57	19	19	0	12	14	2	8	6	-2
332	38	58	17	17	0	12	16	4	7	4	-3
333	38	60	20	20	0	12	14	2	9	6	-3
334	38	69	24	24	0	16	15	-1	9	10	1
335	38	65	20	20	0	13	15	2	8	6	-2
336	38	64	19	19	0	14	14	0	8	6	-2
337	38	61	20	20	0	14	15	1	8	7	-1
338	38	67	22	22	0	12	17	5	9	6	-3
339	38	67	25	25	0	17	15	-2	10	12	2
340	38	54	18	18	0	12	14	2	8	5	-3
341	39	65	23	23	0	14	18	4	8	7	-1
342	39	70	26	26	0	15	17	2	11	10	-1
343	39	70	23	23	0	14	17	3	9	7	-2
344	39	68	25	25	0	15	19	4	10	9	-1
345	39	68	25	25	0	16	17	1	9	8	-1
346	39	55	17	17	0	11	14	3	7	4	-3
347	39	71	26	26	0	17	17	0	10	9	-1
348	39	60	21	21	0	13	15	2	9	6	-3
349	39	69	22	22	0	16	16	0	9	8	-1
350	39	68	24	24	0	17	16	-1	9	7	-2
351	40	61	19	19	0	13	16	3	7	4	-3
352	40	65	25	25	0	15	16	1	9	9	0
353	40	67	23	23	0	13	15	2	8	7	-1
354	40	65	21	21	0	13	17	4	8	6	-2
355	40	60	23	23	0	14	18	4	10	8	-2
356	40	67	25	25	0	15	19	4	9	9	0
357	40	68	21	21	0	12	16	4	8	7	-1
358	40	62	19	19	0	14	14	0	7	4	-3
359	40	71	24	24	0	13	15	2	10	7	-3
360	40	57	17	17	0	11	14	3	8	4	-4

D.2 Calculations

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
1	125	150	90	-60	150	90	-60	90	66	-24	4
2	125	150	90	-60	100	90	-10	90	66	-24	4
3	125	150	90	-60	150	90	-60	90	66	-24	4
4	125	150	90	-60	150	90	-60	90	66	-24	4
5	125	150	90	-60	150	90	-60	90	66	-24	4
6	125	150	90	-60	150	90	-60	90	66	-24	4
7	125	150	90	-60	100	90	-10	90	66	-24	4
8	125	150	90	-60	100	90	-10	90	66	-24	4
9	125	150	90	-60	100	90	-10	90	66	-24	4
10	125	150	90	-60	100	90	-10	90	66	-24	4
11	203	288	168	-120	216	168	-48	132	92	-40	5
12	203	288	168	-120	216	168	-48	132	92	-40	5
13	203	288	168	-120	288	168	-120	198	92	-106	5
14	203	288	168	-120	216	168	-48	132	92	-40	5
15	203	288	168	-120	288	168	-120	132	92	-40	5
16	203	288	168	-120	216	168	-48	198	92	-106	5
17	203	288	168	-120	216	168	-48	198	92	-106	5
18	203	288	168	-120	288	168	-120	198	92	-106	5
19	203	288	168	-120	216	168	-48	198	92	-106	5
20	203	288	168	-120	216	168	-48	132	92	-40	5
21	308	392	224	-168	294	224	-70	273	122	-151	6
22	308	392	224	-168	294	224	-70	182	122	-60	6
23	308	392	224	-168	392	224	-168	182	122	-60	6
24	308	392	224	-168	294	224	-70	182	122	-60	6
25	308	588	516	-72	490	344	-146	273	255	-18	3
26	308	588	516	-72	490	430	-60	273	255	-18	3
27	308	392	224	-168	392	224	-168	182	122	-60	6
28	308	392	224	-168	294	224	-70	182	122	-60	6
29	308	392	224	-168	294	224	-70	182	122	-60	6
30	308	392	224	-168	392	224	-168	182	122	-60	6
31	444	768	648	-120	512	648	136	360	324	-36	4
32	444	768	648	-120	512	432	-80	360	324	-36	4
33	444	512	288	-224	512	288	-224	240	156	-84	7
34	444	512	288	-224	512	288	-224	360	156	-204	7
35	444	512	288	-224	384	288	-96	240	156	-84	7
36	444	512	288	-224	384	288	-96	240	156	-84	7
37	444	768	648	-120	512	540	28	360	324	-36	4
38	444	768	648	-120	384	324	-60	360	324	-36	4
39	444	512	288	-224	256	288	32	240	156	-84	7
40	444	768	648	-120	384	432	48	360	324	-36	4
41	615	972	792	-180	648	528	-120	459	399	-60	5
42	615	972	792	-180	648	396	-252	459	399	-60	5
43	615	972	792	-180	648	528	-120	459	532	73	5
44	615	648	360	-288	648	360	-288	459	194	-265	8
45	615	1296	1248	-48	972	1092	120	612	453	-159	2
46	615	972	792	-180	648	792	144	459	399	-60	5
47	615	972	792	-180	648	528	-120	459	399	-60	5
48	615	972	792	-180	810	792	-18	459	266	-193	5
49	615	648	360	-288	486	360	-126	459	194	-265	8
50	615	648	360	-288	486	360	-126	459	194	-265	8
51	825	1200	948	-252	800	632	-168	760	480	-280	6
52	825	1600	1504	-96	1200	1128	-72	760	736	-24	3
53	825	1200	948	-252	1000	632	-368	570	480	-90	6
54	825	1600	1504	-96	1400	1316	-84	760	552	-208	3
55	825	1200	948	-252	1000	948	-52	570	480	-90	6
56	825	1200	948	-252	800	790	-10	760	480	-280	6
57	825	1600	1504	-96	1000	940	-60	760	736	-24	3

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
58	825	1600	1504	-96	1000	940	-60	760	736	-24	3
59	825	1200	948	-252	800	948	148	570	480	-90	6
60	825	1200	948	-252	800	632	-168	570	640	70	6
61	1078	1694	1302	-392	968	1116	148	693	567	-126	7
62	1078	2178	1998	-180	1694	1554	-140	1155	876	-279	4
63	1078	2178	1998	-180	1210	1332	122	1155	1095	-60	4
64	1078	1694	1302	-392	1210	1116	-94	924	567	-357	7
65	1078	1694	1302	-392	968	1302	334	924	378	-546	7
66	1078	2178	1998	-180	1452	1554	102	924	876	-48	4
67	1078	2178	1998	-180	1694	1554	-140	1155	1095	-60	4
68	1078	2178	1998	-180	1452	1332	-120	924	876	-48	4
69	1078	1210	660	-550	968	660	-308	462	282	-180	10
70	1078	1210	660	-550	726	660	-66	462	282	-180	10
71	1378	3168	3102	-66	2304	1974	-330	1380	1370	-10	2
72	1378	2592	2322	-270	1728	2064	336	1380	768	-612	5
73	1378	2016	1512	-504	1152	1080	-72	1104	880	-224	8
74	1378	2016	1512	-504	1440	1080	-360	1104	660	-444	8
75	1378	1440	780	-660	864	780	-84	828	332	-496	11
76	1378	2016	1512	-504	1440	1296	-144	1104	660	-444	8
77	1378	2016	1512	-504	1440	1080	-360	1104	660	-444	8
78	1378	1440	780	-660	1152	780	-372	828	332	-496	11
79	1378	2016	1512	-504	1440	1080	-360	1104	660	-444	8
80	1378	2592	2322	-270	2016	1806	-210	1380	1280	-100	5
81	1729	2366	1736	-630	1690	1240	-450	1300	1012	-288	9
82	1729	1690	910	-780	1014	910	-104	975	386	-589	12
83	1729	3718	3586	-132	3042	2282	-760	1950	1595	-355	3
84	1729	3042	2664	-378	2366	2072	-294	1300	1180	-120	6
85	1729	1690	910	-780	1352	910	-442	975	386	-589	12
86	1729	2366	1736	-630	2028	1736	-292	1300	506	-794	9
87	1729	3042	2664	-378	2028	1480	-548	1300	1475	175	6
88	1729	3042	2664	-378	2366	1480	-886	1625	885	-740	6
89	1729	2366	1736	-630	1690	1488	-202	1300	759	-541	9
90	1729	4394	4394	0	3380	2704	-676	1950	1625	-325	0
91	2135	3136	2256	-880	2352	1692	-660	1512	864	-648	10
92	2135	3920	3360	-560	2744	2688	-56	1890	1344	-546	7
93	2135	3136	2256	-880	2352	1692	-660	1890	864	-1026	10
94	2135	2352	1260	-1092	2352	1260	-1092	1134	444	-690	13
95	2135	3920	3360	-560	2744	2352	-392	2268	1344	-924	7
96	2135	2352	1260	-1092	2352	1260	-1092	1134	444	-690	13
97	2135	2352	1260	-1092	1960	1260	-700	1512	444	-1068	13
98	2135	3136	2256	-880	2352	1692	-660	1134	1152	18	10
99	2135	3920	3360	-560	2744	2688	-56	1512	1344	-168	7
100	2135	4704	4464	-240	3136	2604	-532	1890	2196	306	4
101	2600	3600	2544	-1056	2250	1908	-342	1305	975	-330	11
102	2600	3600	2544	-1056	2250	1908	-342	1740	1300	-440	11
103	2600	3600	2544	-1056	2250	1908	-342	1740	975	-765	11
104	2600	5400	5040	-360	3600	3360	-240	2175	2490	315	5
105	2600	4500	3780	-720	2700	3024	324	1740	1516	-224	8
106	2600	4500	3780	-720	3150	1890	-1260	2175	1516	-659	8
107	2600	3600	2544	-1056	2250	1908	-342	1740	1300	-440	11
108	2600	4500	3780	-720	3150	3024	-126	1740	1516	-224	8
109	2600	4500	3780	-720	3600	3024	-576	2175	1137	-1038	8
110	2600	4500	3780	-720	3150	2646	-504	2175	1516	-659	8
111	3128	6144	5640	-504	4096	3760	-336	2480	2796	316	6
112	3128	4096	2848	-1248	2560	2492	-68	1984	1456	-528	12
113	3128	5120	4220	-900	3584	2954	-630	2480	1696	-784	9
114	3128	4096	2848	-1248	2560	2492	-68	1488	1092	-396	12
115	3128	4096	2848	-1248	2048	2136	88	1984	1456	-528	12
116	3128	4096	2848	-1248	3072	2492	-580	1984	1092	-892	12

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
117	3128	5120	4220	-900	4096	2954	-1142	1984	2120	136	9
118	3128	6144	5640	-504	4096	4230	134	2976	2796	-180	6
119	3128	6144	5640	-504	3584	3290	-294	2480	2796	316	6
120	3128	4096	2848	-1248	3072	2492	-580	1984	1456	-528	12
121	3723	5202	3564	-1638	2890	3168	278	2244	1215	-1029	13
122	3723	7514	6786	-728	4624	4176	-448	3366	3114	-252	7
123	3723	6358	5148	-1210	4046	3744	-302	2805	2355	-450	10
124	3723	5202	3564	-1638	3468	3564	96	1683	1215	-468	13
125	3723	6358	5148	-1210	4624	4212	-412	2805	1884	-921	10
126	3723	6358	5148	-1210	4046	4212	166	3366	2355	-1011	10
127	3723	6358	5148	-1210	4624	3744	-880	2805	2355	-450	10
128	3723	5202	3564	-1638	2890	3168	278	2244	1215	-1029	13
129	3723	4046	2142	-1904	3468	2142	-1326	1683	642	-1041	16
130	3723	6358	5148	-1210	3468	3744	276	2805	2826	21	10
131	4389	7128	5676	-1452	5184	4128	-1056	2520	2600	80	11
132	4389	9720	9270	-450	6480	6798	318	3780	3660	-120	5
133	4389	5832	3942	-1890	3888	3942	54	3150	1344	-1806	14
134	4389	5832	3942	-1890	3240	3504	264	2520	1344	-1176	14
135	4389	8424	7488	-936	5832	4608	-1224	4410	3444	-966	8
136	4389	7128	5676	-1452	4536	3612	-924	3780	2080	-1700	11
137	4389	4536	2394	-2142	3240	2394	-846	1890	716	-1174	17
138	4389	8424	7488	-936	6480	5184	-1296	3780	2870	-910	8
139	4389	5832	3942	-1890	3888	3066	-822	1890	1344	-546	14
140	4389	8424	7488	-936	6480	4608	-1872	3780	3444	-336	8
141	5130	9386	8216	-1170	7220	6952	-268	4218	3155	-1063	9
142	5130	6498	4338	-2160	2888	2892	4	2812	1479	-1333	15
143	5130	9386	8216	-1170	6498	4424	-2074	3515	3155	-360	9
144	5130	7942	6226	-1716	5054	5660	606	3515	2284	-1231	12
145	5130	7942	6226	-1716	5776	5094	-682	3515	2284	-1231	12
146	5130	10830	10200	-630	7942	6800	-1142	4921	4038	-883	6
147	5130	9386	8216	-1170	5776	5056	-720	4218	3155	-1063	9
148	5130	9386	8216	-1170	6498	6320	-178	3515	3155	-360	9
149	5130	9386	8216	-1170	6498	4424	-2074	4218	3155	-1063	9
150	5130	9386	8216	-1170	6498	5688	-810	4218	1893	-2325	9
151	5950	8800	6798	-2002	6400	4326	-2074	3900	3120	-780	13
152	5950	10400	9672	-728	7200	7440	240	4680	4428	-252	7
153	5950	7200	4752	-2448	4800	3696	-1104	3900	1620	-2280	16
154	5950	8800	6798	-2002	5600	5562	-38	3900	3120	-780	13
155	5950	9600	8280	-1320	6400	4140	-2260	3900	3450	-450	10
156	5950	11200	10416	-784	8000	6696	-1304	3900	5166	1266	7
157	5950	13600	13260	-340	8800	8580	-220	6240	6144	-96	4
158	5950	7200	4752	-2448	4800	4224	-576	3120	1620	-1500	16
159	5950	8800	6798	-2002	6400	5562	-838	3900	2496	-1404	13
160	5950	8800	6798	-2002	7200	4944	-2256	3900	3120	-780	13
161	6853	7938	5184	-2754	6174	3456	-2718	3444	1178	-2266	17
162	6853	11466	9750	-1716	7056	6750	-306	5166	4506	-660	11
163	6853	7938	5184	-2754	5292	4608	-684	4305	1767	-2538	17
164	6853	12348	11340	-1008	7938	6480	-1458	5166	5635	469	8
165	6853	6174	3234	-2940	4410	3234	-1176	2583	962	-1621	20
166	6853	9702	8250	-1452	6174	6750	576	5166	3755	-1411	11
167	6853	7938	5184	-2754	7056	4032	-3024	3444	1767	-1677	17
168	6853	10584	9720	-864	7056	7290	234	5166	4025	-1141	8
169	6853	9702	7392	-2310	7056	5376	-1680	4305	3395	-910	14
170	6853	13230	12150	-1080	8820	8910	90	6027	5635	-392	8
171	7843	12584	10556	-2028	9680	7308	-2372	5676	4070	-1606	12
172	7843	12584	10556	-2028	8712	8120	-592	6622	3256	-3366	12
173	7843	18392	18164	-228	10648	11472	824	8514	8460	-54	3
174	7843	12584	10556	-2028	8712	5684	-3028	4730	4884	154	12
175	7843	12584	10556	-2028	9680	8120	-1560	5676	4884	-792	12

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
176	7843	15488	14816	-672	8712	9260	548	5676	6412	736	6
177	7843	10648	8008	-2640	7744	8008	264	4730	2944	-1786	15
178	7843	9680	7280	-2400	6776	5096	-1680	4730	3680	-1050	15
179	7843	9680	7280	-2400	6776	5824	-952	3784	3680	-104	15
180	7843	11616	9744	-1872	7744	6496	-1248	5676	4884	-792	12
181	8924	13754	11388	-2366	8464	7884	-580	5175	3516	-1659	13
182	8924	10580	6780	-3800	7406	5424	-1982	5175	2079	-3096	19
183	8924	16928	15168	-1760	11638	9480	-2158	7245	5670	-1575	10
184	8924	10580	7860	-2720	6348	6288	-60	5175	3180	-1995	16
185	8924	16928	15168	-1760	10580	9480	-1100	7245	5670	-1575	10
186	8924	10580	6780	-3800	6348	6102	-246	4140	2079	-2061	19
187	8924	12696	9432	-3264	8464	7860	-604	6210	3975	-2235	16
188	8924	14812	12264	-2548	10580	7884	-2696	7245	5274	-1971	13
189	8924	16928	15168	-1760	10580	11376	796	7245	5670	-1575	10
190	8924	10580	6780	-3800	7406	5424	-1982	5175	2772	-2403	19
191	10100	11520	8460	-3060	8064	6768	-1296	5640	3424	-2216	17
192	10100	13824	10152	-3672	6912	6768	-144	6768	3424	-3344	17
193	10100	14976	12246	-2730	9216	9420	204	6768	5676	-1092	14
194	10100	16128	13188	-2940	11520	8478	-3042	7896	4730	-3166	14
195	10100	13824	10152	-3672	8064	6768	-1296	5640	3424	-2216	17
196	10100	14976	12246	-2730	10368	7536	-2832	7896	5676	-2220	14
197	10100	12672	9306	-3366	8064	6768	-1296	6768	3424	-3344	17
198	10100	16128	13188	-2940	10368	8478	-1890	6768	5676	-1092	14
199	10100	13824	10152	-3672	9216	7614	-1602	6768	3424	-3344	17
200	10100	16128	13188	-2940	11520	7536	-3984	6768	5676	-1092	14
201	11375	21250	19720	-1530	12500	12760	260	11025	8071	-2954	9
202	11375	22500	20880	-1620	15000	13920	-1080	9800	9224	-576	9
203	11375	13750	9988	-3762	10000	7264	-2736	6125	4595	-1530	18
204	11375	16250	13130	-3120	8750	9090	340	7350	6090	-1260	15
205	11375	15000	10896	-4104	8750	9080	330	7350	4595	-2755	18
206	11375	17500	14140	-3360	11250	10100	-1150	7350	5075	-2275	15
207	11375	18750	16410	-2340	12500	13128	628	8575	6558	-2017	12
208	11375	12500	9080	-3420	7500	8172	672	4900	2757	-2143	18
209	11375	12500	7880	-4620	7500	6304	-1196	6125	2415	-3710	21
210	11375	17500	14140	-3360	10000	9090	-910	8575	6090	-2485	15
211	12753	17576	12636	-4940	10816	7776	-3040	9282	4920	-4362	19
212	12753	14872	10692	-4180	9464	8748	-716	5304	2952	-2352	19
213	12753	20280	16200	-4080	13520	11880	-1640	7956	6516	-1440	16
214	12753	16224	11664	-4560	12168	8748	-3420	6630	3936	-2694	19
215	12753	17576	15210	-2366	13520	9360	-4160	6630	4680	-1950	13
216	12753	17576	12636	-4940	10816	6804	-4012	7956	3936	-4020	19
217	12753	17576	14040	-3536	10816	10800	-16	7956	6516	-1440	16
218	12753	21632	18720	-2912	10816	10530	-286	7956	8190	234	13
219	12753	14872	9306	-5566	8112	8460	348	6630	2592	-4038	22
220	12753	18928	15120	-3808	12168	9720	-2448	6630	5430	-1200	16
221	14238	27702	25194	-2508	14580	17238	2658	11448	11889	441	11
222	14238	17496	12456	-5040	11664	9342	-2322	7155	5255	-1900	20
223	14238	29160	27720	-1440	17496	19404	1908	10017	11000	983	8
224	14238	21870	17280	-4590	14580	10368	-4212	10017	5795	-4222	17
225	14238	18954	13494	-5460	10206	8304	-1902	8586	5255	-3331	20
226	14238	21870	17280	-4590	13122	11520	-1602	8586	5795	-2791	17
227	14238	21870	17280	-4590	11664	11520	-144	10017	6954	-3063	17
228	14238	17496	13824	-3672	10206	10368	162	5724	4636	-1088	17
229	14238	24786	22542	-2244	16038	17238	1200	11448	7926	-3522	11
230	14238	21870	18720	-3150	14580	12480	-2100	10017	6245	-3772	14
231	15834	21952	17164	-4788	12544	11034	-1510	9240	6170	-3070	18
232	15834	17248	10648	-6600	12544	9680	-2864	7700	2964	-4736	24
233	15834	23520	18390	-5130	15680	13486	-2194	9240	6170	-3070	18
234	15834	26656	22576	-4080	17248	14608	-2640	12320	10640	-1680	15

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
235	15834	25088	21248	-3840	15680	14608	-1072	10780	9310	-1470	15
236	15834	26656	24004	-2652	15680	15532	-148	10780	11264	484	12
237	15834	26656	22576	-4080	14112	14608	496	10780	10640	-140	15
238	15834	23520	18390	-5130	14112	11034	-3078	7700	7404	-296	18
239	15834	20384	14378	-6006	12544	11060	-1484	7700	5600	-2100	21
240	15834	32928	31038	-1890	18816	19214	398	13860	11744	-2116	9
241	17545	26912	20832	-6080	18502	13020	-5482	13224	7866	-5358	19
242	17545	23548	16464	-7084	16820	12936	-3884	8265	4764	-3501	22
243	17545	26912	20832	-6080	16820	14322	-2498	11571	7866	-3705	19
244	17545	25230	19530	-5700	18502	11718	-6784	11571	6555	-5016	19
245	17545	25230	19530	-5700	15138	15624	486	13224	7866	-5358	19
246	17545	26912	20832	-6080	16820	14322	-2498	9918	7866	-2052	19
247	17545	23548	16464	-7084	13456	10584	-2872	9918	5955	-3963	22
248	17545	25230	19530	-5700	13456	11718	-1738	11571	7866	-3705	19
249	17545	21866	15288	-6578	13456	11760	-1696	8265	4764	-3501	22
250	17545	23548	16464	-7084	13456	12936	-520	9918	7146	-2772	22
251	19375	36000	31800	-4200	23400	20670	-2730	14160	12704	-1456	14
252	19375	28800	22080	-6720	16200	17940	1740	12390	8340	-4050	20
253	19375	28800	22080	-6720	16200	13800	-2400	12390	8340	-4050	20
254	19375	32400	26892	-5508	21600	17928	-3672	14160	8988	-5172	17
255	19375	23400	17940	-5460	14400	13800	-600	12390	6950	-5440	20
256	19375	28800	22080	-6720	14400	12420	-1980	10620	8340	-2280	20
257	19375	32400	26892	-5508	21600	19422	-2178	14160	8988	-5172	17
258	19375	21600	14976	-6624	12600	12480	-120	10620	5056	-5564	23
259	19375	25200	17472	-7728	18000	12480	-5520	8850	3792	-5058	23
260	19375	32400	26892	-5508	18000	17928	-72	14160	11984	-2176	17
261	21328	32674	26860	-5814	21142	17380	-3762	13237	11095	-2142	18
262	21328	24986	15158	-9828	17298	12826	-4472	13237	4756	-8481	27
263	21328	34596	28440	-6156	23064	22120	-944	15128	9510	-5618	18
264	21328	28830	19830	-9000	19220	13220	-6000	9455	6695	-2760	24
265	21328	32674	24820	-7854	17298	17520	222	13237	7355	-5882	21
266	21328	28830	19830	-9000	19220	14542	-4678	13237	5356	-7881	24
267	21328	44206	40618	-3588	28830	21192	-7638	18910	15831	-3079	12
268	21328	36518	31958	-4560	19220	20184	964	18910	11767	-7143	15
269	21328	36518	30020	-6498	19220	18960	-260	13237	12680	-557	18
270	21328	44206	40618	-3588	24986	26490	1504	20801	14072	-6729	12
271	23408	40960	35520	-5440	28672	23088	-5584	16128	14208	-1920	16
272	23408	38912	31692	-7220	20480	21684	1204	16128	11718	-4410	19
273	23408	38912	31692	-7220	26624	20016	-6608	16128	11718	-4410	19
274	23408	43008	37296	-5712	24576	21312	-3264	18144	17760	-384	16
275	23408	30720	23130	-7590	16384	18504	2120	12096	7770	-4326	22
276	23408	32768	26688	-6080	16384	20016	3632	16128	13392	-2736	19
277	23408	43008	37296	-5712	26624	26640	16	16128	14208	-1920	16
278	23408	32768	26688	-6080	18432	18348	-84	14112	10044	-4068	19
279	23408	30720	20970	-9750	18432	15378	-3054	10080	5664	-4416	25
280	23408	32768	24672	-8096	20480	16962	-3518	12096	10878	-1218	22
281	25619	34848	26016	-8832	19602	21138	1536	12870	8195	-4675	23
282	25619	47916	41184	-6732	30492	28080	-2412	19305	14984	-4321	17
283	25619	45738	39312	-6426	21780	24336	2556	19305	18730	-575	17
284	25619	34848	23616	-11232	19602	17712	-1890	15015	5980	-9035	26
285	25619	37026	27642	-9384	23958	21138	-2820	15015	8195	-6820	23
286	25619	47916	41184	-6732	30492	28080	-2412	19305	14984	-4321	17
287	25619	47916	41184	-6732	30492	22464	-8028	19305	16857	-2448	17
288	25619	39204	29268	-9936	21780	22764	984	17160	9834	-7326	23
289	25619	34848	28128	-6720	21780	21096	-684	17160	12355	-4805	20
290	25619	37026	27642	-9384	21780	24390	2610	17160	9834	-7326	23
291	27965	36992	24896	-12096	20808	20228	-580	13668	7880	-5788	27
292	27965	48552	41370	-7182	32368	23640	-8728	22780	15776	-7004	18
293	27965	43928	37430	-6498	25432	27580	2148	20502	15776	-4726	18

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
294	27965	43928	35150	-8778	23120	24050	930	18224	11148	-7076	21
295	27965	50864	43340	-7524	32368	29550	-2818	20502	15776	-4726	18
296	27965	46240	37000	-9240	23120	29600	6480	20502	11148	-9354	21
297	27965	41616	30816	-10800	25432	20544	-4888	18224	8630	-9594	24
298	27965	46240	37000	-9240	27744	25900	-1844	15946	13006	-2940	21
299	27965	46240	39400	-6840	30056	27580	-2476	20502	11832	-8670	18
300	27965	39304	29104	-10200	25432	18832	-6600	15946	10356	-5590	24
301	30450	49000	38880	-10120	31850	29160	-2690	19320	15624	-3696	22
302	30450	51450	43470	-7980	29400	33120	3720	21735	14511	-7224	19
303	30450	46550	34200	-12350	26950	21600	-5350	19320	9075	-10245	25
304	30450	41650	27846	-13804	29400	22932	-6468	14490	6636	-7854	28
305	30450	56350	47610	-8740	31850	28980	-2870	21735	16584	-5151	19
306	30450	46550	36936	-9614	29400	29160	-240	16905	9765	-7140	22
307	30450	56350	47610	-8740	39200	35190	-4010	24150	14511	-9639	19
308	30450	56350	47610	-8740	34300	31050	-3250	24150	18657	-5493	19
309	30450	39200	26208	-12992	29400	22932	-6468	14490	8295	-6195	28
310	30450	41650	27846	-13804	26950	21294	-5656	19320	8295	-11025	28
311	33078	38880	25830	-13050	31104	22386	-8718	15336	5232	-10104	29
312	33078	59616	49956	-9660	36288	30408	-5880	20448	21760	1312	20
313	33078	59616	49956	-9660	31104	30408	-696	28116	17408	-10708	20
314	33078	49248	35910	-13338	28512	26460	-2052	20448	13342	-7106	26
315	33078	51840	40800	-11040	33696	24480	-9216	23004	14350	-8654	23
316	33078	54432	42840	-11592	36288	34680	-1608	20448	12300	-8148	23
317	33078	46656	34020	-12636	25920	26460	540	20448	9530	-10918	26
318	33078	57024	47784	-9240	28512	28236	-276	20448	17408	-3040	20
319	33078	49248	35910	-13338	36288	28350	-7938	15336	11436	-3900	26
320	33078	59616	52578	-7038	36288	34290	-1998	23004	20556	-2448	17
321	35853	62974	52348	-10626	38332	34140	-4192	24309	20529	-3780	21
322	35853	54760	39640	-15120	30118	27748	-2370	24309	13993	-10316	27
323	35853	52022	37658	-14364	35594	29730	-5864	21608	13993	-7615	27
324	35853	60236	47036	-13200	38332	34208	-4124	24309	15043	-9266	24
325	35853	46546	30736	-15810	27380	25312	-2068	16206	9155	-7051	30
326	35853	49284	32544	-16740	35594	28928	-6666	24309	5493	-18816	30
327	35853	49284	32544	-16740	35594	25312	-10282	13505	7324	-6181	30
328	35853	49284	32544	-16740	27380	25312	-2068	18907	7324	-11583	30
329	35853	71188	64948	-6240	43808	39968	-3840	27010	27401	391	15
330	35853	49284	32544	-16740	35594	23504	-12090	18907	10986	-7921	30
331	38779	54872	39444	-15428	34656	29064	-5592	22800	12564	-10236	28
332	38779	49096	32232	-16864	34656	30336	-4320	19950	7680	-12270	31
333	38779	57760	41520	-16240	34656	29064	-5592	25650	12564	-13086	28
334	38779	69312	57168	-12144	46208	35730	-10478	25650	23880	-1770	22
335	38779	57760	44760	-13000	37544	33570	-3974	22800	13500	-9300	25
336	38779	54872	39444	-15428	40432	29064	-11368	22800	12564	-10236	28
337	38779	57760	41520	-16240	40432	31140	-9292	22800	14658	-8142	28
338	38779	63536	49236	-14300	34656	38046	3390	25650	13500	-12150	25
339	38779	72200	62700	-9500	49096	37620	-11476	28500	30096	1596	19
340	38779	51984	34128	-17856	34656	26544	-8112	22800	9600	-13200	31
341	41860	69966	53820	-16146	42588	42120	-468	24024	16471	-7553	26
342	41860	79092	68172	-10920	45630	44574	-1056	33033	26230	-6803	20
343	41860	69966	53820	-16146	42588	39780	-2808	27027	16471	-10556	26
344	41860	76050	62250	-13800	45630	47310	1680	30030	22473	-7557	23
345	41860	76050	62250	-13800	48672	42330	-6342	27027	19976	-7051	23
346	41860	51714	30294	-21420	33462	24948	-8514	21021	7252	-13769	35
347	41860	79092	68172	-10920	51714	44574	-7140	30030	23607	-6423	20
348	41860	63882	45612	-18270	39546	32580	-6966	27027	13146	-13881	29
349	41860	66924	51480	-15444	48672	37440	-11232	27027	18824	-8203	26
350	41860	73008	59760	-13248	51714	39840	-11874	27027	17479	-9548	23
351	45100	60800	39482	-21318	41600	33248	-8352	22120	8416	-13704	33
352	45100	80000	65000	-15000	48000	41600	-6400	28440	23472	-4968	24

	Gauss	Jacobi			Gauss-Seidel			Forward and back			Rows
		unsorted	sorted	difference	unsorted	sorted	difference	unsorted	sorted	difference	
353	45100	73600	56212	-17388	41600	36660	-4940	25280	17206	-8074	27
354	45100	67200	47670	-19530	41600	38590	-3010	25280	13740	-11540	30
355	45100	73600	56212	-17388	44800	43992	-808	31600	19664	-11936	27
356	45100	80000	65000	-15000	48000	49400	1400	28440	23472	-4968	24
357	45100	67200	51324	-15876	38400	39104	704	25280	17206	-8074	27
358	45100	60800	43130	-17670	44800	31780	-13020	22120	9160	-12960	30
359	45100	76800	62400	-14400	41600	39000	-2600	31600	18256	-13344	24
360	45100	54400	31756	-22644	35200	26152	-9048	25280	7600	-17680	36

D.3 Durations

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
1	0	10	0	0	0	0	0
2	0	10	0	0	0	0	10
3	10	0	0	0	0	0	10
4	0	0	0	10	0	0	0
5	0	10	0	0	10	0	0
6	10	0	10	0	0	10	0
7	10	0	10	0	0	10	0
8	10	0	0	0	0	10	0
9	0	0	0	10	0	0	0
10	0	10	0	0	0	0	10
11	0	10	0	0	10	10	0
12	10	10	20	0	0	10	10
13	10	0	10	10	0	0	0
14	10	0	10	10	0	0	0
15	10	0	0	0	0	0	0
16	10	0	0	10	0	0	10
17	10	10	0	0	10	10	0
18	10	0	0	10	10	0	0
19	10	10	0	10	0	10	0
20	10	0	10	10	0	0	0
21	10	10	0	0	10	10	0
22	10	10	0	0	0	10	0
23	10	10	10	10	0	0	0
24	10	10	0	0	0	10	0
25	10	20	10	10	10	0	10
26	10	10	0	40	10	0	10
27	10	0	0	10	10	10	0
28	10	0	10	20	0	0	0
29	20	0	10	10	0	10	0
30	10	0	0	10	0	10	0
31	10	20	10	10	0	10	0
32	30	30	40	0	0	10	0
33	10	0	10	10	0	10	40
34	11	10	0	10	0	0	30
35	10	0	10	10	0	0	0
36	10	10	0	10	0	0	10
37	10	20	10	10	0	0	10
38	10	20	20	10	0	10	10
39	0	10	0	10	10	0	0
40	10	10	40	10	0	0	0
41	10	30	0	10	20	0	0
42	20	20	0	10	10	20	0
43	20	20	0	0	0	30	10
44	10	10	10	0	0	10	0
45	20	40	10	20	10	10	0
46	10	30	10	10	10	20	0
47	10	40	10	10	0	10	0
48	20	20	0	0	10	20	0
49	10	20	20	10	0	0	0
50	10	10	10	10	0	0	10
51	20	30	11	10	10	0	0
52	20	40	10	30	0	10	10
53	20	10	10	41	10	10	0
54	40	40	10	10	10	10	0
55	10	20	0	30	10	0	0
56	20	20	10	20	0	10	0
57	30	30	10	10	10	10	0

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
58	30	30	10	10	0	10	10
59	20	30	10	20	0	10	11
60	20	10	0	20	10	0	0
61	40	30	0	10	10	10	0
62	30	30	0	10	20	10	0
63	30	30	10	10	0	10	10
64	20	50	0	10	20	0	0
65	30	40	10	20	0	10	10
66	30	60	10	10	0	10	10
67	40	60	10	20	10	10	0
68	30	40	10	10	10	10	0
69	20	10	0	10	0	0	0
70	20	10	10	10	0	10	0
71	80	50	60	20	10	10	20
72	30	60	10	10	10	10	20
73	40	30	10	10	0	10	0
74	30	30	10	10	11	10	0
75	20	10	0	10	20	20	0
76	30	40	0	10	10	10	0
77	20	20	10	10	10	10	10
78	20	10	10	20	0	0	0
79	30	20	0	30	10	0	0
80	50	50	10	30	20	10	0
81	20	40	0	10	10	10	0
82	20	20	0	0	10	0	0
83	60	50	20	20	20	10	10
84	30	60	10	10	20	10	0
85	20	20	0	0	0	0	0
86	30	30	10	10	10	10	10
87	50	40	20	10	0	0	0
88	60	30	10	10	0	10	10
89	30	40	0	10	10	0	0
90	61	60	40	60	30	40	20
91	20	40	10	10	10	10	0
92	50	40	10	20	30	10	10
93	40	50	20	10	0	10	20
94	30	30	10	0	0	11	10
95	40	50	20	10	10	10	10
96	20	20	0	20	10	10	10
97	40	20	10	10	0	10	10
98	30	40	10	10	10	0	20
99	51	60	20	10	20	10	10
100	70	50	20	20	20	20	20
101	30	40	10	10	10	0	10
102	30	60	10	10	30	10	0
103	30	40	10	10	20	10	0
104	90	50	50	30	60	20	31
105	60	30	20	20	20	0	0
106	40	40	20	20	10	10	10
107	30	50	10	10	20	11	0
108	40	60	20	20	20	10	10
109	40	50	10	30	20	10	10
110	60	30	20	20	20	0	10
111	70	50	40	30	20	10	20
112	50	40	20	10	10	0	11
113	70	40	20	10	20	20	10
114	71	20	20	10	30	20	10
115	40	20	20	10	10	10	10
116	50	40	10	10	21	10	10

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
117	40	40	20	20	30	0	10
118	80	40	30	30	20	10	40
119	90	40	30	30	30	10	20
120	50	40	20	10	10	0	10
121	80	40	20	10	20	11	10
122	140	110	100	80	80	61	120
123	101	50	30	30	30	30	20
124	100	61	30	20	30	10	0
125	90	60	30	30	20	20	20
126	120	90	40	30	30	40	20
127	100	40	20	30	20	10	20
128	70	30	40	21	20	10	10
129	90	30	30	20	10	10	20
130	90	50	50	30	30	20	20
131	100	80	50	50	40	10	20
132	120	140	110	100	110	71	80
133	91	30	30	20	20	10	10
134	100	60	21	20	30	20	10
135	100	70	50	50	50	30	30
136	90	30	50	30	20	50	20
137	70	10	30	20	20	10	0
138	120	120	80	111	70	40	40
139	100	31	20	10	20	10	0
140	130	140	80	71	40	40	40
141	150	70	50	60	60	60	30
142	101	40	20	10	50	30	10
143	120	70	40	60	40	20	30
144	110	90	40	30	30	21	20
145	110	50	50	40	20	20	30
146	150	220	150	160	130	111	100
147	110	90	70	60	50	40	40
148	131	70	50	50	60	70	30
149	90	60	40	50	20	20	40
150	120	60	60	51	50	30	20
151	81	40	20	20	20	10	20
152	110	120	80	100	100	61	90
153	70	30	10	10	20	10	10
154	80	40	20	20	20	10	10
155	90	60	40	40	30	30	30
156	110	71	50	60	90	40	50
157	120	190	180	161	180	130	131
158	70	30	20	30	20	10	10
159	80	40	20	20	20	40	30
160	80	40	31	30	20	10	10
161	90	41	20	30	20	10	10
162	150	80	60	50	50	41	40
163	100	40	41	20	20	50	20
164	170	120	110	111	100	80	100
165	80	50	20	10	20	20	0
166	110	81	30	60	30	20	20
167	100	30	10	40	30	10	10
168	120	70	41	50	30	30	30
169	90	40	30	40	40	10	10
170	170	130	100	90	91	60	60
171	130	80	50	50	40	50	30
172	150	120	100	100	141	90	70
173	250	1072	972	701	801	781	801
174	130	70	40	70	31	40	40
175	160	120	120	100	120	71	60

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
176	150	190	180	190	120	111	110
177	130	60	20	30	30	20	20
178	100	80	20	50	20	20	21
179	120	50	30	30	20	20	30
180	130	140	70	50	40	50	40
181	140	70	40	40	21	30	30
182	60	40	10	20	20	30	20
183	170	270	200	181	130	140	121
184	90	71	30	20	40	20	20
185	190	241	201	170	160	210	130
186	101	40	20	30	20	20	20
187	120	101	40	60	40	30	30
188	150	81	50	50	40	40	30
189	200	381	320	320	271	231	180
190	101	60	30	30	20	20	30
191	100	60	21	10	30	50	40
192	100	50	20	20	30	20	20
193	120	101	70	70	60	50	50
194	120	100	60	70	71	60	30
195	140	70	50	51	40	40	30
196	180	101	90	100	90	90	70
197	110	60	40	30	30	40	20
198	131	160	100	80	60	50	60
199	110	70	30	30	40	40	20
200	170	130	101	90	60	60	60
201	221	480	441	361	411	400	280
202	230	491	441	381	381	330	310
203	100	71	40	30	30	40	20
204	171	110	60	50	60	50	50
205	101	80	40	30	40	40	20
206	161	130	70	80	70	50	50
207	140	140	121	110	120	80	60
208	110	50	20	30	21	10	20
209	130	81	20	20	30	40	20
210	140	110	70	61	60	60	50
211	90	80	30	40	20	30	20
212	110	71	40	30	30	20	30
213	160	211	140	130	110	80	71
214	110	80	40	40	30	20	31
215	140	110	50	70	50	20	21
216	100	70	40	40	20	20	30
217	111	120	60	70	50	40	40
218	170	231	170	140	120	90	111
219	120	60	30	30	51	30	10
220	140	160	100	90	70	51	60
221	241	651	581	410	510	361	431
222	111	80	40	50	40	30	40
223	451	2974	2854	2434	2484	1211	1682
224	140	130	101	140	70	80	60
225	100	81	50	40	20	30	30
226	161	110	50	50	50	60	30
227	171	160	181	100	110	90	80
228	140	70	50	41	30	20	40
229	230	481	431	360	420	321	251
230	130	120	70	81	90	50	70
231	200	200	180	150	130	111	80
232	110	60	40	40	30	30	20
233	200	130	101	110	100	70	50
234	271	881	741	581	681	561	591

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
235	240	360	291	281	280	200	210
236	220	491	401	320	330	271	290
237	180	281	241	250	180	140	180
238	150	110	120	80	81	30	50
239	140	80	50	61	40	30	30
240	331	1953	1743	1171	1482	1182	971
241	200	241	201	200	150	140	120
242	130	81	50	60	30	20	20
243	251	210	171	170	150	140	120
244	130	130	110	100	91	50	40
245	180	150	140	121	140	110	80
246	180	200	170	160	130	151	90
247	150	101	50	50	40	60	40
248	161	130	120	80	110	70	61
249	160	110	90	100	120	41	60
250	171	160	110	90	160	80	91
251	220	801	751	601	621	431	621
252	131	140	160	90	121	80	70
253	160	160	200	131	130	120	151
254	151	240	210	260	140	161	130
255	120	80	60	70	70	60	50
256	130	140	90	90	70	50	71
257	170	531	471	480	400	351	271
258	90	50	20	20	30	20	10
259	90	120	50	80	50	30	30
260	130	310	280	191	231	220	190
261	190	270	230	221	171	170	160
262	141	80	40	70	50	30	30
263	291	460	361	311	360	280	201
264	120	101	50	70	40	30	40
265	171	140	110	100	110	60	60
266	130	80	50	60	30	30	20
267	2032	13360	12889	11616	8823	10245	8432
268	311	1252	1252	751	1111	1372	722
269	261	931	982	651	701	481	1101
270	1392	12508	11526	8952	9995	8162	5247
271	480	2013	2223	1953	1853	2303	1392
272	291	641	591	611	491	370	431
273	421	1702	1593	1583	1271	851	892
274	411	1722	1582	1112	1112	1121	1201
275	170	221	100	90	90	60	50
276	441	1031	941	701	992	821	881
277	431	1993	1832	1542	1673	971	1031
278	250	511	411	331	350	310	251
279	181	160	60	110	70	40	30
280	180	210	110	130	110	70	90
281	271	160	110	120	111	80	80
282	1152	6469	5838	5137	5067	4196	2995
283	1061	4176	3234	2123	2404	2704	2323
284	210	191	150	130	110	100	81
285	270	611	581	471	510	360	301
286	1042	7340	10045	7391	8321	4807	4136
287	1222	5608	4306	5247	3245	2674	2874
288	561	671	781	791	551	411	300
289	260	601	741	711	681	471	421
290	260	891	641	431	671	801	401
291	311	350	280	261	270	440	141
292	721	7060	5728	5789	3636	4586	3034
293	441	2684	2314	1432	2023	2133	1532

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
294	791	1392	1232	1192	1352	701	871
295	801	3175	2694	2864	2503	1813	1232
296	391	1332	591	500	651	431	330
297	280	281	490	490	240	171	140
298	731	881	521	380	320	241	221
299	631	2003	1872	1412	1142	1181	641
300	210	631	210	460	211	221	140
301	651	1613	1712	1191	2424	922	991
302	351	1312	801	571	711	931	461
303	450	812	501	621	380	310	211
304	321	801	541	931	420	231	221
305	1021	5168	3635	4165	4467	2053	1823
306	621	1052	641	561	521	410	260
307	4135	29733	24956	24265	19558	21551	9323
308	561	2844	2254	1943	1852	1612	1472
309	220	160	140	161	110	100	60
310	241	200	180	140	150	120	70
311	240	151	140	160	100	80	70
312	1052	10625	9534	8803	5207	4476	8373
313	1582	7971	7571	5568	6289	5979	3585
314	330	1642	1131	1242	922	771	711
315	450	2053	1202	1312	711	1092	711
316	861	1072	1152	1302	1011	841	631
317	400	1472	872	962	881	781	320
318	481	2023	1532	1472	801	1162	822
319	461	1331	1252	1242	1252	591	651
320	1102	8472	9534	5598	7831	4827	6780
321	1382	10155	8372	6048	5868	4547	5618
322	661	2443	1673	1342	1452	1262	981
323	621	961	511	601	530	361	451
324	992	1842	1852	2123	1753	1202	921
325	641	631	250	350	250	130	201
326	381	1161	831	872	1082	731	260
327	290	1172	540	571	481	250	170
328	310	1142	501	391	541	350	180
329	15592	113073	105491	85122	82239	55851	56421
330	290	631	260	631	200	160	140
331	280	451	431	421	310	691	301
332	610	802	380	420	471	251	170
333	380	1051	981	761	721	811	561
334	1011	7181	6560	6028	4726	3245	4517
335	501	2433	2183	1773	2294	1923	1091
336	581	771	571	641	601	360	331
337	1102	3735	2674	3125	2433	1662	1963
338	911	2173	1833	1252	2293	951	821
339	3645	26168	24355	20790	16324	15352	19338
340	240	281	210	240	160	140	121
341	641	3395	3104	2203	2914	1783	1693
342	1352	10575	10335	7190	7561	6880	6840
343	2043	16604	15082	11847	13209	8913	9223
344	1892	15983	16174	12408	15772	10305	9454
345	2994	22943	27049	22072	22732	13549	11828
346	841	671	461	611	611	641	260
347	6039	48390	40949	35531	36312	25026	25116
348	481	851	651	761	491	601	411
349	931	5788	5267	5308	4186	3515	3966
350	611	5238	4807	5197	3825	2784	2173
351	280	501	461	380	420	311	151
352	811	5077	4807	3866	3675	3114	2714

	Gauss	Jacobi		Gauss-Seidel		Forward and back	
		unsorted	sorted	unsorted	sorted	unsorted	sorted
353	241	530	481	341	361	230	270
354	511	1532	1882	1302	1713	981	911
355	200	491	390	480	371	291	260
356	411	3364	4016	2794	3956	1743	2373
357	220	751	721	501	691	511	411
358	190	221	181	180	140	100	70
359	521	3805	4616	3505	3255	3195	2033
360	220	331	281	250	270	220	130